NONLINEAR IDENTIFICATION WITH NEURAL NETWORKS AND FUZZY LOGIC

Jürgen Van Gorp

Promotoren: Prof. Dr. ir. J. Schoukens
Prof. Dr. ir. R. Pintelon

Proefschrift ingediend tot het behalen van de academische graad van doctor in de toegepaste wetenschappen.

April 2000
NONLINEAR IDENTIFICATION WITH NEURAL NETWORKS AND FUZZY LOGIC

Jürgen Van Gorp

Dit proefschrift, ingediend tot het behalen van de academische graad van doctor in de toegepaste wetenschappen, wordt op 1 september 2000 om 15:00 uur aan de Vrije Universiteit Brussel verdedigd voor een jury die als volgt is samengesteld:

Voorzitter: Prof. G. Maggetto (Vrije Universiteit Brussel)
Promotoren: Prof. J. Schoukens (Vrije Universiteit Brussel)
            Prof. R. Pintelon (Vrije Universiteit Brussel)
Leden: Prof. A. Barel (Vrije Universiteit Brussel)
        Prof. Gábor Horváth (Budapest University of Technology and Economics)
        Prof. Patrick Kool (Vrije Universiteit Brussel)
        Prof. Yves Rolain (Vrije Universiteit Brussel)
        Prof. Jonas Sjöberg (Chalmers University of Technology, Göteborg)
        Prof. Johan Suykens (Katholieke Universiteit Leuven)
The ideas presented in this thesis were developed and evaluated in collaboration with Johan Schoukens, Rik Pintelon and many co-workers in the department of Fundamental Electricity and Instrumentation (ELEC) of the Vrije Universiteit van Brussel (VUB). There is no doubt in my mind that this work wouldn’t have been finished without their helpfull comments and ideas, or without the encouragement I got from the people in ELEC.

Therefore, my gratitude goes to Prof. Alain Barel and Prof. Leo Van Biesen, because they gave me the opportunity to start this PhD thesis at the VUB.

I’m very much obliged to Prof. Johan Schoukens and Prof. Rik Pintelon for being the fathers of many of the ideas that can be found in this thesis. They also took the time and effort to act as proof readers. Johan, thanks also for putting me on the right path on the right moment.

I also like to thank Prof. Yves Rolain for giving me the encouragement to go on, just when I needed it the most.
CONTENTS

Preface v

Contents vii

Glossary xiii

Chapter 1
Introduction 1
1.1 Introduction 2
1.2 Thesis Overview 2
1.3 Contributions to the field of Black Box modeling 7

Chapter 2
A Scheme For Nonlinear modeling 11
2.1 Models considered in this thesis 12
2.2 Different modeling theories brought together 12
2.3 A scheme for linear and nonlinear modeling 13
2.4 Design a good experiment 15
2.5 Selection of a model structure 17
  2.5.1 Overview of a few model classes 18
  2.5.2 Choosing the model 20
  2.5.3 Model order selection 22
  2.5.4 Pruning 25
2.6 The need of carefully choosing the cost function 26
  2.6.1 The LS and WLS cost functions 26
  2.6.2 Cost functions based on probability 27
  2.6.3 Regularization 30
  2.6.4 Stability and robustness terms 33
Chapter 3
Neural Network systems

3.1 The neural network basic model
3.1.1 Introduction
3.1.2 The McCulloh-Pitts neuron
3.1.3 Hebb nets [63] or Rosenblatt's perceptron [86]
3.1.4 The Adaline and Madaline models
3.1.5 The Multilayer Perceptron model
3.1.6 Radial Basis Function NN
3.1.7 Recurrent NN models
3.1.8 Kohonen networks
3.1.9 Other architectures

3.2 Neural Networks for identification of nonlinear dynamic systems

3.3 Neural Networks for control
3.3.1 Introduction
3.3.2 Some nonlinear control schemes

3.4 NN parameter optimization
3.4.1 Supervised vs. unsupervised learning
3.4.2 Batch optimization vs. incremental optimization
3.4.3 Optimization of recurrent networks

Chapter 4
Fuzzy logic systems

4.1 Introduction

4.2 The Fuzzy Logic basic model
4.2.1 Introduction
4.2.2 Linguistic values and rules
4.2.3 Fuzzy Sets
4.2.4 Fuzzy Hedges
4.2.5 Fuzzy operators
4.2.6 t-norms and s-norms
4.2.7 Fuzzy Inference
4.2.8 Defuzzification
4.2.9  Mamdani fuzzy logic systems  88
4.2.10 Takagi-Sugeno fuzzy logic systems  88
4.2.11 Singleton Fuzzy Systems  90
4.2.12 Neurofuzzy systems  90

4.3  Fuzzy Logic models for nonlinear identification  91
4.3.1  Introduction  91
4.3.2  Choosing the model parameters  92
4.3.3  Initialization of Fuzzy Systems  94
4.3.4  FLS parameter optimization  99

4.4  Fuzzy Logic for control  100
4.4.1  Introduction  100
4.4.2  Fuzzy Supervised Linear Control  100
4.4.3  Fuzzy Supervised Fuzzy Control  102

4.5  A scheme for building FL systems  103
4.5.1  Introduction  103
4.5.2  Basic assumptions  103
4.5.3  The modeling scheme  107

4.6  Conclusions  108

Chapter 5
NLq theory with application on Fuzzy Logic Systems  109

5.1  Introduction  110

5.2  NLq theory for stability analysis  110
5.2.1  Definition of NLq systems  110
5.2.2  NLq stability theorems  112

5.3  Representation of SISO Fuzzy Logic Control systems within NLq theory  114
5.3.1  Introduction  114
5.3.2  Formal representation of a FAS with trapezoidal shaped Fuzzy Sets  115
5.3.3  Fuzzy Systems as NLq’s  118
5.3.4  Application  120

5.4  Learning SISO Fuzzy Systems with guaranteed NLq stability  124
5.4.1  Introduction  124
5.4.2  Illustrative example on a linear system  125
5.4.3  Effects of conservatism on the condition number product  128

5.5  NLqp theory  129
5.5.1  Definition of NLqp systems  129
5.5.2  Global asymptotic stability of autonomous NLqp systems  131
5.5.3  I/O stability of NLqp systems with an exogeneous input  133

5.6  Representation of MIMO Fuzzy Logic Control systems within NLqp theory  136
5.6.1  Formal representation of a MIMO Fuzzy Additive System  136
5.6.2  Example  138
Chapter 6
The EIV cost function for nonlinear systems

6.1 Introduction 144
6.2 Problem statement 144
6.3 The Least-Squares cost function for NN 146
6.4 The Errors-In-Variables cost function for NN 149
6.5 Condition relaxation on the variances 153
6.6 EIV cost function learning algorithms for large data sets (M large) 154
  6.6.1 Gradient methods (Backpropagation) 154
  6.6.2 Levenberg-Marquardt 156
6.7 EIV cost function learning algorithms for small data sets (M small) 157
  6.7.1 Gradient methods (Backpropagation) 157
  6.7.2 Levenberg-Marquardt 158
6.8 The need for early stopping 159
6.9 Modeling scheme 161
6.10 Simulation results 162
  6.10.1 Curve Fitting 162
  6.10.2 Circle fitting 164
  6.10.3 Plant modeling 171
  6.10.4 Plane mapping 173
6.11 Reduced Models 175
6.12 Conclusions 175

Chapter 7
On generalization by adding noise 177

7.1 Introduction 178
  7.1.1 Chapter outline 178
  7.1.2 About generalization 178
  7.1.3 Improving generalization using jitter 183
  7.1.4 The bias/variance trade-off 184
7.2 Analysis of the effects of adding noise on measurement data 185
7.3 The relation with regularization 189
7.4 An alternative for adding noise 190
7.5 The MIMO case 193
7.6 Conclusions 196

Chapter 8
Application 197

8.1 Steel plant analysis and optimization 198
  8.1.1 Introduction 198
A.19 Proof of Theorem 7.5 on page 195: effects of input noise on BB modeling, MIMO case 251
A.20 Proof of Theorem 7.6 on page 196: effects of input and output noise on BB modeling, MIMO case 252

Index 255
Notations

- $0_{m \times n}$ Zeros matrix of size $m \times n$.
- $1_{m \times n}$ Matrix filled with ones and of size $m \times n$.
- $A^{-1}$ Inverse of the square matrix $A$.
- $a$ Scalar notation, $a \in \mathbb{R}$.
- $a$ Vector notation, $a \in \mathbb{R}^{n \times 1}$ or $a \in \mathbb{R}^{1 \times n}$.
- $A$ Matrix notation $A \in \mathbb{R}^{n \times m}$.
- $A > 0$ denotes that $A$ is positive definite.
- $A < 0$ denotes that $A$ is negative definite.
- $A > B$ denotes that $A - B$ is positive definite.
- $[a_1 ; a_2 ; \ldots ; a_N]$ column vector of size $N \times 1$ (Matlab notation).
- $[a_1, a_2, \ldots, a_N]$ row vector of size $1 \times N$.
- $C$ Cost Function.
- $C^{(k)}$ Present cost at step $k$ in an iterative scheme.
- $d(t)$ Reference signal depending on time $t$.
- $(k)$ Reference signal at discrete time $k$.
- $D$ Set of input/output measurements $(u_k, y_k)$.
- $H$ Hessian Matrix.
- $I_r$ Identity matrix of size $r \times r$.
- $J$ Jacobian Matrix.
- $k$ Discrete time index.
- $l_2$ Set of square summable sequences.
- $m$ Dimension of the input vector.
• $n$ Dimension of the output vector.
• $N$ Number of independent measurements.
• $P$ Dimension of the parameter vector $\theta$.
• $r$ number of rules in a Fuzzy Rule Base.
• $R$ number of neurons in a network.
• $\mathbb{R}^n$ Real $n \times 1$ dimensional space.
• $\mathbb{R}^{n \times m}$ Real $n \times m$ dimensional space.
• $\sigma$ general linear or nonlinear transfer function.
• $\theta$ Function parameter vector, general representation.
• $\theta_{NN}$ Parameter vector of a neural network system.
• $\theta_{FL}$ Parameter vector of a fuzzy logic system.
• $\theta^*$ true parameters.
• $X^T$ Transpose of a matrix $X$.
• $u^*$ True plant input, or controller output.
• $u(t)$ Controller signal depending on time $t$.
• $x(k)$ State of plant model at discrete time $k$.
• $x^{(k)}$ Present value of $x$ at step $k$ in an iterative scheme.
• $x^{[i]}_k$: $i$-th independent realization of $x_k$.
• $y$ Measured plant output.
• $y^*$ True plant output.
• $\hat{y}$ Estimated plant output.
• $\tilde{y}$ Mean plant output.
• $y(t)$ Measured plant output at time $t$.

Operators

• $A \odot B$ Hadamard-Shur product of two matrices of equal size. Each element of the matrix $A$ is multiplied with the according element of matrix $B$. Also called dot product (\(\cdot \ast\) in Matlab notation).
• $A \backslash B$ Calculation of $A^{-1}B$ using QR decomposition (Matlab notation).
• $a \triangleq b$ $a$ equals $b$, by definition.
• $A(i,:)$ $i$-th row of a matrix $A$ (Matlab notation)
• \( A(:, i) \) i-th column of a matrix \( A \) (Matlab notation).

• \( a() \) The elements of a vector \( a \), sorted column wise (Matlab notation).

• \( A() \) The elements of a matrix \( A \) concatenated to a column vector, column by column (Matlab notation).

• \( \arg \min f(x) \) Minimizing argument of \( f(x) \).

• \( \text{a.s.lim} \) Almost sure limit, the limit with probability one.

• \( \frac{\partial f(x)}{\partial x^p} \) Derivative of the vector valued function \( f(x): \mathbb{R}^n \rightarrow \mathbb{R}^m \) with respect to the vector \( x \in \mathbb{R}^{n \times 1} \). The resulting Jacobian matrix is of dimension \( \mathbb{R}^{m \times n} \).

• \( \frac{\partial^2 f(x)}{\partial x^2} \) Derivative of the scalar function \( f(x): \mathbb{R}^n \rightarrow \mathbb{R} \) with respect to the vector \( x \in \mathbb{R}^{n \times 1} \). The resulting Hessian matrix is of dimension \( \mathbb{R}^{n \times n} \).

• \( E\{x\} \) Expected value of random variable \( x \).

• \( f(\theta) \) Arbitrary \( \theta \)-dependent linear or nonlinear function.

• \( P[x] \) Probability of the event \( x \).

• \( \sigma(x) \) Activation function applied to a vector. The function is applied element wise (diagonal nonlinearity).

• \( \|a\|_n \) n-norm of a vector, defined as \( (|a_1|^n + |a_2|^n + \ldots + |a_n|^n)^{1/n} \) where \( a = \{a_1, a_2, \ldots, a_n\} \). The most important cases are for \( n = 1 \), \( n = 2 \) and \( n = \infty \).

• \( \|A\|_n \) n-norm of a matrix, defined as \( \|A\|_n = \sup_{x \neq 0} (\|Ax\|_n / \|x\|_n) \) with \( A \in \mathbb{R}^{m \times p} \), and \( x \in \mathbb{R}^p \) any arbitrary column vector. For the particular case \( n = 2 \) and \( m = p \) (A square), it can be proven that \( \|A\|_2 = \max_{1 \leq k \leq m} \sigma_k(A) \) with \( \sigma_k(A) \) the singular values of \( A \) [78].

• \( \|A\|_F \) Frobenius norm of a matrix \( A \in \mathbb{R}^{m \times n} \) with elements \( a_{i,j} \), and defined as \( \|A\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n |a_{i,j}|^2 \right)^{1/2} \).

• \( z \) Forward shift operator, \( z x_k = x_{k+1} \).

• \( z^{-1} \) Backward shift operator, \( z^{-1} x_k = x_{k-1} \).

**Abbreviations**

• AI: Artificial Intelligence.

• AIC: Akaike Information Criterion.

• ART: Adaptive resonance theory.
ARX: Autoregressive model structure with exogeneous inputs.
BAM: Bidirectional Associative Memory.
BB: Black box model structure.
BC: Bias Correction.
BIC: Bayesian Information Criterion.
BIBO: Bounded input bounded output.
BP: Backpropagation.
BPTT: Backpropagation through time.
BSB: Brain State in a Box.
CoA: Centre-of-area.
CoG: Centre-of-gravity.
CoS: Centre-of-Sums.
DP: Dynamic Backpropagation.
DUT: Device under test.
EIV: Errors-in-variables.
FAS: Fuzzy Additive System [119].
FIR: Finite Impulse Response model.
FL, FLC, FLS: Fuzzy logic, FL controller, FL system.
FLVQ: Fuzzy learning vector quantization.
FPE: Finite Prediction Error.
FRLS: Forgetting Recursive Least Square.
FOSART: Fully self-organizing simplified adaptive resonance theory.
FPE: Final Prediction Error.
FS: Fuzzy Set.
GA: Genetic Algorithm.
GAS: Global Asymptotic Stability.
GB: Grey box model structure.
GCV: Generalized Cross Validation.
GKFC: Gustafson-Kessel Fuzzy Clustering.
GNG: Growing Neural Gas.
HIP: Hidden neuron to Input layer path.
• LM: Levenberg-Marquardt.
• LMI: Linear Matrix Inequalities.
• LRSI: Local Relative Sensitivity Index.
• LS: Least squares.
• MDL: Minimum Description Length.
• MIMO: Multiple input, multiple output system.
• MISO: Multiple input, single output system.
• MLP: Multilayer Perceptron.
• MSE: Mean Square Error.
• NARMA: Nonlinear autoregressive moving average model structure.
• NARMAX: NARMA with exogeneous inputs.
• NARX: Nonlinear ARX.
• NBJ: Nonlinear Box Jenkins model.
• NFIR: Nonlinear finite impulse response model.
• NL: Nonlinear.
• $\text{NL}_q$: Concatenation of $q$ nonlinear and linear activation functions.
• $\text{NL}_{qp}$: Parallel concatenation of $p$ $\text{NL}_q$ systems.
• NN: Neural network.
• OBD: Optimal Brain Damage.
• OBS: Optimal Brain Surgeon.
• OE: Output Error.
• OHP: Output neuron to Hidden layer path.
• QP: Quadratic Programming.
• RB: Rule Base.
• RBF: Radial Basis Function.
• RBFN: RBF Network.
• RTRL: Real-time recurrent learning.
• SAM: Standard Additive FL Model [119].
• SC: Schwarz’ Criterion.
• SIMO: Single input multiple output system.
• SISO: Single input single output system.
• SMS: Shibata’s Model Selector.
• SNR: Signal to noise ratio.
• SPF: Shibata’s Penalty Function.
• SOM: Self Organizing Maps.
• SQP: Sequential Quadratic Programming.
• SV: Sensitivity Value.
• SVM: Support Vector Machine.
• TDNN: Time Delay Neural Network [159].
• TS: Takagi-Sugeno.
• VC: Vapnik-Chervonenkis.
• WLS: Weighted least squares.

Basic definitions

• Bias of a model: see Definition 6.3 on page 151 and Definition 7.5 on page 182.
• Mean BB mapping, parameter variance case: see Definition 7.1 on page 179.
• Mean BB mapping, measurement variance case: see Definition 7.3 on page 181.
• Mean square error MSE: see Definition 6.5 on page 152.
• True inputs: the noiseless inputs to the DUT.
• True model: Input-Output relation that describes exactly the Device Under Test. For black-box modeling this can be the model that maps the plant exactly. For simulations, the true model can be the functional that was used to create the simulation data.
• True outputs: outputs of the true model, when it is excited with the true inputs.
• True parameters: the model parameters of the true model.
• Variability of a model: see Definition 7.2 on page 180.
• Variance of a model: see Definition 6.4 on page 151 and Definition 7.4 on page 181.
CHAPTER 1

INTRODUCTION

Abstract: This introduction gives an overview of this thesis, approached from two different viewpoints. First, a chapter by chapter synthesis is presented and next, the main contributions to nonlinear modeling are summarized.

As a main contribution to black box modeling, this thesis presents a bird’s eye perspective on linear and nonlinear system identification. Different modeling fields are situated in a unified scheme, and it is concluded that modeling techniques can be shared between these fields. More specific, contributions are made in the fields of Neural Networks and Fuzzy Logic.
1.1 Introduction

System Identification is the art and methodology of building mathematical models of dynamical systems based on input-output data (Ljung [141]). This thesis contributes to the nonlinear part of system identification, with a special focus on the use Neural Networks and Fuzzy Logic modeling.

Neural Networks (NN) started as an imitation of the human brain by copying the functionality of human neurons. In the early days, it was believed that replicating the human brain was sufficient for building robots that worked on human reasoning. Fuzzy Logic (FL) systems were also built from the Artificial Intelligence point of view. The FL framework was initially a means for translating human linguistics and reasoning into machine control actions.

In the past decades Neural Networks and Fuzzy Logic modeling has gained increased interest, mostly because of the increased number of successful applications that were built with both modeling techniques. Currently, a demystification of both NN and FL has been reached, and the metafysic approach of both modeling techniques is replaced by a more mathematical approach.

This thesis tries to add to this demystification of the NN and FL modeling techniques, by situating them in the more general system identification framework. All modeling domains can profit from this, as is explained in more detail in Chapter 2. The final goal should be that FL and NN are accepted by linear theory adepts, while AI adepts are encouraged to look at Fuzzy Logic and Neural Networks as mere mathematical input-output relations.

For the moment, both the NN and FL modeling techniques are too much looked upon as a deus ex machina that will solve all problems, whenever the experimenter is uncapable to tackle a given task with classic linear modeling or control theories. There are still experimenters (and the public) that believe that some sort of hidden intelligence resides in nonlinear modeling that makes it capable of finding the best solution for a given problem. Especially in the early days of Neural Networks, the modeling was based upon a heuristic approach that allowed for exotic optimization techniques. One of these techniques, deliberately adding noise to the measurements, is discussed more in detail in Chapter 7 on page 177.

1.2 Thesis Overview

The chapters of this thesis are organized as follows.

- Chapter 1 gives an overview on the contents of this thesis, and explains briefly the contributions to nonlinear modeling.
• Chapter 2 describes a general modeling scheme and positions existing modeling techniques in this scheme.
• Chapter 3 discusses Neural Networks for modeling and control.
• Chapter 4 discusses some theory on Fuzzy Logic and positions FL in the general modeling scheme.
• Chapter 5 starts from NL\textsubscript{q} theory and develops the NL\textsubscript{q,p} theory for the stability analysis of FL controllers.
• Chapter 6 elaborates on the basic modeling scheme by implementing the EIV cost function for nonlinear systems.
• In chapter 7 the technique of adding jitter for the optimization of NN parameters is commented upon.
• Chapter 8 gives an example on NN modeling and describes how the model can be used for analysis and optimization of a chemical plant.

The contents of the chapters is treated briefly in the following paragraphs. Section 1.3 on page 7 gives an overview of the contributions to nonlinear modeling that can be found in this thesis.

**Chapter 2**

This thesis introduces a general modeling scheme that covers both linear and nonlinear system identification. The scheme separates the collection of measurement data, the model transfer function, the choice of a cost function, parameter initialization and optimization algorithm, and the stopping criterion. Specific techniques that are used in one part of the scheme are then assumed to be generally applicable. An example is the use of the Errors-In-Variables (EIV) cost function which is best known in linear theory. The cost function can also be used with other models, which is described in more detail in Chapter 6.

The scheme positions classic linear theory and nonlinear modeling techniques next to each other. An overview is given of the popular modeling techniques, and these techniques are situated in the general modeling scheme. This chapter doesn’t pretend to be all-inclusive. Possibly, modeling techniques exist that don’t fall exactly in the scheme. The references given, however, indicate its validity.
Chapter 3

Chapter 3 treats Neural Networks more in detail. In the past decades a large number of NN models has been proposed in the literature. Each NN architecture has its own basic model with different transfer functions and sometimes a dedicated optimization scheme. An overview of some popular NN architectures is given and the architectures are positioned in the general scheme of the previous chapter.

Once the classification in the more general identification scheme is done, it can be seen that, despite of the large number of models in the literature, most NN models fit well in the scheme. The optimization of the parameters is based on generally known principles, and it is stated that it isn’t justified to distinguish between different NN architectures, based on the method of optimization only.

This chapter also gives an overview how NN can be used for the modeling of systems with and without memory (recurrent systems). At the end of the chapter, an overview of a few popular control schemes is given. While it is common for linear systems to consider a feedforward or feedback scheme, NN control schemes typically take a more complex form. Many times the NN controller is combined with a classical linear controller, or it is configured to act as a linear controller, such that the classic stability theorems can be used for the whole control scheme.

Chapter 4

Once NN models are positioned in the general modeling scheme, the same is done for Fuzzy Logic systems. The chapter starts with the basic definitions and models that are used in the FL domain and shows that FL systems are nothing more than yet another input-output functional relationship that fits in the general identification scheme on the level of the model. The added gain with FL models is that some linguistic interpretation exists for the parameters that make up the transfer function.

The FL model is then placed in the scope of system identification. Specific problems are pinpointed, such as the significant parameter redundancy that leads to a badly conditioned optimization problem. Within the FL field, clustering was developed as a solution for this redundancy. Going back to the general identification scheme of chapter 2, clustering can be seen as a way to initialize the model parameters. It should be applicable with other identification methods too, and from this point of view the development of neurofuzzy systems becomes natural.

The large number of FL models that have been presented in the literature during the past decades, makes it a difficult task to choose between the different architectures. Indeed, one of
the commonly heared complaints about FL modeling is that no uniform modeling strategy exists. Based on the general modeling scheme of chapter 2 and a few basic assumptions, a modeling approach is presented at the end of this chapter that should be able to tackle farmost of the FL modeling problems.

Chapter 5

One of the main reasons for choosing a linear model for control, is the existence of a well understood and generally accepted stability criteria. For nonlinear systems a straightforward stability theory is not available. Recently, NL\textsubscript{q} theory was developed for the stability analysis of a class of NN systems.

In this chapter it is shown that the theory is also applicable for the stability analysis of SISO Fuzzy Logic systems. An example is given how a SISO FL systems can be described as an NL\textsubscript{q} system, and a simulation example shows how the theory is used to build an optimal FL controller with guaranteed stability.

For the analysis of more complex systems, the NL\textsubscript{q} stability theory is enhanced to NL\textsubscript{p} stability theory. NL\textsubscript{p} makes it possible to analyse the stability of a class of MIMO Fuzzy Logic control systems. An example is given to illustrate the theory.

Chapter 6

One of the choices to be made in the general scheme of chapter 2, is the selection of a cost function. Both in the NN and the FL modeling domains, it is common practise to use the Least Squares (LS) cost function in all cases to match the model and the measurement data. From linear theory it is known that, in case of noisy input-output measurements, a better choice is the use of the Errors-In-Variables (EIV) cost function.

The EIV cost function is applied to nonlinear systems in this chapter, more specifically to a NN model. The EIV cost function is defined for NN systems, and two optimization schemes are deriven to optimize the NN parameters. Next, a number of simulation examples are given, clearly showing the improved performance of a NN model when it is optimized with the EIV cost function instead of the LS cost function.

A drawback of using EIV is an increased risk for overfitting of the nonlinear model. To avoid this undesired behaviour, a modeling scheme is proposed based on a dedicated early stopping algorithm. Both the overfitting behaviour and the use of the early stopping algorithm, are demonstrated with a number of simulation examples.
Chapter 7

The Neural Network model doesn’t originate from the system identification environment, and some methods that are used for NN modeling are purely based on a heuristic approach. One of these methods is the popular technique of adding noisy measurement samples (jitter) to the existing measurements. Later on, this technique was justified by different authors, based on the bias-variance trade off.

Chapter 7 studies this technique. It starts with the definition of variance versus variability and biasing, and some examples are given to illustrate the concepts. Then, the effects of adding jitter are calculated for the case of nonlinear SISO systems. It is shown that adding noise to the inputs leads to biased model parameters, and that adding jitter is the same as imposing a zero order interpolation on the input samples. From that point of view, more elaborate interpolation techniques are proposed as an alternative.

A detailed simulation example is given to show the difference between both techniques, and it is shown that interpolation results in zero bias and very low variability of the model parameters. The calculations are then repeated for MIMO systems.

Chapter 8

An elaborated example on nonlinear system identification is given in chapter 8. A chemical plant is modelled with a NN model. The goal of the modeling is the prediction of the end temperature of a steel plant. A large number of models are mapped on the measurement data, leading to an increased prediction performance of about 72% compared with the current manual prediction rate of 60.7%.

This chapter also introduces the concept of variance backpropagation. Based on the assumption that the early stopping algorithm results in a near-optimal solution of the NN model, the limitation in performance of the model to 72% can only be caused by the noise on the measurements. In other words, the limitation in performance is not caused by variability on the model parameters, but by variance instead. The NN model can then be used to calculate a variance backpropagation of the output values to the inputs measurements. The inputs with a variance that matches the backpropagated variance are likely to be the cause for the reduced performance of the model. The NN model can therefore be used to pinpoint the inputs that should be measured better.
1.3 Contributions to the field of Black Box modeling

A. Definition of a general modeling scheme

This thesis introduces a modeling scheme that clearly separates the measurements, the choice of a model, the selection of a cost function, and the optimization scheme [231] [232]. It is shown that these choices can be made independently from each other.

The scheme clearly positions NN and FL as a nonlinear model, extending the classic linear model class. By placing linear and nonlinear modeling schemes in a unified scheme, the existing modeling techniques can profit of the present knowledge available in other identification domains. The use of a least squares cost function is considered as a natural choice in the NN and FL domains, which is not the case in the more matured field of linear modeling, where the EIV cost function is also considered as an option. On the other hand the clustering method in FL theory, and early stopping methods from the NN field, are candidate theories to be adopted in other modeling fields.

B. Addition to the demystification of NN and FL models

Within Neural Net (NN) and Fuzzy Logic (FL) modeling, e.g. the model itself and the optimization scheme aren’t considered to be separated. Some models only differ in name because of the use of another learning algorithm and a large number of models architectures exist while there is no clear strategy to choose between the models. This contributes to the unstructured impression that some outsiders have of the field.

In recent years, NN and FL were analyzed from a more mathematical point of view, demystifying both modeling techniques. This thesis tries to contribute to this demystification. Chapters 3 and 4 discuss a number of NN and FL models in the literature, from the point of view of the general modeling scheme in chapter 2. The models can be stripped until only the input-output relations remain. The comparison between these transfer functions becomes easier, and making a choice between different models is better feasible. Based on this work, a modeling scheme for FL systems is suggested in chapter 4.

Moreover, the technique of adding jitter is discussed in chapter 7. In recent years, adding jitter has been discussed in the literature from a mathematical point of view, and it is shown that it is a way of regularization. This thesis contributes to the mathematical approach, from the system identification point of view, and comments upon the jittering technique.
C. An optimization scheme for Fuzzy Logic identification

One of the problems within FL theory is the lack of a good modeling strategy. Both NN and FL systems suffer from a proliferation of possible model structures with different local and global transfer functions. Knowing that NN and FL models have universal approximation properties, it should already be clear that the popular models currently in use, are sufficient for most modeling applications. This thesis proposes the earlier mentioned general identification scheme as a general modeling strategy for Fuzzy Logic systems. Based on a few elementary assumptions, a candidate is proposed for the ideal FL model.

D. NLq{p} stability theory

While stability has been studied well in linear theory, it is still an open problem for nonlinear control systems. The NLq theory for stability analysis of a class of nonlinear systems (e.g. NN systems) is given by Suykens et al. [205]. In this thesis the theory is expanded to NLqp theory for use with parallelized nonlinear systems, such as Multi-Input Fuzzy Logic systems [227].

E. Stability of SISO and MIMO Fuzzy Logic Systems

This thesis shows that SISO Fuzzy Additive Systems can be represented in the NLq framework [237]. The stability analysis of the FS is then possible using the stability theorems that are formulated in NLq theory. An example is given how a simple SISO FL control system can be optimized with and without a stability constraint, and it is shown how the tracking performance of the control system decreases while the stability is guaranteed.

The stability analysis of Multi-Input Fuzzy Additive Systems calls for the use of NLqp theory [228]. In this thesis the theory is used on a simple system and it is pointed out that more research must be done to overcome conservatism of the stability theory.

F. Application of the Errors-In-Variables cost function in nonlinear modeling

The Errors-In-Variables (EIV) cost function is mainly used within the field of linear modeling of noisy measurement data [225]. In this thesis the cost function is applied on SISO and MIMO nonlinear systems, with a special application on Neural Networks [235] [236]. It is theoretically proven that EIV reduces biasing effects when optimizing the parameters of a Black Box (BB) model. The theoretical results are supported by a large number of simulations illustrating the increased performance when using the EIV cost function instead of a Least Squares (LS) cost function in the case of noisy input-output measurement data.
The use of the EIV cost function for nonlinear black box modeling isn’t as straightforward as it is the case for linear modeling. Nonlinear BB models suffer from an increased risk for overfitting behaviour and can easily become badly conditioned. In combination with the EIV cost function, where the measurements are allowed to be repositioned within the variance boundaries, an increased risk for overfitting was observed. This thesis introduces a modified cost function for early stopping. The use of the early stopping algorithm is demonstrated and it is shown that it can help to avoid overfitting behaviour.

When using EIV the inputs are also estimated, thus increasing the total amount of model parameters with the number of input measurements. For larger data sets this decreases the optimization speed significantly. The experimenter should therefore decide whether the increased performance justifies the increase in modeling effort.

**G. Introduction of variability versus variance in nonlinear modeling**

Quite a few experimenters in the NN field expect that a Neural Network provides perfect mappings of the measurement data. If the NN doesn’t fit the data well, this is commonly explained as a one time bad mapping, due to local minima in the cost function. The optimization of the NN parameters is then redone with new random starting parameters until the experimenter is satisfied with the identification results.

It must be understood that the NN mappings show a sort of random behaviour when a large number of NN mappings are considered on noiseless measurement data. This is even the case if all measurement points are mapped exactly. This random behaviour can be characterized by its variance, calculated from different initial parameters but with the same data set. This variance is called variability in this thesis. The variability of the NN model is due to the use of random starting values, early stopping and by local minima in the cost function. In the cases where this random behaviour goes to the extreme, it is commonly recognized as overfitting behaviour. Yet, even in the case where no true overfitting is detected, it must be understood that the interconnection from one measurement point to the other is smooth but random.

The concept of variability is different from the better known concept of variance due to measurement noise. Consider a large number of NN mappings based on different sets of noisy measurement data. In that case a variance on the NN mappings is observed too, even if the global minimum is reached for each measurement set. The final nonlinear mapping therefore suffers both from variance and variability.
H. Comments on the use of adding jitter to improve generalization

The technique of adding jitter is often promoted as a means of reducing overfitting behaviour or improve generalization of the NN model. In this thesis it is stated that adding jitter is actually a means to reduce the variability of a NN model, since overfitting is just an extreme case of variability. From the point of view of linear modeling, the use of jitter is heavily disputed [233].

This thesis analyses the effects of adding input and output noise, based on a higher order Taylor series expansion of the popular least squares cost function [229]. It is shown that adding noise imposes a smoothing effect by suppressing higher order derivatives in the model transfer function. The inherent drawback of the method is an increased bias of the model parameters. Moreover, examples show that the use of jitter causes variance in the model parameters while variability remains.

This thesis proposes to use interpolation techniques [230] as a better alternative to adding noise. Examples are given that clearly show how the variability (i.e. overfitting behaviour) can be heavily reduced, with small biasing effects and negligible variance. The selection of interpolating data forces the experimenter to make a consient choice on the desired behaviour of the NN.

I. Variance backpropagation as an analysis tool

Consider the case where a large number of models is mapped on a given set of measurements, and where the prediction performance of the models seems to be limited to a maximal hit rate. Based on the large number of models and under the assumption that early stopping has prevented overfitting, it is expected that the model variability is minimal. As a result the only cause for the reduced prediction performance is measurement noise.

This understanding has been used in the analysis of the MIMO model of a steel plant [226]. Based on a limited number of real-time noisy measurements of a steel plant, a number of NN models was mapped. Although these models already reached a significant improvement in the prediction of the steel quality, the hit rate seemed to be limited. A novel technique was used, called variance backpropagation, to analyze the output variance based on the NN model. Variance backpropagation allows to pinpoint the inputs that limit the performance. The NN model therefore not only improved the prediction hit rate, but also suggests how the model could perform even better, if some inputs are better measured.
This chapter introduces a general identification scheme for system modeling, clearly separating the model, cost function and optimization scheme. Although NN and FL originate from the Artificial Intelligence domain, it is shown that they fit well in the identification scheme as a nonlinear model. The gain for both modeling schemes is that they can profit from present knowledge within nonlinear system identification, such as cost function properties and optimization schemes. The references given, show that the current bibliography on FL and NN can be situated within each stage of the identification scheme, thus justifying its use.
2.1 Models considered in this thesis

This thesis considers models that can be described as a functional relationship

\[ \hat{y} = f(\hat{u}, \theta) \]  

(2-1)

with \( \hat{u} \in \mathbb{R}^m \) the estimated value of a measured input \( u \in \mathbb{R}^m \), \( \hat{y} \in \mathbb{R}^n \) the estimated value of the measured output \( y \in \mathbb{R}^n \) and \( \theta \) the parameter space of the linear or nonlinear relationship \( f \).

If the estimates \( \hat{u} \) and \( \hat{y} \) are used for modeling, we can find ourselves in the case of EIV modeling, which is discussed more in detail in Chapter 6 on page 143. When using Output Error modeling, it is assumed that \( \hat{u} = u \) such that the relationship \( \hat{y} = f(u, \theta) \) is estimated.

The functional input-output relationship \( f \) can be any black box representation \( f_{BB} \) of the measurement data, e.g. linear systems, NN, FL, ... systems. In all cases the input-output relationship is described by the parameter vector or matrix \( \theta \), i.e. in opposition to non-parametric models (e.g. static SVM’s [206] [238]) this thesis considers parametric models such that the model parameters \( \theta \) are estimated. The parameter space is considered to be static.

2.2 Different modeling theories brought together

Linear theory has for long ruled the domain of modeling and control of physical processes. In the past decades alternative theories were developed that could handle more complex systems. These theories were not necessarily built upon the conventional linear theory. To date, two specific alternative modeling techniques have become popular in the engineering field: Fuzzy Logic Systems and Neural Networks. FL was invented by Zadeh, based upon possibility theory and the need for controllers that can handle complex control rules. NN had a direct relationship with Artificial Intelligence and were based upon the human neuron.

Neural Networks (NN) and Fuzzy Logic (FL) systems are often considered as separate fields, apart from the classic linear domain. Since both modeling theories are closely related to the Artificial Intelligence domain, the terminology used within the fields is not always the same. While a cost function is minimized in conventional linear theory, NN adepts speak of learning a network and Fuzzy Logic researchers often refer to adaptation when they change the model parameters. Adaptation within classic linear theory is more often used to indicate that the linear parameters are slowly updated because of a time-varying process. Sometimes the same term is used differently within a field. Bayesian NN can be referred to as networks that use a Bayes cost function, while sometimes a network is meant that is based on probability values.
instead of classical neurons. A short overview of apparent different terminologies is given in TABLE 2-1.

<table>
<thead>
<tr>
<th>Linear theory</th>
<th>Neural Networks</th>
<th>Fuzzy Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimization</td>
<td>learning</td>
<td>adaptation</td>
</tr>
<tr>
<td>optimization step</td>
<td>learning rule, search</td>
<td>learning step</td>
</tr>
<tr>
<td></td>
<td>direction, update</td>
<td></td>
</tr>
<tr>
<td>cost function</td>
<td>penalty</td>
<td>cost</td>
</tr>
<tr>
<td>gradient methods</td>
<td>backpropagation</td>
<td>backpropagation</td>
</tr>
<tr>
<td>Weighted Least Squares</td>
<td>Bayesian</td>
<td>mapping</td>
</tr>
<tr>
<td>modeling</td>
<td>mapping, fitting</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 2-1. Comparison of terminology.**

Within each field, efforts have been made to provide a uniform mathematical framework, e.g. Neural Networks (Haykin [86]), Fuzzy Logic (Lewis [130]), linear theory (Ljung [140]) or general Black Box modeling (Juditsky [110]). In this thesis it is assumed that black box models can be regarded as a nonlinear extension to linear theory on the level of the model (see further). For that reason the linear theory conventions are used in this book whenever possible. Terms that have become very accustomed in the nonlinear field (e.g. the use of backpropagation to describe optimization), are adopted in this thesis.

### 2.3 A scheme for linear and nonlinear modeling

Regardless of the model used (linear, FL $^1$, RBF, ...), the goal of the modeling is to find the proper model parameters $\theta^*$ that, either describe the true input-output relationship of a measured plant, or correlate with the true physical parameters of a measured system. In practise this means that a cost function $C$ that matches the measured data with the model, is minimized. The identification is done in three major steps.

The first major step is taking basic decisions prior to the modeling: which measurements are needed for training? Which linear or nonlinear model is used, and which cost function is minimized? Every single one of these three choices has a direct influence on the subsequent modeling steps. Therefore, these three basic decisions are the most important choices to be made.

1. All abbreviations can be found in the glossary on page xiii.
The second step considers the minimization of the cost function and is more a technical problem once the three basic choices are made. The parameters are initialized and an optimization scheme is chosen. The experimenter must also decide when to stop the optimization.

Third comes a validation of the model. Usually the model is tested on a separate test set, or on new data. Optionally a robustness or stability analysis is performed as a validation of the model.

After the model validation, the model can be rejected or the experimenter can decide to change some of the basic choices to obtain better performance. This can be considered as an overall recursive procedure that includes the given modeling scheme, as it is shown in FIGURE 2-2. This thesis only treats how a single model is derived and doesn’t go into detail on the overall iterative procedure.

Chen et al. [40] already recognized the need for a separation of the structure selection and the parameter optimization when they formulated the SSPE (Structure Selection and Parameter Estimation) principle but, as shown here, there is more to be taken care of than just choosing the model and optimize the model parameters.

The analysis of the model robustness and stability doesn’t necessarily come at the end of the modeling. It is possible to enforce these criteria even before the validation section by choosing the proper cost function. This is explained more in detail in section 2.6.

An overview of the three sections is given in TABLE 2-2. The way how these sections are related to each other, is given in FIGURE 2-1. In the following paragraphs, the sub parts of this scheme are discussed more in detail. Throughout the book the different chapters are always referred to this modeling scheme.

<table>
<thead>
<tr>
<th>Basic Choices</th>
<th>Collect measurements.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Select a model.</td>
</tr>
<tr>
<td></td>
<td>Select cost function.</td>
</tr>
<tr>
<td>Optimization</td>
<td>Model parameter initialization.</td>
</tr>
<tr>
<td></td>
<td>Optimization of the parameters.</td>
</tr>
<tr>
<td></td>
<td>Determine when to stop optimization.</td>
</tr>
<tr>
<td>Validation</td>
<td>Model validation.</td>
</tr>
</tbody>
</table>

**TABLE 2-2. Identification steps.**
2.4 Design a good experiment

Consider a robot setup where a gripper is positioned by a collection of individually controlled motors. The combination of all possible motor positions results in various positions and angles for the grippers. Call the whole set of all possible gripper positions and angles and the set of all according motor positions the domain of interest $\psi \subset \mathbb{R}^D$. 
Assumption 2.1 In this book we assume that for each modeled plant a domain of interest \( \psi \subset \mathbb{R}^D \) can be defined.

To comply with Assumption 2.1 it can be enough to define minimum and maximum boundaries for all inputs and outputs. In the case of the robot example, however, it is very well possible that some restricted regions of forbidden motor positions exist. The combination of these motor positions could e.g. lead to robot damage. On the one hand, the domain of interest becomes more complicated, on the other hand this could allow for the choice of models that have less complexity.

Independent of the model chosen, it is impossible to identify a reliable model without measurements that fully cover the domain of interest. Once the model is optimized for the given measurements, interpolation or generalization is very well possible, even with nonlinear models or under conditions of sparse data [13] [192] [198] [214]. Extrapolation of a nonlinear system, however, can lead to unpredictable results [223]. The measurements should therefore cover the whole \( \psi \) domain (e.g. the application on RBFN given by Gorinevsky [79]).

Definition 2.1 Generalization is the ability to predict future observations, using new data, even when the model was optimized using noisy or sparse data.

The basic decision is, therefore, the choice of proper excitation signals that cover the \( \psi \) domain fully. The only way of getting valuable information needed for modeling or control (observability and controllability demands) from the system is the selection of good measurements. In this scope we can consider the use of expert knowledge as a way of
extracting information from the plant [181]. It is a severe misunderstanding to think that expanding the measurement set by adding random noise, adds information (this is proven in detail in Chapter 7). This technique, however, is often used in the field of NN [23] [75] to overcome overtraining or to increase generalization [13] [145].

### 2.5 Selection of a model structure

Regardless of the fact that someone uses linear modeling [140] [188], a NN mapping [26] [192] [205], a FL system [259] [262] or any other nonlinear model [60] [83] [110] [121] [141] [194] [202] [214], regardless of the fact that someone uses White Box, Black Box or Grey Box [69] [110] [141] [192] [225] modeling, a static model is just a mapping from a given input to a given output. This mapping is a mathematical representation dependent on static or time-dependent model parameters. A White Box representation of a plant is favoured whenever detailed information of a system is demanded, e.g. the effects of transistor dimensions on the HF properties in a VLSI design. Whenever it is not needed to extract detailed information, Black Box or Grey Box modeling can be sufficient.

It is clear that the basic choice here, is the selection of the model. There have been many fierce discussions in favour or against different models [2]. Yet, for any given continuous function that operates in \( \mathbb{R} \) it is possible to prove universal approximation properties when used in a well chosen BB structure (Kolmogorov’s theorem, given in Appendix A.1) [41] [73] [97] [119] [122] [241] [249], such that from a theoretical point of view any basic BB model would fit the task.

From a practical point of view, however, the best fit is the one that needs the least number of parameters to model the measurement data with a user defined sufficient small error. A sine wave is best fitted using a sine model while an exponential is best fitted using an exponential model. Locally linear systems are best fitted using e.g. splines [55] or a FL system. The best model that can be used to fit a functional, is the functional itself. Choosing the model is fully based upon typical transfer functions, or convergence properties whenever the parameters are optimized.

<table>
<thead>
<tr>
<th>Model Cost Function</th>
<th>Optimization Scheme</th>
<th>Stopping Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Measurements</td>
<td>Model validation</td>
</tr>
<tr>
<td>Plant Model</td>
<td>Model</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Plant Model</td>
</tr>
</tbody>
</table>
2.5.1 Overview of a few model classes

A. Systems that are linear-in-the-parameters

A simple linear system is given as

\[ y = Au \]  

such that \( \Theta = A \) \hspace{1cm} (2-2)

and e.g. polynomial mappings have an input-output relationship

\[ y = a_0 + a_1 u + a_2 u^T u + \ldots \text{ such that } \Theta = [a_0 \ a_1^T \ a_2 \ldots]^T. \]  

(2-3)

For larger intervals the degree of the approximating polynomial can become very large. In the theory of splines [52] [55] the input domain is therefore divided in sufficiently small intervals, such that on each interval a polynomial can be defined with relatively small degree. This can be done such that the composite function has several continuous derivatives. There are two types of splines commonly used. The \textit{ppform} divides the the input domain in \( l \) intervals with boundaries \( \xi_i, i = 1, 2, \ldots, l + 1 \) with \( a = \xi_1 < \ldots < \xi_{l+1} = b \). Each local polynomial \( y_j(u) \) of order \( k \) is denoted as

\[ y_j(u) = \sum_{i=1}^{k} a_{ji} \frac{(u - \xi_j)^{k-i}}{(k-i)!} \]  

(2-4)

where \( j = 1, 2, \ldots, l \) and the \( a_{ji} \) are local polynomial coefficients. The \textit{B-form} describes the spline as a linear combination

\[ y(u) = \sum_{i=1}^{N} a_i B_{i,k}(u) \]  

(2-5)

where the \( a_i \) are elements of the parameter space and the \( B_{i,k}(u) \) are ppsplines with degree smaller than \( k \) and breaks (knots) at \( \xi_{i}, \ldots, \xi_{i+k} \). The \( B_{i,k}(u) \) are normalized, nonnegative and zero outside the interval \([\xi_{i}, \xi_{i+k}]\). Many theoretical properties exist for B-splines, such as recurrency relations, approximation properties, and best interpolant properties with respect to the derivative degree.

More general, it is possible to describe a model linear in the parameters as

\[ y = Af(u) \text{ where } \Theta = A. \]  

(2-6)

In all cases the transfer function is linear in the parameters, even if the functional relationship \( f(u) \) is nonlinear with respect to \( u \). The transfer function is globally defined and the second
derivative with respect to the parameters equals zero. The optimization of (2-3) for the \( a_i \) resolves to a linear regression model [40]. The properties of systems that are linear in the parameters, are well known and identification methods are treated in detail in the literature. The optimization of the parameters is a convex problem with a single global minimum.

### B. Neural Networks

The input/output relationships of NN are treated more in detail in Chapter 3. There is no general definition for NN systems that covers all possible NN topologies. Many forms exist, a popular one being the Multilayer Perceptron (MLP) architecture with an input-output relationship

\[
y = \sigma_1 A_1 (\sigma_2 (\ldots \sigma_R (A_R u + \beta_n) \ldots + \beta_2) + \beta_1)
\]

with \( \sigma_i \) a nonlinear function that is applied element wise (e.g. \( \sigma(x) = \tanh(x) \)), the \( A_i \) matrices with dimension \( n_i \times n_{i+1} \) with \( n_1 = n \) and \( n_{R+1} = m \) and the \( \beta_i \) vectors of size \( n_i \times 1 \). The parameter space is chosen as

\[
\theta = [A_1(:,):\beta_1(:,):A_2(:,):\beta_2(:,):\ldots: A_n(:,):\beta_n(:,)]
\]

The input-output relation is typically a soft nonlinear function. Sometimes hard-bounded functions are used, such as \( \sigma(x) = sat(x) \) or \( \sigma(x) = sign(x) \), mainly for pattern recognition [26] and classification. The optimization of the parameters suffers from local minima and is often badly conditioned.

### C. Fuzzy Logic and Neurofuzzy systems

FL systems vary a lot with respect to the input-output relationship, just as is the case for NN. Modeling is typically done with a fuzzification part, an inference part and a defuzzification part. Different models exist for each sub part. Two models rule within Fuzzy Logic: the Mamdani systems and the Takagi-Sugeno systems. The Takagi-Sugeno model is defined in [171] as

\[
y = \frac{\sum_{i=1}^{R} b_i(u) \mu_i(u)}{\sum_{i=1}^{R} \mu_i(u)}
\]
with \( \mu_i(u) \) the membership degree of the fuzzified input and \( b_j \) a \( u \)-dependent functional, e.g. the relationship

\[
b_j = a_{i,0} + a_{i,1}u + a_{i,2}u^T + \ldots.
\]

(2-10)

\( R \) is the number of rules that make up the inference mechanism. The Mamdani model defined as a Generalized Additive Fuzzy System in [119] uses the special case that \( b_j = a_{i,0} \).

FL systems can be configured as locally linear systems with a smooth interpolation from one operating point to the other. The parameter space \( \Theta \) includes the \( a_{i,j} \), the rules that form the rule base (also called Knowledge Base), and the specific parameters that form the input and output fuzzy sets.

The optimization depends on the chosen parameters, but is difficult due to local minima and redundancy within the many parameters. A technique to overcome this redundancy is to learn each sub part separately, or by fixing a sub part, e.g. the Knowledge Base. A popular method is the feedforward calculation of the fuzzy sets and learn the parameters \( a_{i,j} \) in (2-10).

D. Radial Basis Function Networks

RBFN are in [41] and [191] defined as

\[
f_{RBF} = \sum_{i=1}^{R} w_i f_{RBF,i}(u) \quad \text{with} \quad f_{RBF,i}(u) = \exp\left(-\frac{\|u - c_i\|^2}{2\sigma_i^2}\right),
\]

(2-11)

\( w_i \) some weight function, \( R \) the number of basis functions, \( c_i \) the centre vector and \( \sigma_i^2 \) the spread of the basis functions.

RBFs are much used for the mapping of local hard nonlinear features and can be used in a NN configuration [79] such that \( \sigma_i = f_{RBF,i} \) in equation (2-7). The hard nonlinear properties makes them also useable for classification [121]. Sometimes \( c_i \) and \( \sigma_i \) are fixed to avoid local minima in the cost function [23] (only the linear parameters are optimized), but this can severely decrease the performance of the RBF for black box modeling. The 'Radial' part in the definition denotes that the function \( f_{RBF,i} \) only depends on the radius \( \|u - c_i\| \).

2.5.2 Choosing the model

To fix the ideas, FIGURE 2-3 on page 21 shows a few transfer functions based on a 2-input, single output model. The figure shows the output of six different models, with the phase plane \((u, \dot{u})\) with \( \dot{u} = du/dt \) as the input. The inputs \( u \in [-1, 1] \) and \( \dot{u} \in [-1, 1] \) are normalized.
and the normalized output $f \in [-1, 1]$ is shown, showing the different characteristic input-output relationships.

It is up to the experimenter to make a choice which of the typical transfer functions corresponds best with the measurements. A measurement set that is best mapped using a Radial Basis Function network, could also be modelled using a Fuzzy Logic system. In the latter case, however, the experimenter would need e.g. a large number of fuzzy sets to describe the input-output relationship with the same accuracy, i.e. the number of parameters is much larger.

As mentioned earlier, the choice of the model is not only based on the input-output relationship on its own. Other elements to take into consideration are convergence properties when
optimizing the model, the availability of stability theorems, ... or maybe just the availability of
the proper mathematical tools to do the modeling.

Of course, the list given in the figure and in the previous sections, is not exhaustive at all. Sometimes
very dedicated White Box models or complex Black Box models are used, such as
Feature Space Mapping [60], Support Vector Machines, triangular nonlinear structures [190]
or saturating linear systems [93]. The very general definitions for NN and FL makes them
appropriate for mixed models, such as Neurofuzzy systems [104] [133] [247] or mixed with PI
and PD controllers [156] [257] [260]. Overviews of other nonlinear systems are given in [60]
[83] [110] [121] [141] [194] [202] and [214].

All of these models are based on the same principle: how to provide a mathematical
relationship between a given input and a given output? The remaining task is the choice of the
model parameters that should be learned or optimized.

### 2.5.3 Model order selection

#### A. Introduction

For systems that are linear in the parameters, different methods exist for the choice of the
optimal number of parameters. A number of analytical criteria exist that provide a penalty
factor for an increased model order, under the assumption that the noise is independent and
identically distributed. In general, the penalty factor \( C_P \) takes the form

\[
C_P = \frac{r(d/N)}{N} \sum_{k=1}^{N} (y_k - f_{LIN}(u_k, \theta))^2
\]  

(2-12)

with \( u_k \) and \( y_k \) the input and output measurements, \( N \) the total number of measurements and
\( f_{LIN} \) the system linear in the parameters with parameters \( \theta \). \( d \) is the model complexity
(degrees of freedom) of the linear system. Denote \( p \) as \( p = d/N \), then different forms of \( r(p) \)
exist in the literature which all work for large data sets [43]:

- Final Prediction Error (FPE): \( r(p) = (1 + p)/(1 - p) \).
- Schwartz’ Criterion (SC): \( r(p, N) = 1 + p \frac{\ln(N)}{2(1 - p)} \), which correlates
  with the Minimum Description Length (MDL) [183] method.
- Generalized Cross-Validation (GCV): \( r(p) = (1 - p)^{-2} \).
- Shibata’s Model Selector (SMS): \( r(p) = 1 + 2p \), also called Shibata’s
  Penalty Function (SPF).
For nonlinear systems, the choice of the number of parameters needed for an optimal identification, is an open problem. Often the number of parameters are chosen by rules of thumb or just trial and error. A too low number of model parameters can lead to biasing and bad generalization, while a too high number of parameters can lead to overfitting and badly conditioned Jacobian and Hessian matrices during optimization of the parameters.

**B. Cross-validation**

Cross-validation is the technique of using a test set for the validation of the data. It demands that a larger number of models is mapped against the measurement data, and the model with the smallest parameter vector and sufficient performance is selected out. Cross-validation is time consuming, but remains one of the most popular selection procedures [240].

**C. Akaike Information Criterion**

A frequently used method for the selection of the optimal model order, is to start with a low number of parameters and then optimize more models with an increasing number of parameters. For systems linear in the parameters this could mean that the order is increased, while e.g. for neural networks the number of neurons in the network is increased.

This is done until the increase in performance of the BB model on the test set becomes too low compared with the increase of parameters. The method resembles early stopping for the model order complexity. This can be regarded as an overall iterative scheme on the chosen model and is a practical approach to the Akaike Criterion (AIC) [5].

**D. Vapnik-Chervonenkis (VC) theory**

VC theory [238] defines a measure of complexity (called the VC-dimension). For linear systems it provides a principled solution to the problems of finding a meaningful complexity index of a given model, and the estimation of the unknown prediction error (based on a test set) compared with the known prediction error (based on the learning set) [43].

VC theory orders the set of possible models according to their complexity. Therefore, the whole set $S$ of all possible models must have a structure of nested submodels, such that $S_1 \subset S_2 \subset \ldots \subset S_k \subset \ldots$, where each model $S_k$ has a finite VC dimension $h_k$. Because of the definition of the VC dimension, also the VC dimensions should be ordered, such that $h_1 \leq h_2 \leq \ldots \leq h_k \leq \ldots$.

For linear systems, the VC dimension can be set as the number of free parameters: $h = d$, where $d$ was defined in the previous section. The optimal order for systems linear in the parameters can then be based on the penalty function (2-12), where $r$ is calculated as.
A Scheme For Nonlinear modeling

\[ r(p, N) = \left( 1 - \sqrt[p]{p \ln(p) + \frac{\ln(N)}{2N}} \right)_+^{-1} \] \[ (2-13) \]

For polynomial estimators of degree \( d_p \), the VC dimension equals \( h = d_p + 1 \). For nonlinear systems the VC dimension is harder to calculate. VC-theory can’t be applied to nonlinear estimators easily, since the VC dimension can’t be accurately estimated for nonlinear models.

E. Bayesian methods

Bayesian methods for model order selection can be found in e.g. Mackay [143]-[147], Neal [160], Vila [240] and Wright [251]. Bayesian learning starts from a prior distribution of the model parameters, which is then multiplied by the likelihood to obtain the probability of the model parameters given by the measurement data (the posterior distribution). Predictions on the model are made by integrating over the posterior. In practise, the prior distribution is often criticised for being subjective.

The Bayesian framework assumes that the parameters of the model are random variables, for which also the distribution should be estimated. The evidence (distribution) for some of the parameters seems a good measure for the determination of the model order. Yet, the practical difficulties in computing reliable estimates, limits the use of Bayesian networks severely. The evaluation of the estimates typically requires the computation of high-dimensional integrals with no closed-form analytical expression [15].

F. Other methods

It is beyond the scope of this thesis to give a detailed description of all possible methods to do model order selection. In this section methods were given that are mainly based on some penalty function describing the complexity of the model. Next to using such a penalty function, growing methods [186] and pruning methods (some of which are described in the next section) are available.

A brief overview of some other methods available in the literature is given here.

- Final Prediction Error (FPE) (Ljung [140]).
- Final Prediction Error Biased criterion (FPEB) (Alippi [8]).
- Generalized Prediction Error (GPE) (Moody [154]).
- Maximum A Posteriori (MAP) (e.g. Liu et al. [138]).
- Minimum Eigenvalue criterium (MEV) (Liang et al. [131]).
- Network Information Criterion (NIC) (Murata et al. [157]).
- Orthogonal Least Squares methods (OLS) (Hong et al. [95]).
• Reversible jump Markov Chain Monte Carlo (MCMC) method (Andrieu et al. [15]).
• Two-Dimensional Autoregressive (2D AR) model order estimation (Aksasse [6], [7]).

2.5.4 Pruning

A. Optimal Brain Damage

Instead of starting from a simple model, some BB modeling techniques start from large models with a very large number of parameters, and reduce the number of parameters in the optimization process. This is called pruning. Within the NN field, Optimal Brain Damage (OBD) has become a popular method for pruning. OBD ranks the weights according to importance or saliency. The saliency $\varsigma_i$ of a parameter $\theta_i$ is calculated as

$$\varsigma_i = -\theta_i \frac{\partial C(u_k, y_k, \theta)}{\partial \theta_i} \bigg|_{\theta = \theta(k)} + \frac{1}{2} \theta_i^2 \frac{\partial^2 C(u_k, y_k, \theta)}{\partial \theta_i^2} \bigg|_{\theta = \theta(k)}$$

in which $C(u_k, y_k, \theta)$ is any of the cost functions described in the following section. Neurons with a saliency below a user-defined threshold ($|\varsigma_i| < \text{threshold}$), are removed from the network.

Ideally, the saliencies should be calculated in the true parameters, i.e. $\theta = \theta^*$. During the optimization, $\theta^*$ is unknown, such that parameters with small saliencies are repeatedly removed based on the present parameter values $\theta = \theta(k)$.

A correlated pruning technique, Optimal Brain Surgeon (OBS), is discussed in Hassibi et al. [85].

B. Sensitivity Term Pruning based on LRSI

A simple method for pruning Neural Networks was introduced in Karnin [112] for global pruning and Ponnapalli et al. [178] for local pruning. The Local Relative Sensitivity Index (LRSI) method is described here. The technique has the advantage that it uses the sensitivity values (SV’s) of a model. These SV’s are elements of the Jacobian matrix, such that little extra calculation effort is needed when combining the pruning method with gradient based optimization schemes.

The pruning method is dedicated for MLP networks that are optimized with backpropagation (or other gradient methods) and based on the LS cost function. To fix the ideas, assume a two-layer SISO MLP neural network. The pruning method treats the neurons in each layer differently, and distinguishes between the hidden neuron to input layer paths (HIP) and the output neuron to hidden layer paths (OHP).
During each iteration step, the sensitivity values are calculated as follows:

$$SV_{jk} = \sum_{i=0}^{K-1} \left[ \frac{\partial C_{LS}}{\partial w_{jk}} \Delta w_{jk}(i) \right] \times \frac{w_{jk}(final) - w_{jk}(initial)}{w_{jk}(final) - w_{jk}(initial)}$$  \hspace{1cm} (2-15)

with $SV_{jk}$ the sensitivity value of the weight $w_{jk}$ in the MLP network. $C_{LS}$ is the LS cost, such that $\partial C_{LS}/\partial w_{jk}$ is an element of the Jacobian matrix. $K$ is the number of optimization iterations and $\Delta w_{jk}(i)$ is the change in weight of $w_{jk}$ in the $i$-th iteration. $w_{jk}(initial)$ is the initial weight when starting the optimization, while $w_{jk}(final)$ is the weight value at iteration $K$.

Based on the SV’s, the LRSI is calculated separately for the HIP and OHP paths. This is done using the equation

$$LRSI_{jk} = \frac{|SV_{jk}|}{\sum_{i=1}^{p}|SV_{ji}|}$$  \hspace{1cm} (2-16)

with $p$ the number of neurons in the layer for which the LRSI is calculated.

By nature the LRSI is a value between zero and one. Neurons with a low LRSI (e.g. smaller than 0.01) are assumed to be of lesser importance and can be pruned during the optimization. This procedure is followed as long as the LS cost decreases.

### 2.6 The need of carefully choosing the cost function

#### 2.6.1 The LS and WLS cost functions

With no doubt, the least squares (LS) cost function is the most used to date. The cost function is easy to comprehend, and easy to apply. Define $n$ as the length of the estimated output vector and $N$ as the number of MIMO input-output measurement pairs $(u_k, y_k)$ with $u_k \in \mathbb{R}^m$ and $y_k \in \mathbb{R}^n$. The LS cost function is then defined as

$$C_{LS} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left( f_j(u_k, \theta) - y_{k,i} \right)^2$$  \hspace{1cm} (2-17)
with \( y_k = [y_{k,1}, y_{k,2}, \ldots, y_{k,n}]^T \) and \( f_i(u_k, \theta) \) the \( i \)-th element of the MIMO function \( f(u_k, \theta) \). Define the error vector \( e = [(f(u_1, \theta) - y_1); (f(u_2, \theta) - y_2); \ldots; (f(u_N, \theta) - y_N)] \).

Then the LS cost can be written as

\[
C_{LS} = \frac{1}{N} e^T e. \tag{2-18}
\]

Strictly spoken, the \( 1/N \) factor is only needed to make the cost independent of the number of samples and to guarantee a finite cost function when the number of experiments goes to infinity, and it is possible to omit the factor during optimization.

The measurements can also be noisy such that

\[
\begin{align*}
    u_k &= u_k^* + n_{u,k} \\
y_k &= y_k^* + n_{y,k}
\end{align*} \tag{2-19}
\]

where \( u_k^* \) and \( y_k^* \) denote the noiseless inputs and outputs respectively and \( n_{u,k} \) and \( n_{y,k} \) are additive noise contributions that are independent of \( k \). Assume that for each measurement the output noise variance \( \sigma_{y,k}^2 \) can be calculated and that the input is exactly known \((n_{u,k} = 0)\).

The variance can be used to normalize the cost function (2-17), giving the WLS cost function [26] [145] [236], also called the Bayes cost function [26]

\[
C_{WLS} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \frac{(f_i(u_k, \theta) - y_{k,i})^2}{\sigma_{y,k,i}^2}. \tag{2-20}
\]

A larger variance leads to a smaller influence of that measurement in the cost function, leading to a smaller model uncertainty.

In practise the proper choice of the cost function is often neglected and a simple LS cost function is selected, because ‘everyone does it like that’. The use of the WLS cost function results in a smaller model uncertainty and should be preferred over the LS cost whenever output variances are available. None of these two costs, however, deals with noisy input samples. In Chapter 6 it will be shown how this can lead to erroneous model parameters when modeling. The following sections present a few other cost functions that can be used instead of the LS cost in certain circumstances.

### 2.6.2 Cost functions based on probability

Define the same input-output pairs \((u_k, y_k)\) as in the previous section, with \( k = 1, 2, \ldots, N \) and \( n \) the length of the estimated output vector. It is possible to consider two distinct cases. The case where the input measurements are exactly known, results in the Weighted Least
Squares cost function. The case where noisy inputs are considered results in the Errors-In-Variables (EIV) cost function.

The Bayes cost function is based on the maximization of the probability of the optimal parameters $P(\theta^*|\text{measurements},f_{BB})$, given a set of measurements and a certain model $f_{BB}$. We can use the Bayes theorem

$$P(\theta^*|\text{measurements},f_{BB}) = \frac{P(\text{measurements}|\theta^*,f_{BB})P(\theta^*)}{P(\text{measurements}|f_{BB})}$$

in which $P(\text{measurements}|f_{BB})$ is independent from $\theta^*$. In practice $P(\theta^*|f_{BB})$ is generally unknown and, therefore, assumed to be constant (in the Bayes framework, this corresponds to a uniform distribution). The maximization of $P(\theta^*|\text{measurements},f_{BB})$ then reduces to the maximization of $P(\text{measurements}|\theta^*,f_{BB})$ which is called the likelihood of the measurements, when $\theta^*$ and $f_{BB}$ are given. The resulting estimator is called the maximum likelihood estimator. Assume that the noise contributions on the $y_k$ are also e.g. Gaussian and independently distributed over $k$, such that

$$P(\text{measurements}|\theta^*,f_{BB}) = \prod_{k=1}^{N} P(y_k|\theta^*,f_{BB})$$

with

$$P(y_k|\theta^*,f_{BB}) = \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left( -\frac{(\hat{y}_k-y_k)^2}{2\sigma_{y,k}^2} \right).$$

Instead of maximizing the likelihood, it is more convenient to minimize the negative logarithm of the likelihood and the maximization of (2-21) becomes

$$\arg\max_{\theta, y_k, i} (P(\text{measurements}|\theta^*,f_{BB})) = \arg\min_{\theta, y_k, i} (C_{WLS}).$$

In this equation the WLS cost is defined as

$$C_{WLS} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left[ \frac{(\hat{y}_k,i-y_k,i)^2}{\sigma_{y,k,i}^2} + \lambda_k (f_i(u_k, \theta) - \hat{y}_k,i) \right].$$

The multiplication with the $1/N$ term has the advantage that the cost function does not grow with the size of the measurement data set. Expressing the stationarity of the Lagrange parameters $\lambda_k$ gives
The need of carefully choosing the cost function

\[
\frac{\partial C_{WLS}}{\partial \lambda_k} = 0 \quad \Rightarrow \quad \hat{y}_{k,i} = f_i(u_k, \theta), \quad (2-26)
\]

such that, under the assumption that the noise is Gaussian distributed, we become the WLS cost function (2-20) again.

For unknown or constant variances \(\sigma^2_{y,k,i} = \sigma^2_y\) the WLS cost function can be simplified to become the much used LS cost function (2-17). The maximum likelihood approach, therefore, supports the intuitive choice of the LS and WLS cost functions given in the previous section.

On the other hand, the maximum likelihood approach also gives a way to deal with noisy input measurements. If also the inputs \(u_k\) are noisy with known or estimated variances \(\sigma^2_{k,i,u}\) and assuming for simplicity that all noise contributions \(n_{k,u}\) and \(n_{k,y}\) are independent realisations, then the Bayes probability function becomes

\[
P(\theta^*|u, y, f_{BB}) = \frac{P(u|\theta^*, f_{BB})P(y|\theta^*, f_{BB})P(\theta^*|f_{BB})}{P(u|f_{BB})P(y|f_{BB})} = \prod_{k=1}^N \left[ P(u_k|\theta^*, f_{BB}) \right] \prod_{k=1}^N \left[ P(y_k|\theta^*, f_{BB}) \right] P(\theta^*|f_{BB}) \quad (2-27)
\]

In this case both \(P(u|f_{BB})\) and \(P(y|f_{BB})\) are assumed to be independent from \(\theta^*\), and the noise contributions on the inputs \(u_k\) are assumed to be independent and Gaussian distributed. \(P(\theta^*|f_{BB})\) is again assumed uniformly distributed and Equation (2-24) becomes

\[
\arg \max_{\theta, y_{k,i}, u_{k,i}} (P(\theta^*|\text{measurements}, f_{BB})) = \arg \min_{\theta, y_{k,i}, u_{k,i}} \left\{ \frac{1}{N} \sum_{k=1}^N \sum_{i=1}^m \left[ \frac{(\hat{u}_{k,i} - u_{k,i})^2}{\sigma^2_{k,i,u}} + \frac{n \sum_{i=1}^n (\hat{y}_{k,i} - y_{k,i})^2}{\sigma^2_{k,i,y}} + \sum_{i=1}^m \lambda_k (f_i(\hat{u}_k, \theta) - \hat{y}_{k,i}) \right] \right\}.
\]

Expressing the stationarity for the Lagrange parameters \(\lambda_k\), we finally become the Errors-In-Variables cost function [221][236]

\[
C_{EIV} = \frac{1}{N} \sum_{k=1}^N \left[ \sum_{i=1}^n \frac{(f_i(\hat{u}_k, \theta) - y_{k,i})^2}{\sigma^2_{k,i,y}} + \sum_{i=1}^m \frac{(\hat{u}_{k,i} - u_{k,i})^2}{\sigma^2_{k,i,u}} \right] \quad (2-29)
\]
with \( \hat{u}_k \) the estimated values for the true inputs and \( m \) the length of the input vector. Note that the cost function must be minimized for both the BB parameters \( \theta \) and the estimated inputs \( \hat{u}_k \), thus enlarging the parameter space with \( m \) times \( N \) additional parameters.

It should be stressed that the only way to implement stochastic properties in the model, is by choosing the proper cost function. Neither the chosen model, nor the optimization method deal with the problem of noisy measurements. Choosing the wrong cost function inevitably leads to biasing and thus to faulty model parameters.

### 2.6.3 Regularization

#### A. Weight decay

Except for the stochastic properties, other properties can be enforced by adding extra terms in the cost function. A way to reduce the risk that parameters can become infinitely large, and to prevent degeneration of the cost function, is regularization [68] [86] [177] [192]. It implements the experimenter’s prior information that the input-output mapping should be smooth.

A simple method of regularization is by using a cost function

\[
C_{R,2} = C(u_k, y_k, \theta) + \delta \| \theta - \hat{\theta} \|^2 \tag{2-30}
\]

with \( \hat{\theta} \) some nominal guess of the true model parameters, \( \theta \) the estimated parameters, \( \delta \) the regularization parameter [192], and \( C \) again any of the earlier discussed cost functions.

If no guess for \( \hat{\theta} \) can be given, it is often set to zero, i.e. \( \hat{\theta} = \theta \). This reduces the regularization to the constraint that the BB parameters must be small. In the latter case, the \( \delta \) parameter is usually called the weight decay rate. Although statistical techniques have been proposed to determine a value for the decay rate [146], the choice of the optimal value is problem dependent and remains an open question.

A way of implementing an adaptive regularization parameter is given by Andersen et al. [14]. Basically it is an implementation of a gradient descent step (see section B.) for the regualizer \( \delta \), based on the derivative \( \partial C_R / \partial \delta \). The idea behind the regularization term is that too high values for the BB parameters are punished, thus reducing the risk that two parameters, that cancel away each other, can become increasingly large during the optimization process.

The regularization given in (2-30) can easily be changed into

\[
C_{R,1} = C(u_k, y_k, \theta) + \delta \| \theta - \hat{\theta} \|_1 . \tag{2-31}
\]
Both regularization methods can be considered as Bayesian methods with a Gaussian prior in the case (2-30) or a Laplace prior in the case (2-31) [240].

B. Regularization based on Radial Basis Functions

Regularization is also reached by adding a regularization term in the cost function based on the derivatives of the BB mapping, such that the new cost function becomes

$$C_R = C(u_k, y_k, \theta) + \delta\|Df(u_k, \theta)\|^2_2$$

(2-32)

with $\theta$ the estimated parameters, $\delta$ a positive real regularization parameter [192], and $C$ is any of the above discussed cost functions. $D$ is a linear differential operator that acts on the model $f(u_k, \theta)$ and that should guarantee that the model is smooth. Although a two-norm is chosen in (2-32), other norms are possible, but less used.

For the case where $C(u_k, y_k, \theta)$ is chosen as the LS cost function $C_{LS}$, a solution for the regularization problem is described in Yee [255], Haykin [86] and Poggio and Girosi [177]. The derivation is rather lengthy and out of the scope of this thesis, such that only the results are given here.

Solving (2-32) for the MISO case in particular and choosing $C(u_k, y_k, \theta) = C_{LS}(u_k, y_k, \theta)$ leads to the Euler-Lagrange equation

$$\tilde{D}D\delta f(u) = \frac{1}{\delta} \sum_{k=1}^{N} [y_k - f(u_k, \theta)] \delta(u - u_k)$$

(2-33)

with $\tilde{D}$ the adjoint of $D$ and where $f_\delta(u)$ is the minimizing functional of the cost $C_R$ (2-32).

It is shown that the solution of the partial differential equation (2-33) can be expressed in terms of Green’s functions $G(u, u_k)$

$$G(u, u_k) = \exp\left(-\frac{1}{2\sigma_k^2}\|u - u_k\|^2\right)$$

(2-34)

where the $u_k$ are the centres and the $\sigma_k$ denote the width of the functions. Hence an optimal solution for (2-32) is given by the superposition of multivariate Gaussian functions (Radial Basis Functions) as follows

$$f_\delta(u) = \sum_{k=1}^{N} w_k \exp\left(-\frac{1}{2\sigma_k^2}\|u - u_k\|^2\right),$$

(2-35)
The linear weights $w_k$ are found by solving $w_k = (y_k - f(u_k, \Theta))/\delta$. The variances $\sigma_k$ are often chosen as $\sigma_k = \sigma$, without losing the universal approximation properties of the RBF’s.

In brief, regularization can be reached by choosing a linear superposition of Gaussians centered at the N datapoints $u_k$. The remaining problem is the choice of the variances $\sigma$. An estimate for the choice of $\delta$ is given in [86], but it is still hard to derive. In Van Gorp and Rolain [230] the variances are chosen based on an upper frequency, provided by the user. This results in an easy to use rule of thumb for the calculation of the $\sigma_k$. The resulting Gaussian network is then used for interpolation with guaranteed smoothness.

C. Weight Smoothing Regularization

Weight smoothing regularization [3] is an extension to weight decay, specifically for MLP networks with one hidden layer without bias (based on the multilayer definition given in (2-7) on page 19). It simplifies the calculation of the Jacobian matrix for the MLP

$$y = A_1 \sigma(A_2 u). \quad (2-36)$$

With $u = [u_1, u_2, ..., u_n, ..., u_m]^T$ and $y = [y_1, y_2, ..., y_k, ..., y_n]^T$ this can also be written as

$$y_k = \sum_j w_{jk}\sigma\left(\sum_i w_{ij}u_i\right) \quad (2-37)$$

where the $w_{jk}$ are elements of the matrix $A_1$, and the $w_{ij}$ are elements of the matrix $A_2$. The calculation of the Jacobian matrix for this MLP is done as

$$J_{ik}(A_1, A_2, u) = \frac{\partial y_k}{\partial u_i} = \sum_j w_{jk}\sigma\left(\sum_i w_{ij}u_i\right)w_{ij}. \quad (2-38)$$

Starting from the Tikhonov regulizer, (2-32) can also be written as

$$C_R = C(u_k, y_k, \Theta) + \delta\|BJ\|_2^2 \quad (2-39)$$

with $J$ the Jacobian matrix of the BB mapping $f_{BB}(u, \Theta)$ with respect to the inputs $u$, and $B$ determines the regularization matrix. To minimize the discrete second-derivative of the Jacobian profile, one can choose $B$ as in (2-40). To minimize the discrete third-derivative of the Jacobian profile, one can choose $B$ as in (2-41) [3].
The need of carefully choosing the cost function

\[
B_1 = \begin{bmatrix}
-1 & 2 & -1 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & 0 & \ldots \\
\vdots & & & & & \\
0 & \ldots & 0 & -1 & 2 & -1
\end{bmatrix}
\]  \hspace{1cm} (2-40)

\[
B_2 = \begin{bmatrix}
-1 & 3 & -3 & 1 & 0 & \ldots & 0 \\
0 & -1 & 3 & -3 & 1 & 0 & \ldots \\
\vdots & & & & & & \\
0 & \ldots & 0 & -1 & 3 & -3 & 1
\end{bmatrix}
\]  \hspace{1cm} (2-41)

Instead of calculating \(\|BJ\|_2\) in (2-39), Weight Smoothing calculates

\[
\|BJ\|_2 = \|B_1 J_k\|_2 \leq M^2 \|A_1\|_2 \|B A_2\|_2
\]  \hspace{1cm} (2-42)

in which \(M = \max_u \sigma(u)\) (\(M = 1\) if the hyperbolic tangent function is used for \(\sigma\)).

D. Other regularization methods

Two other regularization methods for multilayer perceptron neural networks are given in Moody et al. [155] and Weigend et al. [245]. Both methods are extensions on the weight decay method and are more calculation extensive.

Leung et al. [127] discusses the Forgetting Recursive Least Square technique (FRLS) and shows that FRLS is similar to the weight decay method. The performance of the FRLS compared with the classic Least Squares methods then depends on the forgetting factor in the algorithm, such that the problem of finding the proper regularization factor becomes the problem of finding the proper forgetting factor.

From the point of view that regularization is considered as a technique for smoothing the mapping of a BB model, adding noise can be considered as a way to do Tikhonov regularization. This is discussed in Chapter 7 on page 189. Based on noisy measurements, regularization has also been discussed in Bayesian theory (Mackay [143]-[147]).

Another way to do regularization that is reported, is the use of Double Backpropagation in [59].

2.6.4 Stability and robustness terms

Adding stability [205] and robustness [156] constraints can be done by adding boundary terms in the cost function [67]. The general idea is that, again, extra terms are added into the cost function, giving
A Scheme For Nonlinear modeling

\[ C_R = C(u_k, y_k, \theta) + \delta C_S \]  

(2-43)

with \( C \) any of the above cost functions and \( C_S \) some cost which is based on stability demands. In Chapter 5 a stability theorem is given where some model dependent stability term \( \zeta \) is demanded to be smaller than one, e.g. \( \zeta = 0.98 \). The added term in the cost function then becomes \( C_S = (\zeta - 0.98)^2 \).

The \( \delta \) term determines some balance between the two costs and is difficult to determine prior to the minimization. Choosing \( \delta \) too low can cause the stability criterion never to be reached, while choosing \( \delta \) too high can cause the optimization algorithm to put too much effort in the stability constraint, thus neglecting the performance cost \( C \). A solution is to make \( \delta \) adaptive but it is our finding that extreme care must be taken to avoid destabilization of the optimization algorithm.

The addition of the extra term in the cost function also leads to bias in the model parameters. Indeed, because of the extra term, the cost function doesn’t have a minimum in the true model parameters \( \theta^* \) anymore. In opposition to the unwanted biasing effects caused by noise, this is a controlled biasing where a degradation in performance of the cost \( C \) is allowed for in favour of some other property.

One should remind that putting the stability and robustness demands in the cost function, is just one way of implementation. More dedicated algorithms for constrained optimization exist, such that these constraints are applied on the level of optimization instead of that of the cost function. It must be noted that these algorithms, again, do nothing more than minimize an additional cost term.

2.6.5 The Bias-adjusted estimator

Amemiya [11] proposes a regularization term in the cost function to reduce the bias in a nonlinear model, due to noisy input data. This method was applied on a nonlinear model by Vandersteen [221]. The bias term is estimated and corrected in the cost function, starting from the LS cost function.

A first order calculation of the bias correction is done as follows. Assume the modeling error \( e_k = (f_{BB}(u_k, \theta) - y_k) \) based upon noisy input samples \( u_k = u_k^* + n_{k,u} \) with \( u_k^* \) the true, but unknown, input value and \( n_{k,u} \) a stochastic value with the same dimension as the vector \( u_k^* \). A first order approximation of the BB model \( f_{BB} \) is written as

\[
\begin{align*}
    f_{BB}(u_k, \theta) &\equiv f_{BB}(u_k^*, \theta) + \frac{\partial f_{BB}(u_k^*, \theta)}{\partial u_k^*} n_{k,u} + n_{k,u}^T \frac{\partial^2 f_{BB}(u_k^*, \theta)}{(\partial u_k^*)^2} n_{k,u}.
\end{align*}
\]  

(2-44)
Definition 2.2 The notation \( E\{ \cdot \} \) denotes the expectation with respect to the probability distribution of a function \( g(x) \) and is calculated as

\[
E\{ g(x) \} = \int_{-\infty}^{\infty} g(x)f_d(x)\,dx
\] (2-45)

with \( f_d(x) \) the probability density function of a stochastic variable \( x \).

A first step to determine the stochastic properties of the LS cost function for noisy inputs, is to calculate its expected value. Knowing that

\[
\frac{\partial f_{BB}(u_k^*, \theta)}{\partial u_k^*} n_{k,u} = 0
\] (2-46)

the stochastic behaviour of (2-44) becomes

\[
E\{ f_{BB}(u_k, \theta) \} = f_{BB}(u_k^*, \theta) + E\left\{ n_k^T \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} n_k \right\}
\] (2-47)

where

\[
E\left\{ n_{k,u}^T \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} n_{k,u} \right\} = E\left\{ \text{trace} \left( n_{k,u}^T \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} n_{k,u} \right) \right\}
\]

\[
= E\left\{ \text{trace} \left( \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} n_{k,u} n_{k,u}^T \right) \right\}
\]

\[
= \text{trace} \left( \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} E\{ n_{k,u} n_{k,u}^T \} \right)
\] (2-48)

and \( E\{ n_{k,u} n_{k,u}^T \} = \sigma_k^2 I_m \) in the case of uncorrelated noise contributions \( n_{k,u} \), such that

\[
E\{ f_{BB}(u_k, \theta) \} = f_{BB}(u_k^*, \theta) + \text{trace} \left( \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} \right) \sigma_k^2.
\] (2-49)

In the ideal case, the expected value of the BB functional \( f_{BB}(u_k, \theta) \) should behave as if there were only noiseless measurements, i.e. the outcome should equal \( f_{BB}(u_k^*, \theta) \). From equation (2-49) we learn that there is an added \( \theta \)-dependent term which grows with the noise variance. This term leads to bias in the model parameters.
The derivatives with respect to the true values $u_k^*$ are not exactly known. For that reason it is assumed that the derivatives of the model with respect to $u_k^*$ can be replaced by the derivatives with respect to the measured value $u_k$, i.e. it is assumed that the model has a good generalization.

Once the variance on the input measurements is known (or a $\hat{\sigma}_u^2$ is calculated, see [189]), it is possible to define a bias corrected cost function

$$ C_{BC} = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{n} \sum_{i=1}^{n} \left( f_i(u_k^*, \theta) - y_{k,i} - \text{trace} \left( \frac{\partial^2 f_{BB}(u_k^*, \theta)}{\partial u_k^{*2}} \right) \hat{\sigma}_{k,u}^2 \right)^2 $$

(2-50)

with

$$ f_{BB}(u_k, \theta) = [f_1(u_k, \theta), f_2(u_k, \theta), ..., f_n(u_k, \theta)]^T $$

(2-51)

and for which the mean value approximately equals zero in a first order approximation.

It is our observation, based on the modeling of a large number of NN models on different plants, that the occurrence of the second order derivative in the cost function leads to a large number of local minima during optimization. It is therefore advisable to use the result of an OE optimization as the starting value for the bias-corrected cost function. The examples, given in section 6.10 show that in the mean the EIV results perform significantly better than when using the LS error.

### 2.6.6 Minkowski cost function

Although the cost functions given in 2.6.2 are commonly used, they are definitely not the only error functions available. Another representation for the Gaussian distributions (2-22) and (2-23) is the form

$$ P(x) = \frac{r}{2 \Gamma(1/r) \beta^{1/r}} \exp \left( -\frac{\|x\|_r}{\beta} \right) $$

(2-52)

with $\|x\|_r$ the $r$-norm, $\beta$ controlling the variance of the distribution (e.g. $\beta = \sigma^2$ for the case $r = 2$ ) and where the gamma-function equals

$$ \Gamma(x) = \int_0^\infty u^{(x-1)} e^{-u} du. $$

(2-53)

The term $r/(2 \Gamma(1/r) \beta^{1/r})$ ensures that $\int P(x) dx = 1$. The special case $r = 2$ folds back to the Gaussian distribution used in section 2.6.2. Omitting irrelevant constants with respect to
The need of carefully choosing the cost function

the minimization, and assuming no input noise and independent identically distributed (iid) output noise, the negative loglikelihood of a measurement set becomes

\[ C_M = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left\| f_i(u_k, \theta) - y_{k,i} \right\|_r \]  

which is called the Minkowsky cost function. Apart from the case \( r = 2 \), another popular value is \( r = 1 \), leading to \( \Gamma(1) = 1 \) and

\[ C_{M,1} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left| f_i(u_k, \theta) - y_{k,i} \right|. \]  

The major advantage of this cost function is that large errors have less influence, such that the impact of outliers is reduced compared with the case \( r = 2 \). A value for \( r \) less than two reduces the sensitivity of the cost function to outliers, compared with the LS estimator, while larger values emphasize the outliers.

### 2.6.7 Choosing the cost function

From the above list, the cost functions that can be considered, are basically the LS and Minkowski cost, the WLS and the EIV cost. In each of these cost functions, extra terms can be added such as a stability constraint or a regularization term.

LS and Minkowski don’t use extra prior information except the measurements. For WLS the variance on the output measurements is needed, or must be estimated. EIV also needs the variances on the input measurements. Therefore, if a more complex cost function is chosen, more information is needed.

The occurrence of local minima heavily depends on the model chosen, but the choice of the cost function also influences the minimization. In practise a more complex cost function typically also results in more local minima. This is certainly the case for the EIV cost function. Added terms in the cost function add to the complexity and, therefore, also to the risk that more local minima exist.

One way of minimizing the risk of being trapped in local minima, is starting from a lesser complex cost function, and using the results of the optimization as starting values for a more complex cost function. This implies more subsequent optimizations and seems time consuming. The technique is successfully used in section 6.6 on page 154, when optimizing the EIV cost function.
2.7 Minimization of the cost function

2.7.1 Parameter initialization

The easiest way to find initial model parameters is by simply picking random values. The majority of the NN models are trained using such random starting parameters. For nonlinear problems this means that the optimization only guarantees that a local minimum of the cost function is reached. Often the model is optimized more than once with different starting values as a validation, and the best performing model is selected.

The ideal way of initialization is to start close to the optimal solution, using a 'good guess'. In the field of FL the initial parameters can be found using linguistic rules [181], Fuzzy Patches [119] [120] or clustering methods [241]. Many times it is believed that these methods for initialization are sufficient to guarantee a satisfying performance, and further optimization of the Fuzzy Sets is omitted. Because of the elaborated methods for parameter initialization within FL, it was recently combined with the NN field to become Neurofuzzy systems.

Another technique is the use of a simpler model or less demanding cost function prior to the demanded modeling. E.g. when using the Errors-In-Variables cost function, it is observed that the optimization suffers from many local minima [189] [221] [236] such that the initial parameters are best chosen as the results of an Output Error minimization, using the LS cost function. A popular technique is learning a linear model (with a global minimum for the model parameters). Next, these parameters are used as a starting point for a nonlinear model.

2.7.2 Select an optimization method

A. Introduction

Although the name of the optimization step within different modeling fields is often chosen differently, the idea is the same: change the model parameters such that the resulting model performs better (such that the cost function is minimal). The optimization is done by recurrently updating the model parameters

$$\theta^{(k+1)} = \theta^{(k)} + \eta \Delta \theta^{(k)}.$$  \hspace{1cm} (2-56)

$\eta$ is the optimization rate (also called learning rate within NN). In the optimal case $\eta$ equals one, but it can be taken smaller than one to ensure stability of the optimization method. A
small value for $\eta$ has the disadvantage that the optimization routine can be very slow. Therefore, the optimization rate is often made adaptive as in Listing 1.

\begin{verbatim}
WHILE stopcriterion not reached
    increase $k$
    calculate new optimization step $\Delta \theta^{(k)}$
    calculate new parameters $\theta^{(k+1)} = \theta^{(k)} + \eta \Delta \theta^{(k)}$
    calculate the new cost $C^{(k+1)}$
    IF $C^{(k+1)} \geq C^{(k)}$
        decrease the optimization rate $\eta^{(k)} = \eta^{(k-1)}/4$
        calculate new parameters $\theta^{(k+1)} = \theta^{(k)} + \eta \Delta \theta^{(k)}$
        calculate the new cost $C^{(k+1)}$
    ENDIF
    increase optimization rate
ENDWHILE
\end{verbatim}

**Listing 1.** Optimization with adaptive optimization rate

It is observed that using a faster decrease rate (e.g. $\eta^{(k)} = \eta^{(k-1)}/10$) leads to a better performance of the LM minimization scheme. The initial learning rate is typically taken very small, e.g. $\eta^{(0)} = 10^{-3}$.

The optimization rate is sometimes used to avoid local minima during the optimization. In the case that many local minima can be expected, a slight increase of the cost function is allowed for. This is done using a momentum term $mt \in [0, 1]$ [149]. A typical value for the momentum term is $mt = 0.8$. The optimization routine changes as shown in Listing 2.

Although the use of the previous optimization step $\Delta \theta$ could lead to an increase of the cost at a given iteration $k$, it is still partly used due to the momentum term.

Various methods exist for the calculation of the update vector $\Delta \theta^{(k)}$. The choice is based on robustness of the optimization method, the memory needs, speed of optimization and the ability to avoid local minima. Nearly all current methods make, in one way or another, use of the Jacobian matrix $J^{(k)} \in \mathbb{R}^{N \times P}$ with elements $J_{i,j}^{(k)}$ that are calculated as

$$J_{i,j}^{(k)} = \frac{\partial e_{i}^{(k)}}{\partial \theta_{j}}.$$  \hfill (2-57)

**Comment:** For notational reasons the index $(k)$ is omitted in the remainder of this chapter, except when it is explicitly needed to clarify the text.
The index $j$ equals $j = 1, 2, ..., P$ with $P$ the number of model parameters that must be optimized. The error $e_i$ depends on the chosen cost function. In the case that the LS cost function is used, the error would be

$$e_{k,i} = f_i(u_{k,i}, \theta) - y_{k,i}$$

with $i = 1, 2, ..., N$.

A number of commonly used optimization steps are given below. Note that only the cost function deals with properties that are related to the quality of the measurements (e.g. noise contributions), and that a clear distinction must be made between the cost function and the optimization process. As such, the optimization algorithm only minimizes the model errors of the chosen model compared with the measurements. For that reason it has no sense to speak e.g. about 'Backpropagation Neural Networks', since backpropagation is a learning scheme, while the NN is a model.

**B. Gradient Descent and Backpropagation**

The gradient descent (or Gaussian) update vector [17] [192] is defined as

$$\Delta \theta = -\frac{1}{N} J^T e.$$  

(2-59)

The error $e$ is a column vector and equals $e = [e_1, e_2, ..., e_N]^T$. 

---

**Listing 2.** Optimization with adaptive optimization rate and momentum term

```plaintext
WHILE stopcriterion not reached
  increase $k$
  calculate the new optimization step $\Delta \theta^{(k)}$
  apply momentum term $\delta \theta^{(k)} = m t \cdot \delta \theta^{(k-1)} + (1 - m t) \Delta \theta^{(k)}$
  calculate new parameters $\theta^{(k+1)} = \theta^{(k)} + \eta \delta \theta^{(k)}$
  calculate the new cost $C^{(k+1)}$
  IF $C^{(k+1)} \geq C^{(k)}$
    decrease the optimization rate $\eta^{(k)} = \eta^{(k-1)} / 4$
    calculate new parameters $\theta^{(k+1)} = \theta^{(k)} + \eta \delta \theta^{(k)}$
    calculate the new cost $C^{(k+1)}$
  ENDIF
  increase optimization rate
ENDWHILE
```

The error $e$ is a column vector and equals $e = [e_1, e_2, ..., e_N]^T$. 

---

40
Gradient methods can be rather slow, but have minimal memory needs. These low memory requirements makes the Gradient Descent step an ideal method for e.g. the Errors-In-Variables cost function because of the large number of parameters that are involved with EIV.

In order to improve the optimization speed, the learning rate can be made adaptive. The method still converges to a local minimum, but care must be taken to make sure that the use of a momentum term doesn’t lead to instability of the optimization routine.

Currently, the method is implemented for practically every existing model, due to its ease of use. For the specific case where layered models are used (as is the case for NN), the calculation of the Jacobian is often done using Backpropagation. To fix the ideas, assume that a plant is modeled using a layered BB model

\[ f_{BB} = w_1 g_1(w_2 g_2(w_3 g_3(u))) . \]  

(2-60)

The \( w_i \) are the tuneable parameters, while the \( g_i \) are linear or nonlinear activation functions with fixed parameters. The elements of the Jacobian matrix are calculated using the equations

\[
\begin{align*}
\frac{\partial f_{BB}}{\partial w_1} &= g_1(w_2 g_2(w_3 g_3(u))), \\
\frac{\partial f_{BB}}{\partial w_2} &= w_1 g'_1(w_2 g_2(w_3 g_3(u)))g_2(w_3 g_3(u)) \text{ and} \\
\frac{\partial f_{BB}}{\partial w_3} &= w_1 g'_1(w_2 g_2(w_3 g_3(u)))w_2 g'_2(w_3 g_3(u))g_3(u).
\end{align*}
\]

(2-61)

One can see that the calculation of the derivative of each layer is partially based on the derivative with respect to the previous layer, and partially on the result of the current layer. It is possible to calculate the Jacobian elements, and therefore also the parameter update vector, layer by layer, thus decreasing the number of calculations needed.

Despite the popularity of the Backpropagation algorithm, it is known to be a rather slow optimization method. An accelerated algorithm for layered models was given by Ergezinger et al. [61], making use of a linearization of the saturating activation functions.

C. Newton and Gauss-Newton

The Newton step is calculated as

\[ \Delta \theta = -H \nabla^T e \]  

(2-62)

with \( H \) the Hessian matrix with elements \( h_{i,j} = \frac{\partial^2 f_{BB}}{\partial \theta_i \partial \theta_j} \). In practise the calculation of the full Hessian \( H \) is often computational prohibitive. As a solution the Hessian is calculated as \( H \approx J^T J \), neglecting second order derivatives with respect to the first order derivatives. The resulting Gauss-Newton optimization step \( \Delta \theta \) is then calculated as
Even the use of $J^T J$ instead of the Hessian, is computational demanding, and this is certainly the case for the inversion of the matrix. Moreover, in practise the calculation of $J^T J$ should be avoided for numerical reasons. The optimization step can, therefore, be calculated as

$$\Delta \theta = -(J^T J)^{-1} J^T e.$$  \hfill (2-63)

In case that the Hessian is not positive definite, Newton and Gauss-Newton methods are known to be easily trapped into a saddle-point, or even at the maximum of a functional rather than the minimum. The method can become unstable far more easily than the simple Steepest Descent algorithm.

**D. Levenberg-Marquardt**

In order to make the Hessian matrix in the Gauss-Newton method positive definite, an identity matrix times a constant factor $\lambda$ is added to it. The resulting LM optimization step is then calculated as

$$\Delta \theta = -(J^T J + \lambda I)^{-1} J^T e$$  \hfill (2-65)

with $\lambda$ the Marquardt-factor. The need for inversion of the new Hessian matrix $(J^T J + \lambda I)$ makes that LM needs a lot of calculation power and memory. Yet, the method is very robust with respect to saddle points, which makes it a good choice for the minimization of the LS and WLS cost functions.

To avoid the calculation and inversion of the $J^T J$ matrix and for numerical reasons, equation (2-65) can in practise also be calculated as

$$\Delta \theta = -\tilde{J} \tilde{e}$$  \hfill (2-66)

with

$$\tilde{J} = \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix} \quad \text{and} \quad \tilde{e} = \begin{bmatrix} e \\ 0 \end{bmatrix}.$$  \hfill (2-67)

Remark that LM balances between the Gradient Descent step and the Gauss-Newton step, dependent on the Marquardt-factor. This balancing is a dynamic process during the optimization. The optimization routine is shown in Listing 3.
choose initial Marquardt factor, e.g. $\lambda = 10^{-4}$

**Listing 3.** Optimization with Levenberg-Marquardt

Note that the Marquardt factor is decreased at a slower pace than when it is increased. In practise, it is observed that this can lead to a faster convergence.

It is the experience of the author that the Levenberg-Marquardt optimization method is extremely robust. This was also concluded by Ngia and Sjöberg [163] who state that in off-line applications the Levenberg-Marquardt algorithm is often superior to other training algorithms (more precise: steepest descent and Gauss-Newton), in a way that it provides faster convergence. The increased number of calculations needed for LM are easily compensated by the increase in optimization speed.

**E. Hebbian Learning**

Hebbian learning originally stems from the NN domain. It is specifically designed for layered models with saturating activation functions in the layer. Hebbian learning goes one step further than Backpropagation where the layers are learned separately, by learning each node in the layer on its own. A node is considered to use the output of other nodes as its input, and has different outputs to other nodes, as shown in FIGURE 2-4.

For each node in a layered model, the adaptation of the parameters is done using the optimization step
A Scheme For Nonlinear modeling

\[
\Delta \theta_i = y_i y_{i-1}^T \quad (2-68)
\]

with \(y_i\) the outcome vector with size \(n_i \times 1\) of the node and \(y_{i-1}\) the outcome vector with size \(m_i \times 1\) of the nodes of the previous layer that are connected to this node. The use of the optimization rate in (2-56) is obligatory, such that the new parameter value for a particularly node becomes

\[
\theta_i^{(k+1)} = \theta_i^{(k)} + \eta y_i^{(k)} y_{i-1}^{T(k)}. \quad (2-69)
\]

The idea behind this optimization step is that nodes that interact a lot are strengthened.

If linear activation functions would be used in the layers, it is clear that this routine leads to an exponential growth of the connection matrices between different layers. Therefore the optimization step can be changed into

\[
\Delta \theta_i = (y_i - \bar{y}_i)(y_{i-1} - \bar{y}_{i-1})^T \quad (2-70)
\]

where \(\bar{y}_i\) and \(\bar{y}_{i-1}\) are the mean values of \(y_i^{(k)}\) and \(y_{i-1}^{(k)}\) with respect to \(k\). It can be seen that changes in the parameters only take place whenever the inputs and outputs have changed.

The advantage of Hebbian learning is its ease of use and high speed for large and complex interconnected models, since there is no need to inverse the whole parameters space at once. Moreover, the learning is performed asynchronously for all nodes. The main drawback is obviously the potential for instability in the optimization process. This is certainly the case for large optimization rates, such that a slow optimization is preferred.

A few variants on the Hebbian learning rule exist (e.g. in Fausett [63]), but the general theme where each node in the model structure is optimized independently, remains.

**F. Monte Carlo, Metropolis, Simulated Annealing**

The Monte Carlo scheme bases the optimization step on a random walk, where \(\Delta \theta\) is a random vector with a small variance. The update is only performed if the cost function decreases [26]. In the Metropolis variant of the scheme, the optimization step is accepted also when the cost function \(C\) increases. The acceptance is based on a probability function \(p\), e.g.
Minimization of the cost function

\[ p = \frac{P(\theta^{(k+1)}|\text{measurements})}{P(\theta^{(k)}|\text{measurements})} \]  

(2-71)

where \( P(\theta^{(k)}|\text{measurements}) \) is defined as in the Bayes theorem of equation (2-21) on page 28. In the particular case where the LS cost function is used based on the loglikelihood (2-25) on page 28, the acceptance of the parameters \( \theta^{(k+1)} \) when the cost \( C^{(k+1)} \) is larger than the cost \( C^{(k)} \), is based on the probability

\[ p = \exp(C^{(k)} - C^{(k+1)}). \]  

(2-72)

Although much better than the simple Monte Carlo method, the Metropolis algorithm is still very slow for complex networks [160].

Both methods are known to be trapped in local minima easily. For that reason the Metropolis variant is often changed into Simulated Annealing [86]. A temperature \( T^{(k)} \) is defined with a high initial value \( T^{(0)} \) and the acceptance of a higher cost function is based on a probability

\[ p = \exp\left(\frac{C^{(k)} - C^{(k+1)}}{T^{(k)}}\right). \]  

(2-73)

The algorithm easily breaks out of local minima whenever the temperature is high. During optimization the temperature is typically decreased exponentially, e.g.

\[ T^{(k+1)} = \alpha T^{(k)} \]  

(2-74)

with \( \alpha < 1 \) (typically between 0.8 and 0.99).

G. Stability of the optimization algorithm

In everyday use, there is only very seldom a need to prove that the optimization scheme used, is stable. The stability could be an issue, however, whenever a controller is optimized in situ. More and more controllers are required to provide an automated means of self-tuning by a simple push on the button. The controller then adapts its parameters until a new minimum in error is reached.

From the theoretical point of view, gradient descent optimization algorithms are guaranteed to converge to a local minimum of the cost function [38]. This makes them an ideal candidate for self-tuning controllers.

H. Conclusion

The list given in the sections above is not exhaustive but gives a good idea about the large number of optimization techniques in use. The use of error surfaces, Conjugate Gradients,
Scaled Conjugate Gradients [26], more complex Gradient Descent methods [17] [173] [175], Boltzmann Learning [86], the Delta Rule [16], Line Search [86] [153], Linear Quadratic Programming, or even the use of Genetic Algorithms or Reinforcement Learning have not been addressed in detail in this section.

Some of these optimization methods are dedicated to a very special type of model, or adapted for the use with a specific cost function. Nevertheless, all of these optimization routines perform the same action: minimize a cost function, based upon given measurements and a given model.

2.7.3 Choose the stopping criterion

The decision when to stop the optimization can be taken apart from the optimization routine itself. The optimization can just be stopped when the cost function reaches its minimum. When using noisy measurements with black box modeling, however, the risk of overfitting is eminent and stopping is not necessarily done when the cost function reaches its minimum. Black box models fit any data, so also noisy data and outliers in the measurements. In the case of spurious data, BB models can take any form in the measurements gaps, provided that the model has enough free parameters.

Within linear identification theory the condition number (the ratio of the largest to the smallest singular value) of the Jacobian matrix should not exceed the software calculation precision. This is done for numerical reasons. Within BB modeling this rule of thumb is easily broken, due to severe overparametrizing.

The reason for choosing a high number of parameters lays in the demand for good generalization, such that details in the data can be mapped. A too low number of parameters involves a risk that certain features in the measurement data are not mapped. Yet, even for a low number of parameters it might be impossible for the optimization routine to direct inactive parameters to these unmapped features due to local minima in the cost function. The result is that even in the case of a low number of BB parameters, some parameters can remain ineffective for the learning data, but could cause severe problems when the BB model is applied on new data.

Therefore, some way must be found to handle overfitting. There are a number of methods described in the literature. For example model selection methods, which are hardly possible for BB models, and regularization methods (see also section 2.6). A popular way is the use of a validation set, which is also called early stopping [9] [193] [146] or stopped training [187] [219]. The measurement data is split into three parts: the learning set, the validation set and the
test set. The optimization of the BB parameters is performed on the learning set, while monitoring the validation set. Optimization is stopped when the error on the validation set increases. This technique is well known in the field of NN, but as suggested in FIGURE 2-1 it is also possible to use early stopping for other models.

Mackay [146] states that instead of early stopping, one should resort to the use of a regularizer in the cost function. Sjöberg [192], however, showed that early stopping can be seen as a means of an implicit regularization, while Amari [9] proves that, when comparing early stopping to the optimal stopping time, both in the asymptotic as in the nonasymptotic case the increase in error is small.

The drawback of early stopping is the need for a validation set. In the particular case of sparse data, this can lead to a lack of measurement data for optimizing. Sometimes the validation set is just a single sample (leave-one-out technique) and a large number of models are mapped, with a different validation sample for each model. When using this technique, it can very well happen that the model, used on the test set, performs far better than when used on the learning set or validation set. This is quite atypical and is caused by the variance on the model, due to variance on the measurement data. The use of the leave-one-out technique is therefore highly questionable when used with noisy data.

More sensible techniques of choosing a proper validation set are based on Vapnik-Chervonenkis (VC) dimensions in [113] and on statistical theory in [9]. In his excellent paper Amari [9] shows that, under mild conditions, the size of the validation set should be no larger than

\[
M = \frac{1}{\sqrt{2N}} \cdot 100\% \quad (2-75)
\]

with \(M\) the size of the validation set in percent and \(N\) the size of the whole measurement set. If the size of the whole measurement set equals \(N = 500\), then less than 3.2\% of the set, i.e. 16 samples, are sufficient for the validation set with only a very small increase in modeling error.

### 2.8 Model validation

Despite of the measurements made, and hours of optimization, no guarantee can be given that the resulting model complies with the users’ needs. Therefore the model must be evaluated.

A first step in this evaluation is the validation of the model. As mentioned in the previous section, the model can be validated
A Scheme For Nonlinear modeling

using a third measurement set, called the test set. The test set did not contribute in the optimization scheme, while this is not the case for the learning and validation sets. The test set should, therefore, only contain independent data. In many cases it is observed that using the test set on the model, results in a model error that is distinctively larger than was the case with the validation error. The choice to reject or keep the final model can therefore only be based on a test set.

A second step in the evaluation could be an analysis of the stability of the model. A separation must be made between absolute and relative stability. The latter is based upon phase or amplitude margins and is studied in detail for linear systems. To date, only absolute stability is available for nonlinear systems. Most stability theorems for nonlinear systems make use of a Lyapunov function [86] and expect the nonlinear function to be sector bounded [205]. The resulting stability theorem is usually too conservative for practical use and better theorems are yet to be found. A method to partially overcome conservatism is given by NL_q theory [207], applied on NN. The stability of NN is also studied in [211]. FL systems are studied in [32] [109] [120] and [254]. The stability of systems with saturation is studied in [93] and [139], and the stability using LMI’s in [114], [139] and [212].
Abstract: in this chapter special interest is shown in the Neural Networks modeling technique. The NN models used in today's engineering applications have a very general structure which allows for use with a wide variety of nonlinear functions. In practice, some other BB modeling techniques (RBF networks, FL systems, ...) can also be described using some NN topology.

This chapter gives an overview of the most common NN models and places them in the more general modeling framework. It will be shown that this is very well possible, even taking into account the large number of Neural networks that have been described in literature.
3.1 The neural network basic model

3.1.1 Introduction

The first Neural Networks were developed in the early 1940s, based upon knowledge about the human brain architecture. This research was done by Warren McCulloch and Walter Pitts, who are quite generally regarded as the inventors of the NN model. Their model, named after the inventors, included a nonlinear activation function in the neuron and a threshold such that the neuron only fires if the input is larger than the threshold. The model was soon followed by the Hebb network.

The growing interest in NN led to the golden ages in the 1950s and 1960s, giving birth to a large number of models, such as the Perceptron and Adaline architectures. The majority of the NN models that are currently used were actually invented in those years. The main part of the work was still done in the field of psychology, and after the big rush in the early years, research was quite a bit less in the 1970s. The development of NN hadn’t led to a revolution in understanding the human brain or real artificial intelligence. The fact that NN couldn’t solve the XOR problem, the absence of optimization algorithms and the lack of real applications lead to a disbelief in the capabilities of NN.

Widrow and Hoff introduced the NN model into electrical engineering. A more mathematical approach to NN started in the 1970s, but only gained success in the early 1980s. The early work came from Kohonen, Anderson, Carpenter but especially Grossberg who wrote many mathematical and biological papers on the subject. In the 1980s more work was done on learning algorithms (e.g. the Backpropagation algorithm), new network topologies [38] and the universal approximation properties of neural networks [73] [74] [97] [249]. The Hopfield and Neocognitron networks and the Boltzmann machine were invented, and a lot of work was done in the field of combining neural nets with radial basis functions.

The number of applications are growing, but NN still have to fight for a place under the sun. The combination of fuzzy logic with NN was promising, but the interest died out in favour of FL applications. To date, a lot of work should still be done on stability-based design rules for NN models. Although the NN model can profit from a growing interest in engineering, it still isn’t considered as a standard solution for today’s modeling problems. As a result, true end user applications are still largely missing.

This thesis deliberately does not give the description of a NN, based upon the human neuron architecture. On the contrary, a NN is given explicitly as a mathematical function to stress the fact that the NN properties come from just a simple nonlinear input/output relationship. The
following sections place the NN architectures most used within the general framework of Chapter 2.

An extension of NN is the use for dynamic systems and control, where some memory is needed to model the time dependency of the model. Isermann [102] distinguishes two ways to introduce memory into the network: external and internal dynamics. The case of external dynamics leads to the use of NARX and NFIR models (defined further) and the time delay neural networks, while partially or fully recurrent networks inherently have some internal dynamics. Both methods are explained more in detail in the sequel.

3.1.2 The McCullogh-Pitts neuron

The McCullogh-Pitts neuron is probably the first NN model described. It has a binary nonlinear relationship, weights that describe the importance of an interconnection between the neurons, and a threshold that keeps a neuron from firing. An input $u_{i,j} \in \{0, 1\}$ of the $i$-th neuron $\mathbf{u}_i = [u_i^+, \mathbf{u}_i^-]$ is either excitatory or inhibitory. The excitatory inputs are defined as $u_i^+ = [u_{i,1}, u_{i,2}, \ldots, u_{i,n_i}]$. Any of the $m_i$ elements of the inhibitory inputs $u_i^- = [u_{i,n_i+1}, u_{i,n_i+2}, \ldots, u_{i,n_i+m_i}]$ prevent the neuron from firing, regardless of the other inputs. The transfer function of a single neuron $y_i = f_{MCP}(\mathbf{u}_i, u_i^+, w_i)$ with $w_i$ the weight of the neuron, can then be written as the logical relationship

$$y_i = \begin{cases}
1 : (w_i \sum_{j=1}^{n_i} u_{i,j} \geq \kappa_i) \text{ and } \left( \sum_{j=n_i+1}^{n_i+m_i} |u_{i,j}| = 0 \right) \\
0 \text{ : else}
\end{cases} \tag{3-1}$$

with $\kappa_i$ the threshold value for the neuron and the output vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]$.

Remark that all inputs have the same weight, such that the first part of the logical relationship can be replaced by $\gamma_i \sum_{j=1}^{n_i} u_{i,j} > 1$ with $\gamma_i = w_i / \kappa_i$, thus using the same nonlinear transfer function for all neurons. An inhibitive input can be given a large negative weight $-N$ with $N > \max_i (\gamma_i n_i - 1)$, such that the transfer function for a neuron becomes

$$y_i = \begin{cases}
1 : ([\gamma_i \mathbf{1}_{1 \times n_i} + -N \mathbf{1}_{1 \times m_i}] [\mathbf{u}_i^+ - \mathbf{u}_i^-]^T) \geq 1 \\
0 \text{ : else}
\end{cases} \tag{3-2}$$

The output $y_i \in \{0, 1\}$ can be used as an input to other neurons. There is no need for a strict layered structure. The typical transfer function of a McCullogh-Pitts neuron is given in FIGURE 3-1.
3.1.3 Hebb nets [63] or Rosenblatt’s perceptron [86]

The Hebb NN model only has one layer of neurons. The output \( y = [y_1, y_2, ..., y_R]^T \) is modelled using \( R \) neurons. Each neuron \( y_i = f_H(u_i) \) has a single output and is based upon \( m \) inputs \( u_j \) plus a bias term \( b_i \) which could be considered as an extra input that always equals one. All inputs have a different weight \( w_{i,j} \). The input-output relationship becomes

\[
y_i = \begin{cases} 
1 : \sum_{j=1}^{m} (w_{i,j}u_j) + b_i \geq \kappa_i \\
0 : \text{else}
\end{cases} 
\]  

(3-3)

Remark that the output has a binary state. Not all of the inputs need to be used. In the formulation of equation (3-3) this can be enforced by setting some of the weights to zero.

Also in this case it is possible to eliminate the threshold \( \kappa_i \). The parameter vector to update \( \theta_i = [\gamma_{i,1}, \gamma_{i,2}, ..., \gamma_{i,m}, \beta_i]^T \) is then formed by the resulting weights \( \gamma_{i,j} = w_{i,j}/\kappa_i \) and biases \( \beta_i = b_i/\kappa_i \) and the transfer function (3-3) becomes

\[
y_i = \begin{cases} 
1 : (\lfloor u_1, u_2, ..., u_m, 1 \rfloor \theta_i) \geq 1 \\
0 : \text{else}
\end{cases} 
\]  

(3-4)

Remark that this transfer function can be simplified further by putting the threshold in the bias vector and demand that the inequality with respect to one becomes an inequality with respect to zero. The parameter vector is typically optimized using the Hebbian optimization scheme. Using equation (2-68) this means that the parameters are updated with the parameter update vector
Hebb nets are typically used for digital valued identification, such as pattern recognition. To overcome some drawbacks of the Hebb net, such as poor generalization properties, the output is often allowed to become negative, such that the nonlinear relationship is altered to a \( \text{sign}(\cdot) \) function instead of the \( \text{larger than} \) relationship.

### 3.1.4 The Adaline and Madaline models

The name Adaline was given by Widrow and his students to the Adaptive Linear Neuron. Despite its name, the neuron is actually a Hebb NN model with a transfer function

\[
\Delta \theta_i(t) = y_i(t)[u_1(t), u_2(t), ..., u_m(t), 1]^T. \tag{3-5}
\]

\[
y_i = \text{sign} \left( \sum_{j=1}^{m} (w_{i,j}u_j) + b_i - \kappa_i \right) \tag{3-6}
\]

or, with an alternate bias,

\[
y_i = \text{sign} \left( \sum_{j=1}^{m} (w_{i,j}u_j) + \beta_i \right). \tag{3-7}
\]

Adaline NN models are configured as single layer networks. The main difference between the Hebb net and Adaline models is therefore the transfer function of the nonlinear part. The reason why they are treated differently is the way how the parameter vector is updated. While Hebb nets are optimized using the Hebbian optimization method (see section 2.7.2), the parameters of the Adaline model can be optimized using gradient methods with a LS cost function \[18\].

As a stopping criterion the first derivative with respect to the number of iterations is chosen. If the decrease in cost drops below a user defined value, the cost function is assumed to be in the local minimum and the optimization is stopped.

\[
\Delta \theta_i(t) = y_i(t)[u_1(t), u_2(t), ..., u_m(t), 1]^T. \tag{3-5}
\]
Adaline neurons can also be placed in a multilayer architecture, in which case the NN model is called a Madaline network.

3.1.5 The Multilayer Perceptron model

With no doubt the Perceptron NN model is the most used NN architecture to date. Originally the Perceptron model contained three layers of neurons: the sensory units, the associator units and a response unit. In later years Perceptron models with only two layers were proven to have universal approximation capabilities. Using two layers only has the added advantage that derivatives are less complex and the programming of the optimization process becomes easier. The nonlinear transfer function can be binary, as for the McCulloch-Pitts and the Hebb networks, but choosing a transfer function with finite higher order derivatives makes the optimization of the parameters easier when using gradient optimization methods. A Perceptron includes a bias term and weights and its transfer function is written as

\[
y_i = \sigma\left(\sum_{j=1}^{m} w_{i,j}x_j\right) + w_{i,m+1}\cdot
\]

with \( y = [y_1, y_2, ..., y_n]^T \) the output vector, \( w_{i,j} \) the weights and \( w_{i,m+1} \) the bias term. The parameter vector contains all weights and biases. A popular nonlinear activation function is the sigmoid function

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

or the hyperbolic tangent function

\[
\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]

for the case where the output is allowed to be negative. Other transfer functions that can be found in the literature regularly are the hard-limit functions \( \sigma(x) = \text{sat}(x) \) (defined in (5-4) on page 111) \( \sigma(x) = \text{sign}(x) \) and \( \sigma(x) = (x \geq 0) \), and RBF functions. When used in a multilayer configuration, the outputs of one layer act as the inputs for the next layer. A three layer Perceptron is then defined as

\[
y = \sigma_1(W_1 \sigma_2(W_2 \sigma_3(W_3 u + b_3) + b_2) + b_1)
\]

with

\[
W_1 = [w_{1,1}^T, ..., w_{1,n_1}^T]^T, \quad W_2 = [w_{2,1}^T, ..., w_{2,n_1}^T]^T \quad \text{and} \quad W_3 = [w_{3,1}^T, ..., w_{3,n_2}^T]^T
\]
the weight matrices, \( n_1 \) and \( n_2 \) the number of neurons in the first and second layer respectively and \( \mathbf{u} = [u_1, u_2, ..., u_m]^T \) the input vector. The bias vectors equal

\[
\mathbf{b}_1 = [b_{1,1}, ..., b_{1,n_1}]^T, \mathbf{b}_2 = [b_{2,1}, ..., b_{2,n_2}]^T \text{ and } \mathbf{b}_3 = [b_{3,1}, ..., b_{3,n_2}]^T. \quad (3-13)
\]

The parameter vector that must be updated then equals

\[
\mathbf{\theta} = [\mathbf{W}_1(:,); \mathbf{b}_1(:,); \mathbf{W}_2(:,); \mathbf{b}_2(:,); \mathbf{W}_3(:,); \mathbf{b}_3(\cdot)]. \quad (3-14)
\]

Remark that the weights are given in capitals here to indicate that each layer is a MIMO configuration.

There is an ongoing discussion whether the NN model (3-11) should be called a three-layer Perceptron or a two-layer Perceptron, thus only counting the "hidden" layers. In this thesis the convention is followed that each layer of neurons is counted and consequently the formulation three-layer network is adopted.

The simple one-layer Perceptron is usable e.g. for pattern recognition. Multilayer Perceptron models have been used to model a large number of nonlinear plants, typically with a soft nonlinear relationship between inputs and outputs. The modeling capabilities of MLP have been investigated by many authors. Basically any continuous hypersurface can be uniformly approximated to within an arbitrary accuracy, provided that there are a sufficient number of hidden neurons in the network, and enough measurement data in the whole region of interest is present to optimize the parameters.

3.1.6 Radial Basis Function NN

It would be possible to include the RBF neural network when describing the MLP because of the similarities between a RBF and the two-layer MLP. RBF networks, however, are currently widely used and studied such that a separate discussion seems justified.

The RBF network is typically configured in a two-layer structure. Each node in the hidden layer has a centre \( c_i \) that is compared with the input \( \mathbf{u} \) by a norm \( \|\mathbf{u} - c_i\| \), and a width \( \rho_i \). All hidden neurons are linearly combined in the output layer. The transfer function

![Perceptron neuron](image)
Neural Network systems

$f_{RBF,i}(\|u - c_i\|, \rho_i)$ of each neuron in the hidden layer can have different forms. Most commonly it is the same as the Gaussian function defined in section 2.5:

$$f_{RBF,i}(\|u - c_i\|, \rho_i) = \exp\left(\frac{\|u - c_i\|^2}{2\sigma_i^2}\right)$$

(3-15)

where $\rho_i$ equals $\sigma_i^2$. Some other typical choices of $f_{RBF,i}$ are the thin-plate-spline function

$$f_{RBF,i}(\|u - c_i\|, \rho_i) = \|u - c_i\|^2\log (\|u - c_i\|)$$

(3-16)

with $\rho_i \equiv 1$, the multiquadratic function

$$f_{RBF,i}(\|u - c_i\|, \rho_i) = \sqrt{\|u - c_i\|^2 + \rho_i^2}$$

(3-17)

or the inverse multiquadratic function

$$f_{RBF,i}(\|u - c_i\|, \rho_i) = (\|u - c_i\|^2 + \rho_i^2)^{-1/2}.$$  

(3-18)

The way how the neurons are combined, is shown in FIGURE 3-5. It is easy to see that the RBF input-output relation

$$y = \sigma_1(W_1\sigma_2(W_2u + b_2) + b_1)$$  

(3-19)

with $\sigma_1(x) = x$ and $\sigma_2(x) = f_{RBF}(x)$, has the same form as the MLP model (3-11). The approximation capabilities of RBF networks are alike to those of more general MLP networks, independent of the choice of the transfer function $f_{RBF}$. This suggests that the selection of the nonlinearity function $\sigma_i$ is not critical for the performance. It has a severe effect, however, on the number of local minima when the cost function is optimized.
3.1.7 Recurrent NN models

All of the above NN models can be considered as feedforward networks, i.e. whenever neurons are coupled, this is done in a layered or grouped manner. No cyclic paths exist in the network topology. This is no more true in the Recurrent NN model, where each neuron is coupled to some or all other neurons. The result is a recursive structure in which a form of storage is possible. A drawback is that the NN can become unstable, e.g. caused by too large interconnection weights.

One of the simplest recurrent networks is known as the linear autoassociator. All neurons within the NN are connected to each other. No exogeneous inputs are applied, and one or more of the neurons are observed as an output. The transfer function of the linear autoassociator is

\[ y(t) = W y(t-1). \]  

(3-20)

The weight matrix \( W \) is symmetric, i.e. \( w_{i,j} = w_{j,i} \), and usually the output of a neuron is not connected to its own input, i.e. \( w_{i,i} = 0 \). The weights are initialized in a random way and the whole network is evaluated at discrete time steps. This evaluation can be the correlation of the NN output with a demanded output (e.g. for pattern recognition), or just an observation of the behaviour.

It is clear that, when left on its own, the behaviour of a recurrent network easily becomes unstable. In the Brain-State-in-a-Box version, a nonlinear function of a saturating nature is applied in each neuron. The transfer function then becomes

\[ y(t) = \sigma(W y(t) - 1) \]  

(3-21)

with \( \sigma \) e.g. the function \( \sigma(x) = \text{sign}(x) \). BSB networks allow a nonsymmetrical weight matrix \( W \) and also the self-connection \( w_{i,i} \) doesn’t need to be zero.

In case of a Discrete Recurrent Hopfield network the nonlinear function is chosen as \( \sigma(x) = (x \geq \kappa) \) with \( \kappa \) the treshold vector. The weight matrix is considered symmetric with zero diagonal elements, but exogeneous inputs are allowed. The transfer function becomes

\[ y(t) = \sigma(W y(t) - 1 + V u(t)) \]  

(3-22)

with \( u \) the exogeneous inputs. In the case of Hopfield networks each element in the weight matrix is optimized asynchronously. This asynchronous update, together with the zero diagonal elements in the weight matrix, guarantee that a Lyapunov functional can be minimized such that the optimization of the network is BIBO stable [63].
Different activation functions can be used for the Hopfield network. An activation function much used is the $sat(\cdot)$ function. It is possible to use the inputs $u$ to initiate the network and choose the $V$ matrix as the null matrix after the first optimization iteration.

Hopfield networks can be used in a multilayer structure, where the output of each layer is fed back to its input, and the inputs are continuously applied. Networks in this configuration and with a sigmoid activation function typically, are called Elman networks. It is common to use linear neurons in the last layer of the network. The topology of the network is shown in FIGURE 3-7.

Specific optimization and analysis schemes exist for recurrent NN, e.g. based on unfolding in time [81]. This unfolding is easier done for locally recurrent NN, such as Elman networks, and is explained more in detail further.

### 3.1.8 Kohonen networks

The Kohonen network is an example of a topology-preserving map. This means that the behaviour of a neuron is related to placement of the neuron in the network: the weight matrix has a physical property. The network is based upon the organization of the human brain, where

---

**FIGURE 3-6** Hopfield NN model.

**FIGURE 3-7** Elman NN model.
only neurons that lie close to each other, have interconnection matrices. The neighbourhood of a neuron is designated by a radius \( r \in \{0, 1, 2, ...\} \) where the neurons are organized in a two-dimensional matrix structure. Each neuron \( i \) with \( i = 1, 2, ..., R \) has a transfer function \( y_i = f_{NN}(u, \theta_i) \), where \( u = [u_1, u_2, ..., u_m] \) is the exogeneous input, \( \theta_i \) the parameter vector containing the weights and bias of the specific neuron, and \( f_{NN} \) is any of the neuron transfer functions discussed in the previous sections. Each neuron is tested separately against a measured output \( y \). This is typically done with a LS cost function

\[
C_i = (f_{NN}(u, \theta_i) - y)^2.
\]

(3-23)

The neuron with the lowest cost \( C_i \) is selected, and not only this neuron but also all neurons within a neighbourhood are updated using a gradient descent optimization step typically. The Kohonen topology is shown in FIGURE 3-8. Not all interconnections are drawn, and the neighbourhoods of a neuron are shown for radii two, one and zero. The optimization routine starts with a large radius and then decreases the radius size when optimizing the network with a sequence of measurements.

![FIGURE 3-8 Kohonen NN model.](image)

The idea behind the Kohonen architecture is that clusters in the network specialize in mapping specific features in the input data. Kohonen maps have been used for e.g. character recognition and the travelling salesman problem. The model given in FIGURE 3-8 can also be used in a multilayer topology.

### 3.1.9 Other architectures

A less used architecture is the **Bidirectional Associative Memory**, invented by Kosko. Two layers of neurons are used, and each layer has its own exogeneous inputs. All neurons of one layer are connected to all neurons of the other layer, but no interconnections exist within one layer. Several variations on this NN topology exist, but all have the same two-layered structure. Again, different nonlinear transfer functions can be used within each neuron, leading...
Neural Network systems

to e.g. the discrete BAM (binary transfer function) and continuous BAM (sigmoid transfer function) configurations.

The *Maxnet* NN has a regular Hopfield structure, but bases its output on the largest input only. The *Mexican Hat* variant has neurons that only allow interaction with neurons that lie in a neighbourhood. Moreover, neurons that lie symmetric to a specific neuron get the same weight, but a different sign. *Hamming nets* mix both techniques: neurons also base their output on the neurons that lie close by only, and then selects the neuron with the largest output. The optimization of the net is based upon an LS cost function.

*Counterpropagation* networks are multilayer networks used for function approximation. They are based on lookup tables that are optimized in two phases. First the input layer is clustered and then the other layers are optimized. By nature, counterpropagation networks resemble fuzzy logic systems with local-linear fuzzy sets.

*ART maps* resemble Kohonen networks, but are more elaborated in a way that a user can control the degree of similarity of inputs that are correlated with the same cluster of neurons. ART1 was designed for digital data, while ART2 also accepts continuous data. As is the case with many other NN architectures, ART maps were developed using their proper optimization scheme based upon a designated cost function (in the case of ART, the norm of the input vector).

Starting from a multilayer Perceptron architecture with either continuous or binary transfer functions, a number of NN topologies was invented where the only difference is the optimization scheme used. *Boltzmann machines* have a regular single layer or multilayer architecture with binary outputs, but use the Simulated Annealing optimization scheme when optimizing the model parameters. *Gaussian machines* use a sigmoid function as the activation function and Gaussian distributed noise when optimizing with simulated annealing. *Cauchy machines* are based on Cauchy noise and allow for more noise used in the optimization scheme.

The *neocognitron* net was developed in the field of the recognition of handwritten numericals. It has nine layers made with a large number of neuron arrays. The interconnections between the arrays are sparse: every array in one layer is connected to a limited number of arrays of a previous layer only. The input of the net is a $19 \times 19$ bitmap with a handwritten numeral, while the net has 10 output neurons that are used as indicators of the numerals 0, ..., 9. Because of the large number of neurons involved, the parameter optimization of the net is typically done layer by layer, fixing the parameters of the other layers in the meantime.
A lot of other architectures were studied in the decades of NN research. Basically, the differences between all of these NN models can be summarized as follows:

- A different topology how the neurons are interconnected. This includes the choice whether neurons are auto-connective or not, and the choice of using exogeneous inputs.
- A different nonlinear activation function.
- The choice of a fixed cost function.
- The choice of a fixed optimization scheme.

Some NN models are named differently only because another optimization routine was used (e.g. Backprop NN, Boltzmann and Cauchy machines), or because specific cost functions were used (e.g. Hamming nets). The proliferation of neural networks "architectures" is one of the reasons why NN research appears so obscure and doesn’t get the attention it earns. Yet, the real differences between NN models are small once the model is separated from the optimization process.

### 3.2 Neural Networks for identification of nonlinear dynamic systems

For identification we consider the case where the plant is observable and where input-output measurements are available. We do, however, not consider the case where the plant is strongly observable, so that the internal states of the plant are unmeasured.

Consider a dynamic system with exogeneous inputs \( u(k) \in \mathbb{R}^m \) and an output \( y(k) \in \mathbb{R}^n \). Dependent on the kind of inputs used, two types of models can be distinguished: parallel and series-parallel models.

- Parallel model: the output of the model itself is used to create the time-lagged inputs. This model can be considered as a fully recurrent model. The parallel model is able to give predictions over a short period of time. The model is said to have \textit{internal dynamics}.

- Series-parallel model: the outputs of the actual system are used as inputs to the model. Only a one-time ahead prediction is possible. The model is said to have \textit{external dynamics}.

Both models are shown in FIGURE 3-9 In both cases the prediction error of the model, compared with the true plant outputs are used as a measure to optimize the model parameters.
For dynamic systems and for controllers, the model must have some way to implement time lags. In other words: some memory function must be present in the model. In NN modeling this can be done twofold: either delayed inputs and outputs are used as extra external inputs, or some memory is included in the individual neurons.

**External dynamics**

The use of the series-parallel model is the easiest from the modeling point of view. The model is just implemented with more inputs and optimized using the input-output measurements. Assume a NN model $f_{NN}$ that produces an output $\hat{y}(k) \in \mathbb{R}^m$ based on an input $u(k) \in \mathbb{R}^n$ and the zero mean stochastic noise contributions $n(k) \in \mathbb{R}^m$ that denote the process noise. The prediction error is then defined as $e(k) = y(k) - \hat{y}(k)$. Denote $N_y$, $N_u$, $N_e$ and $N_n$ as the time lags of the model output, control input, model error and noise respectively. Depending on the time lagged inputs that are used for the model, we can use the NFIR (see Glossary for abbreviations)

$$\hat{y}(k) = f_{NN}(u(k), u(k-1), ..., u(k-N_u), \theta),$$  

(3-24)

TDNN or NARX

$$\hat{y}(k) = f_{NN}(u(k), u(k-1), ..., u(k-N_u), y(k-1), y(k-2), ..., y(k-N_y), \theta),$$  

(3-25)

NARMA

$$\hat{y}(k) = f_{NX}(y(k-1), y(k-2), ..., y(k-N_y), n(k-1), n(k-2), ..., n(k-N_n), \theta) + n(k)$$  

(3-26)
or NARMAX

\[
\hat{y}(k) = f_{NN}\left( u(k), u(k-1), ..., u(k-N_u), y(k-1), y(k-2), ..., y(k-N_y),
\begin{array}{cccccc}
n(k-1), n(k-2), ..., n(k-N_n), \theta
\end{array}\right) + n(k)
\]  

(3-27)

approach. Starting from these dynamic models, we find ourselves back in the general identification scheme of Chapter 2: the NFIR, NARMA and NARMAX can be seen as Multi-Input nonlinear models.

Remark that in the case of external dynamics, no feedback from the NN outputs to the NN inputs is present in the model, once the parameter optimization is finished. The advantage is that the model is unconditionally input-output stable. The drawback is that a large time lag \(N_u\) is needed for an accurate modeling of the dynamic effects. As a rule of thumb, \(N_u\) can be chosen according to the rise time of the process, typically \(N_u = 10T_s\) to \(N_u = 40T_s\) where \(T_s\) is the sampling period [102].

For the NARMA and NARMAX models, delayed noise terms are included and these are generally unmeasured. The solution to this problem is to replace the \(n_i\) by the predicted noise terms \(\hat{n}_i\) in the identification process [205]. The next step is to choose a cost function and start optimizing the model. Mistry [152] uses a least squares error approach to learn a NN model with noiseless data. For noisy input data, however, one should use the EIV cost function.

**Internal dynamics**

Dependent on the time lagged inputs that are used for the model, we can use the NOE

\[
\hat{y}(k) = f_{NA}(u(k), u(k-1), ..., u(k-N_u), \hat{y}(k-1), \hat{y}(k-2), ..., \hat{y}(k-N_y), \theta),
\]  

(3-28)

or NBJ

\[
\hat{y}(k) = f_{NN}\left( u(k), u(k-1), ..., u(k-N_u), \hat{y}(k-1), \hat{y}(k-2), ..., \hat{y}(k-N_y),
\begin{array}{cccccc}
e(k-1), e(k-2), ..., e(k-N_e), n(k-1), n(k-2), ..., n(k-N_n), \theta
\end{array}\right) + n(k)
\]  

(3-29)

models. Despite the advantage of the parallel model that it gives better predictions, it is known that a large number of time lagged inputs is needed to accurately model the plant [102], leading to a large parameter space. This is treated more in detail in section C., where backpropagation in time is explained.

Another disadvantage that comes with the parallel model, is the initial state of the system. The system contains some memory and as a result transients occur whenever the model is brought
Neural Network systems

into a new state or whenever the model is used the first time. These transients become worse if $N_y$ is large. Therefore, a balance must be found between the ability to predict future observations and these transients.

Choosing a model

Models with external dynamics can be seen as one step ahead predictors. Models with internal dynamics are best used for simulation purposes, as the model doesn’t need the true plant outputs. The latter case has a higher potential for output errors on the long term. This is certainly the case for nonlinear systems, where the internal nonlinearities can drive the system into a chaotic state.

The criteria that determine the choice of the model, are the following.

- Which information is available? It seems obvious that all possible information must be used for the model, i.e. the true outputs, the noise contributions, and the past inputs. The result is an increasing number of model parameters that need to be optimized. If only the exogeneous input $u$ is know, then the only models possible are NFIR or NOE. If the plant outputs can be measured, a NARX model can be taken into consideration. In general it is not possible to know the process noise $n$, but it could be estimated for use with a NARMA or NARMAX model.

- Can significant time lags be estimated? The determination of the values of $N_y$, $N_u$, $N_e$ and $N_n$ is still an open question. It seems obvious that large values allow for better prediction of the future state of the NN. However, large time lags also result in large parameter vectors that need to be optimized.

- How many measurements are available for the optimization of the parameter vector? Models with a large number of parameters, don’t match with a small measurement set.

- Is the model used for control of simulation purposes? For simulation purposes, only NFIR, NOE and NBJ models fit the job, because they don’t need the current plant outputs. On the other hand these models are hard to optimize as the smallest initial error can have unpredictable results after a large number of iterations.

- Are transients an issue? Models with large time lags or with internal dynamics, can suffer from large transients when the system is brought into a new state.
Since it is the goal to have the least number of parameters, it makes sense to choose the simplest model available, and switch over to more complex models if this yields a significant improvement in performance. The first model to be considered is, therefore, not an NFIR model but its FIR counterpart, i.e. a linear model.

3.3 Neural Networks for control

3.3.1 Introduction

There is a good reason to use the classic linear control theory for controlling linear systems. Not only it is possible to control the plant to within precisely defined boundaries of overshoot, settling time and final error, also stability criteria can easily be defined and are fully understood.

This leaves two areas where nonlinear control can come into play. The first area is the case where unmodelled dynamics or model errors (e.g. due to a time-varying process) come into play. In this field it has been observed that Fuzzy Logic controllers provide excellent robustness.

The second area is the case where the system contains soft nonlinearities, e.g. soft saturation effects. These effects can be measured and modelled using e.g. Neural Networks. When using a NN for control, the experimenter can choose from different approaches

- The NN is used to linearize the dynamics of the controlled plant. The plant is translated to the closest possible linear model, which is then controlled using the classic PID controllers mentioned above.
- The NN directly controls the plant output. This means that the NN also includes the controller itself. The optimization of the NN parameters is then based on demands for overshoot, settling time and final error.
- A combination of the two previous configurations is where a linear controller is used for the plant. A NN gradually takes over control where nonlinearities in the model cannot be controlled with the PID controller. The NN is optimized based on the plant output error.
- The NN is used for a White Box modeling of the plant, i.e. model the inertial terms, the centrifugal and coriolis terms, the gravity terms, the friction terms, the static and dynamic behaviour of the plant, and do a robustness study for the unmodelled dynamics. All of these terms can be handled using proper measurements and multiple NN’s [152].
The description of all possible configurations where NN have been used for control, would go far beyond the scope of this book. Dozens of configurations can be found in the literature, sometimes quite dedicated for a single application. A few generally useable control schemes are given in the next section.

### 3.3.2 Some nonlinear control schemes

A number of nonlinear control schemes try to eliminate the nonlinearity in a plant using e.g. a NN or FL model [62] [109] [224]. In this thesis this will be called linearizing control. The resulting plant is then considered as an ordinary linear plant for which the regular and well known linear control theories apply.

The scheme is given in FIGURE 3-10 with $d(k)$ the reference trajectory of the plant, $e_r(k)$ the tracking error, $e_p(k)$ the plant error, $\hat{u}(k)$ and $\hat{y}(k)$ the estimated control input and plant output respectively, $\tilde{u}(k)$ the corrected control input and $u(k)$ the measured control input.

For linearizing control the nonlinear model must be trained on-line and full input-output measurement data of the actuator-plant combination must be available. The NN is then optimized such that the concatenation of the NN, the actuator and the nonlinear plant, acts like the linear reference plant. Only then real time control of the plant is possible. A drawback of this way of working is that while the plant itself is stable the NN inversion of the nonlinearities can be non minimum phase and specific techniques must be used to stabilize the NN model (such as bringing in delays).

![FIGURE 3-10 Linearizing control.](image_url)

It seems a bit awkward to include a linear controller in the closed loop, except for the sake of using classic stability theorems to determine the plant behaviour. The linear controller can be included in the reference plant to become the desired output response of the plant and the NN is then used for direct control of the plant [31] [76] [87] [88] [124] [128] [156] [159]. This control scheme is defined in this thesis as direct adaptive control and shown in FIGURE 3-11.
A drawback of both the linearizing and direct control schemes is that the NN must be optimized while the plant is perturbed. A bad convergence on the NN parameters can result in a damaged actuator or damage in the plant itself. To solve this problem partially, the initial state of the NN can be chosen as an identical behaviour with \( f_{NN}(d(k)) = d(k) \) such that \( \tilde{u}(0) \equiv \hat{u}(0) \). This asks for a prior modeling of the NN.

Another drawback of the direct control given in FIGURE 3-11 is that, in order to calculate the gradients for optimizing the NN parameters, the transfer functions of the actuator and the nonlinear plant should be known. This is not always the case. Therefore, the model predictive control configuration of FIGURE 3-12 can be chosen [65] [103] [126] [158] [202] [205] [207] [210] [239], which is also called indirect adaptive control [159].

The measurements on the actuator-plant combination are used to optimize a NN model. The tracking performance of this NN model is then used to adapt a NN controller which forces the plant to respond as a reference model. For the calculation of the gradients to optimize the NN controller, the transfer function of the NN model can be used.
Remark that both with direct neural control (FIGURE 3-11) and model predictive control (FIGURE 3-12), the control of the plant is actually done in an open loop configuration. For that reason, neither of the two optimization schemes are recommended for use with non-minimum phase systems.

Both in FIGURE 3-11 and FIGURE 3-12 the linear reference plant can be chosen as the identity matrix and a closed loop configuration can be chosen [87] [106] [108] [116] [169] [170] [175] [180] [215]. The $y(k)$ and $\hat{y}(k)$ outputs are then used as an input to the NN controller. The resulting control schemes are given in FIGURE 3-13 and FIGURE 3-14.

![FIGURE 3-13 Direct neural network control.](image)

![FIGURE 3-14 Model predictive neural network control.](image)

A correcting control configuration is given in FIGURE 3-15 [129] [184] [257]. This configuration is sometimes favoured whenever one doesn’t want to give up the well known basics on classic linear control. The initial state of the controller is $\tilde{u}(0) \equiv 0$, such that the plant is initially controlled by the PID controller, without correction. During optimization, the NN parameters are adapted until a feedforward control is realized.

The NN gradually takes over the PID controller in the regions where a residual error remains after the linear control. As a result, the linear controller takes care for the linear part of the plant and actuator, while the NN takes care for the nonlinear part. The advantage of this way of working is that the initial NN is easy to initialize: all parameters are set to zero. This could cause problems when optimizing the controller, as all parameter updates can become zero too.
A way to overcome this deficiency using Narendra’s e-modification, is described by Lewis [129].

The NN is typically trained on-line, but since it is the goal to minimize the tracking error $e_i(k)$, there is a smaller chance that the controller becomes unstable in the whole.

3.4 NN parameter optimization

3.4.1 Supervised vs. unsupervised learning

Within system identification, the model output is typically compared with a measured output. The comparison is based on a cost function and the model parameters are optimized for the cost function. Within NN research this approach is called supervised learning: someone is telling the neural network what to do and what not to do. The approach is justified e.g. for pattern recognition, function approximation or classification.

NN researchers sometimes also use so called self-organizing neural nets. Some networks are set up without input-output measurements while the behaviour of the network is observed. The behaviour is then only dependent on the complexity and type of transfer function that is used as an activation function for the neurons. Practical examples of such networks can be found in the "life" games and the creation of Mandelbrot figures. The direct practical use for system identification is doubtful.

In most cases of unsupervised learning, the nets are trained using input measurements only, and the model parameters are updated such that the same input patterns activate the same neurons in the network. A typical example of networks that can be used with unsupervised training, are Kohonen networks.

In the case that the network is trained with input-data only, it is expected that the data is centralized in certain regions of the network. The NN is then expected to map the separate
regions. This expectance also, is a sort of user imposed cost function: NN that do not sufficiently group the measurement data, are rejected.

One could argue for the use of the terminology of unsupervised learning. In this book, however, the approach is accepted where any designed model is measured one way or another to distinguish between "good" and "bad" models. This can be considered as a manual cost function which makes the distinction between supervised and unsupervised optimization sometimes blurry [18]. For that reason we consider ourselves always in the case of supervised learning.

There is, however, one application on unsupervised learning where the neurons are typically clustered such that the input data always activate one or some neurons, i.e. is observable. This is a sort of clustering which is also used in fuzzy logic to optimally cover features in the data. In this book, this clustering is considered as parameter initialization, and not as a real optimization with respect to measurement data.

3.4.2 Batch optimization vs. incremental optimization

Also called off-line and on-line optimization, respectively, the difference between batch and incremental optimization is the way how the measurement data are used to optimize the model parameters. In the first case, the errors of the whole learning set are used for the calculation of the parameter update vector. This is done repeatedly until convergence of the model parameters is reached on the whole data set. In the latter case the parameters are updated each time a measurement is taken.

3.4.3 Optimization of recurrent networks

A. Introduction

Consider the discrete time recurrent network of FIGURE 3-16 for which the parameters $\theta$ of the model

$$y(k) = f_{BB}(u(k), y(k-1), \theta)$$

must be optimized. Because of the recurrency, each step in the optimization process is also influenced by the previous outputs of the model. The network contains some memory which can be, from a theoretical point of view, infinitely large. In practise the memory is limited by the calculation precision, environmental noise, and acceptable error levels.
The recurrency poses a special problem when optimizing the network. For a good generalization, the network must not only be optimized with sufficient measurements in the domain of interest, but also with all possible histories.

In the sequel, two ways for the optimization of recurrent networks are described:

- Dynamic Backpropagation (DP) [159] [205] starts from the state-space representation of the model and calculates a sensitivity model for the optimization of the parameters. The same technique is used by Hayes [86] when he describes the Real-Time Recurrent Learning (RTRL) [248] algorithm. The model parameters are optimized with each new measurement sample and since the measurements are taken at distinct time steps, this results in a real-time optimization.

- The Backpropagation through time (BPTT) (Fausett [63], Haykin [86], Hertz [91]) algorithm unfolds the network a few time steps and is an extension of the backpropagation algorithm. The method given here basically differs from the definition of Backpropagation Through Time by Werbos [246] and Williams et al. [248], which has more correlation with dynamic backpropagation.

### B. Dynamic Backpropagation

Consider the more general state-space representation of the recurrent network in FIGURE 3-16

\[
x(k + 1) = g_{BB}(x(k), \tilde{u}(k), \Theta_g)
\]

\[
y(k) = f_{BB}(x(k), \tilde{u}(k), \Theta_f)
\]

with \( \tilde{u}(k) \in \mathbb{R}^{m+n} \) the enhanced input, \( y(k) = [y_1(k), y_2(k), ..., y_n(k)]^T \in \mathbb{R}^n \) the output, and \( x(k) \in \mathbb{R}^M \) the state vector of the system and where \( g_{BB} \) and \( f_{BB} = [f_{BB,1}, f_{BB,2}, ..., f_{BB,n}]^T \) are linear or nonlinear mappings. The state space description (3-30) can easily be replaced by (3-31a) (3-31b) by choosing \( \tilde{u}_k = [u^T, y^T]^T \) and selecting the proper linear model for \( g_{BB} \), viz. \( x(k + 1) = g_{BB}(x(k), \tilde{u}(k), \Theta_g) = Ax(k) + B\tilde{u}(k) \).
The training set is based on the input-output measurements \((\hat{u}(k), \hat{y}(k)), k = 1, 2, ..., N\). To fix the ideas, and without loss of generality, assume that an LS cost function is selected for the optimization of the BB parameters \(\theta_g\) and \(\theta_f\), such that

\[
C_{LS}(\theta_g, \theta_f) = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} (f_{BB,i}(x(k), [\hat{u}^T(k), y^T(k)]^T, \theta_f) - \hat{y}_i(k))^2.
\]

(3-32)

is minimized. With the definition of the error

\[
e(k) = [e_1(k), e_2(k), ..., e_n(k)]
\]

(3-33)

where

\[
e_i(k) = f_{BB,i}(x(k), [\hat{u}^T(k), y^T(k)]^T, \theta_f) - \hat{y}_i(k) = f_{BB,i}(x(k), \bar{u}(k), \theta_f) - \hat{y}_i(k),
\]

(3-34)

the cost can be written as

\[
C_{LS}(\theta_g, \theta_f) = \frac{1}{N} \sum_{k=1}^{N} e^T(k) e(k).
\]

(3-35)

Just like the backpropagation optimization method, dynamic backpropagation is based on the Jacobian matrix defined in (2-57) on page 39. In this case the calculation of the Jacobian is dependent on the parameters of both mappings \(g\) and \(f\), such that the partial derivatives \(\partial e_i(k)/\partial \theta_{g,j}\) and \(\partial e_i(k)/\partial \theta_{f,j}\), \(j = 1, 2, ..., n\), must be calculated. The gradient can be calculated by a sensitivity method, using the equations

\[
\frac{\partial x(k+1)}{\partial \theta_g} = \frac{\partial g_{BB}}{\partial x(k)} \frac{\partial x(k)}{\partial \theta_g} + \frac{\partial g_{BB}}{\partial \theta_g},
\]

(3-36a)

\[
\frac{\partial y(k)}{\partial \theta_g} = \frac{\partial f_{BB}}{\partial x(k)} \frac{\partial x(k)}{\partial \theta_g}, \text{ and}
\]

(3-36b)

\[
\frac{\partial y(k)}{\partial \theta_f} = \frac{\partial f_{BB}}{\partial \theta_f}.
\]

(3-36c)

The sensitivity model is a dynamical system that generates the outputs \(\partial y(k)/\partial \theta_g\) and \(\partial y(k)/\partial \theta_f\) at each iteration step in the optimization. The sensitivity model is then used for the calculation of the Jacobian like
such that with each new measurement, the new Jacobian can be calculated and the model parameters can be updated. For reasons of stability of the algorithm, a small adaptation factor is recommended \cite{86}.

\section*{C. Backpropagation through time}

Within the NN domain the optimization of recurrent networks using the backpropagation scheme, is called \textit{backpropagation in time}, or also \textit{recurrent backpropagation}. The general idea is that only a limited number of time steps are taken into account for optimizing, while older history is neglected. Choosing the number of time steps is a difficult task, but should be based upon the memory capacity of the network (i.e. its ability to 'forget').

The model has a parameter vector $\theta$ of size $P \times 1$. After expanding FIGURE 3-16 one time step, we get the situation of FIGURE 3-17

\begin{equation}
J(k) = \begin{bmatrix}
\frac{\partial f_{BB}}{\partial x(k)} \frac{\partial x(k)}{\partial \theta_g} \\
\frac{\partial f_{BB}}{\partial \theta_f}
\end{bmatrix}.
\end{equation}

(3-37)

The expansion of the recurrent system for one time step, can be considered as a system with $2P$ parameters. If the expansion is performed $M$ times, $(M + 1)P$ parameters result.

The idea behind recurrent backpropagation is that the parameter update vector $\Delta \theta$ is calculated for each of the sections in FIGURE 3-17 The recurrent model parameters are then updated with the total, or the average, of the individual parameter update vectors. Backpropagation in time is in fact a dedicated optimization scheme. The addition, or the averaging, of the individual parameters vectors is consistent with the use of gradient methods for the optimization step.

Although exact gradients could be calculated if the unfolding in time would be done over all time-steps, this leads to growing memory requirements that are proportional to the length of the training sequence.
Neural Network systems
Abstract: this chapter gives a brief overview of common FL terminology and modeling techniques. In Chapter 2 it was already shown that the FL input-output relationship should be considered as just another nonlinear model. Yet, the FL community has profiled FL modeling as a complete new research domain. Many problems needed to be tackled that were (partly) solved in other domains, e.g. stability issues. Sometimes, techniques were re-invented, e.g. optimization of the model parameters. Some problems are still open, such as a uniform scheme for modeling.

This doesn’t mean that FL research is a domain on the loose. The approach of tackling the initialization of the model parameters using a linguistic approach or by clustering, is new in the modeling field. Fuzzy Logic offers new and valuable techniques in the field of automatic control. Its ease of use, and the robustness against model errors makes it a good replacement for classic controllers.

The much criticized problem that no modeling scheme is available for FL systems, is treated more in detail in the last section where a general modeling approach is suggested.
4.1 Introduction

Fuzzy Logic (FL) is a mathematical description of processes based on Fuzzy Set theory, that allows for degrees of truth and falseness. Since Zadeh [259] introduced Fuzzy Sets, many discussions have taken place whether FL deserves a place in the modeling and control theories.

Fuzzy Logic has to deal with a lot of scepticism, and this is not only because people distrust any new technology that is perceived as revolutionary [29]. It has become popular as an alternative control theory where classic control fails to give satisfying results. This can be caused by the complexity of the controlled system but, sometimes, this is caused by a lack of expertise of the user who is then attracted by the seemingly simple linguistic formulation of fuzzy control rules. It is this aura of simplicity that has made senior control and identification researchers to revoke Fuzzy Logic in the whole [2] [215]. Yet, FL is a promising theory that gained a lot of interest in practical applications [1].

Three properties speak in favour of FL control. The first being its robustness against parameter uncertainty [76] [167] [168], the second the fact that the FL controller output is normalized. While linear PID controllers assume that the controller output can be much larger than the maximal actuator output, FL systems by nature restrict the controller value to a normalized interval. The third property in favour of FLC is the ‘linguistic’ interpretation of the control scheme [64] [124].

In this chapter FL is analysed within the scope of identification theory. It is shown that FL fits well in the general scheme of chapter 2 as another model that can be seen as a superset for linear control theory. It is not the goal to give an exhaustive description of all Fuzzy Models that exist in the literature. More detailed descriptions of FS can be found in the many references given, e.g. Jager [107], Nowé [165] and Passino et al. [171] and Sjöberg et al. [194].

4.2 The Fuzzy Logic basic model

4.2.1 Introduction

A basic structure of a fuzzy rule-based system is shown in FIGURE 4-1. Consider the measured input values $u_k = [u_{k,1}, u_{k,2}, \ldots, u_{k,m}] \in \mathbb{R}^{m \times N}$ and output values $y_k = [y_{k,1}, y_{k,2}, \ldots, y_{k,n}] \in \mathbb{R}^{n \times N}$ with $N$ the number of measurements, $m$ the number of inputs and $n$ the number of outputs. In Fuzzy Set theory these measurements are called the crisp values. The whole sets of input and output measurements are called the universes of...
discourse and are defined in the whole space of real numbers. For convenience effective universes of discourse are defined which restrict themselves to the practical measurement values.

In linear theory the universes of discourse are defined as the input and output domains, and the latter definitions are also used further in this book. Further, it is possible to say that both effective universes of discourse combine into the domain of interest $\mathcal{Y}$, defined in Chapter 2.

The inputs are fuzzified based on a number of Fuzzy Sets. The resulting Fuzzy inputs $\tilde{u}$ are the inputs for an Inference Engine which is basically a collection of rules $R$ in the form "If a set of premises Then a set of consequences". These rules are stored in the Rule Base (sometimes also called the Knowledge Base). The consequences of the inference engine are the fuzzified output values $\tilde{y}$ which are then defuzzified to become the crisp output values $y$.

The implementation of the Rule Base and the way how the fuzzification and defuzzification take place are determined by a number of parameters which are stored in the Data Base. In the sequel of this chapter, the Data Base is denoted as $[\phi^T, \theta^T]^T$. It contains the parameter vector $\theta$ that must be optimized, and a vector of fixed FLS parameters $\phi$ that can be initialized in the initialization part of the general scheme but are not further optimized.

**Definition 4.1** The width of a single input or output in the domain of interest is defined as the measure $|\beta_i - \alpha_i|$ where $\beta_i$ is the absolute maximum value and $\alpha_i$ is the absolute minimum value of the $i$-th input or output.

**Assumption 4.1** In the sequel of this chapter it is assumed that all inputs and outputs are normalized to the interval $[-1, 1]$.

### 4.2.2 Linguistic values and rules

Fuzzy Logic has become well known because of its linguistic interpretation of the Rule Base. One way to denote the linguistic values is as follows.

---

**FIGURE 4-1** Fuzzy rule-based system.
Definition 4.2 Each single input $u_{k,i}$ and output $y_{k,j}$ is defined by its linguistic representation $\tilde{u}_i$ and $\tilde{y}_j$ respectively. This linguistic representation is the formal definition of the parameter in the Fuzzy domain.

Examples of these linguistic representations are e.g. $\tilde{u}_i =$"tracking error" and $\tilde{y}_j =$"output voltage". In other words, the linguistic representations are in general chosen as the labels of each input and output.

Definition 4.3 For each crisp input or output value, a set of $r_i$ linguistic variables $\tilde{A}_i = \{\tilde{A}_i^j; j = 1, 2, ..., r_i\}$ is defined that describe the crisp value in the Fuzzy domain.

Examples of linguistic variables are $\tilde{A}_1^1 =$"very low", $\tilde{A}_3^3 =$"zero" or $\tilde{A}_4^4 =$"high". It is possible to write "$\tilde{u}_i \text{ is } \tilde{A}_1^1$" which denotes the linguistic condition that the "tracking error is very low".

Definition 4.4 A linguistic rule is an If ... Then structure based on linguistic variables.

An example of a linguistic rule $R_i$ is

$$R_i: \text{If } (\tilde{u}_1 \text{ is } \tilde{A}_1^1) \text{ and } (\tilde{u}_2 \text{ is } \tilde{A}_2^3) \text{ Then } (\tilde{y}_1 \text{ is } \tilde{B}_1^3)$$  

(4-1)

which would represent the Fuzzy Logic representation of the spoken phrase "If the tracking error is very low and the change in error is zero, then the correction is zero".

Definition 4.5 The Rule Base is the collection of all linguistic rules of a Fuzzy system or a Fuzzy controller.

Remark: Assume a system with $m$ inputs where each input $i$ is described with $r_i$ linguistic variables. A full Rule Base would need $\prod_{i=1}^{m} r_i$ linguistic rules. Sometimes less rules are used if it is known that certain input combinations are never reached in the domain of interest. On the other hand, some Fuzzy systems make use of far more rules than necessary for a full Rule Base, i.e. the Rule Base contains redundant rules.

Definition 4.6 A Fuzzy Rule Base is redundant or inconsistent if there are at least two rules with the same premise (antecedent), but different rule-consequent.

4.2.3 Fuzzy Sets

The goal of using Fuzzy Sets is the partitioning of the input domain of a Fuzzy System. In the Single-Input case the input is divided into sections. In the Multiple Input case the input space is
The Fuzzy Logic basic model

divided into hypercubes. The edges between the cubes are not strictly separated, such that a given input can belong to different Fuzzy Sets.

**Definition 4.7** The transformation of a crisp input to Fuzzy Sets is called the *fuzzification*.

Consequently, it is possible to define the inverse operation of the fuzzification as follows.

**Definition 4.8** The transformation of a set of Fuzzy Sets into a single crisp value, is called the *defuzzification*.

One of the basic elements of a FS is the *membership function*. It is a measure how a crisp value can be transformed into a Fuzzy Set. The actual fuzzification is done by applying the membership function on the crisp input.

**Definition 4.9** The *membership function* $\mu_{A_i}(u_i)$ is the measure how much the crisp input $u_i$ belongs to the fuzzy set $A_i$.

**Definition 4.10** A *Fuzzy Set* is the collection of inputs with the related membership degrees and is defined as

$$A_i = \{ (u_i, \mu_{A_i}(u_i)); u_i \in \psi \} .$$ (4-2)

Typical membership degrees are given in FIGURE 4-2. The input $u_i = 0.12$ belongs with a membership degree $\mu_{A_i}(u_i) = 0.9$ to the Fuzzy Set $A_i^4$ and a membership degree $\mu_{A_i}(u_i) = 0.15$ to the Fuzzy Set $A_i^5$. The linguistic representations of each set is given above each set. There is no relationship between the shape of the FS and the linguistic interpretation.

Note that the first and last sets ($A_i^1$ and $A_i^6$) are *open sets*, used to guarantee that the whole input domain is covered with Fuzzy Sets. One could argue that this is not necessary once the
domain of interest is defined and the inputs are correctly normalized. The use of open sets is therefore a means to guarantee the operation of e.g. a Fuzzy controller in the case of model errors.

To clarify this reasoning, consider the case where the width of an input is chosen wrongly due to model errors. It could be very well possible that a given input leaves the domain of interest at a certain moment. If the Fuzzy Sets were restricted to the domain of interest, this would leave the Fuzzy controller in an undefined state. This is not the case when using open sets at both ends.

**Definition 4.11** A Fuzzy Set is *convex* if and only if the following inequality holds:

\[
\mu_A(\gamma u_1 + (1-\gamma)u_2) \geq \min(\mu_A(u_1), \mu_A(u_2))
\]  

(4-3)

where \(0 \leq \gamma \leq 1\), \(u_1 \in \mathbb{R}\) and \(u_2 \in \mathbb{R}\).

**Assumption 4.2** In this thesis it is assumed that all Fuzzy Sets that are used as input FS, are convex types.

FIGURE 4-3 shows how a two-input space is divided in Fuzzy regions, using triangular Fuzzy Sets. A short list of possible definitions for Fuzzy sets is given in TABLE 4-1. The \(\gamma_1, \gamma_2, \ldots\) are FS parameters for which yields that \(\gamma_1 \leq \gamma_2 \leq \gamma_3 \leq \gamma_4\) [89].

![Division of a two dimensional input space in Fuzzy regions.](image)

**Remark:** The linguistic rule (4-1) on page 78 is equivalent to the notation

\[
R_i: \text{If } (u_1 = A_1^1) \text{and } (u_2 = A_2^3) \text{ Then } (y_1 = B_1^3).
\]  

(4-4)

Although this notation is not fully correct from the mathematical point of view where scalar inputs cannot be compared with sets of vectors, it is largely used in the literature because of
notational reasons. Moreover there is little or no danger that the notation (4-4) is interpreted wrongly. In this book the notation (4-4) is used to denote the rules in the Rule Base, and the notation (4-1) is used to stress explicitly that linguistic rules are used, e.g. when creating a Rule Base from inquiry forms.

In the example of FIGURE 4-2 it is possible to state that the condition \( u_i = A_i^4 \) has a membership degree \( \mu_{A_i^4}(u_i) = 0.9 \), while e.g. the condition \( u_i = A_i^1 \) has a membership degree \( \mu_{A_i^1}(u_i) = 0 \).

Remark: From the Fuzzy Logic inference point of view, the premise doesn’t treat the first input applied on its first fuzzy set \( u_i = A_i^1 \) different from the second input, applied on its third fuzzy set \( u_2 = A_2^2 \). A linguistic rule is therefore written as

\[
R_i : \text{If } (u_i^1 = A_i^1) \text{ and } (u_i^2 = A_i^2) \text{ and ... and } (u_i^p = A_i^p) \text{ Then } (y_i^r = B_i^r) \tag{4-5}
\]

in the sequel of this chapter. The \( u_j^r \) in this equation can be any of the inputs \( u_i \), while the \( A_j^r \) can be any of the fuzzy sets \( A_i^r \) that correspond with the particular input \( u_j \). The membership degree of the premise \( u_i^r = A_i^r \) in equation (4-5), is then written as \( \mu_{A_i^r}(u_i) \).

**Definition 4.12** The \( r_i \) Fuzzy Sets \( A_i(u_i) \) with \( j = 1, 2, ..., r_i \) that correspond to a crisp value \( u_i \) are considered to be normal if the following equation holds:

\[
\sum_{j=1}^{r_i} \mu_{A_i}(u_i) = 1 \tag{4-6}
\]

for all inputs \( u_i \in \mathbb{R} \).

Some references make use of all overlapping sets. For a SISO system this would mean that an input would be mapped on more than two Fuzzy Sets. An input that would normally be marked as "very large" with a membership degree of 0.8 and "large" with a membership degree of e.g. 0.2, could also be mapped on "zero" and "small", albeit with smaller membership degrees. It is still possible to guarantee that the input sets are normal, but changing the consequences of Fuzzy rules will not have a trivial change in the FLS or FLC input-output mapping [107].

Note that the input is not limited to the domain of interest \( \psi \), but can be any real value. Some references limit this demand to the domain of interest. In this book the more restricting definition with \( c \in \mathbb{R} \) is chosen to force the use of open sets.
4.2.4 Fuzzy Hedges

Although in this book FL is used for system identification and control, it was basically invented as a translation between linguistic commands and computational devices. These linguistic commands include the use of words like "very" or "more or less", which are called linguistic hedges. It is out of the scope of this book to include a list of all known hedges, while it must be well understood that a large variety of hedges exist, such as powered hedges [262], shifted hedges and scaled hedges [107].

The basic idea behind hedges is contrast intensification. For powered hedged, defined by Zadeh [259], the membership degree is then defined by
The Fuzzy Logic basic model

\[ \mu_{\alpha}(u_i) = \mu_{\alpha}^A(u_i). \]  \hspace{1cm} (4-7)

The operation of *concentration* is then defined as \( \alpha = 2 \), while the membership degree for *dilatation* is defined for \( \alpha = 0.5 \).

This book adopts the definition of shifted hedges [107], i.e.

\[ \mu_{\text{very}(A)}(u_i) = \mu_A(u_i - s) \]  \hspace{1cm} (4-8)

where \( s \) denotes some shift away from the origin. There is a reason for using shifted hedges, since it has the property that the basic form of the Fuzzy Set remains the same. The property that the shape of the FS is preserved is needed to guarantee that the input sets remain normal.

### 4.2.5 Fuzzy operators

While earlier it was said that a large variety of Fuzzy hedges exist in the literature, this is even more the case for Fuzzy logical operators, such as the **and** and **or** operators. In the past years of research on Fuzzy Logic, seemingly everyone wanted to define his or her own way for performing logical operations on Fuzzy Sets. To date, two methods are accepted to be the most operable: the min-max and the product-sum operator combinations. The definitions for a few cases are given in TABLE 4-2.

An operator that must not be left away in this discussion is the **not** operator, which is defined as

\[ \mu_{\text{not}(A)} = 1 - \mu_A. \]  \hspace{1cm} (4-9)

Here also, there is no uniform definition for the **not** operator in the literature. The remaining question is: which of the many existing operator definitions should be used?

In practice, Fuzzy systems are proven to be universal approximators [65] [102] [107] [119] [133] [194] [209] [243] [247]. Based on the general identification scheme, it is possible to state that the use of different operator definitions only leads to another model with different parameters, but with the same universal approximation properties. The experimenter should, therefore, just pick a model which suits his needs for computational speed and modeling precision.
4.2.6 t-norms and s-norms

In the early days of Fuzzy Logic, the and, or and not operators discussed in the previous section, were the subject of many mathematical publications. FL was more considered as a translation from spoken language and human reasoning to exact math.

It is from this mathematical point of view, that t-norms were defined. The counterpart of a t-norm is the t-conorm, also known as s-norm. Despite of the word norm in the definition, this has nothing to do with the generally accepted definition of norms.

<table>
<thead>
<tr>
<th>and operator</th>
<th>or operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-max</td>
<td></td>
</tr>
<tr>
<td>$\mu_{A \land B} = \min(\mu_A, \mu_B)$</td>
<td>$\mu_{A \lor B} = \max(\mu_A, \mu_B)$</td>
</tr>
<tr>
<td>Algebraic product -</td>
<td></td>
</tr>
<tr>
<td>algebraic sum</td>
<td>$\mu_{A \land B} = \mu_A \mu_B$</td>
</tr>
<tr>
<td>product-sum</td>
<td>$\mu_{A \land B} = \mu_A \mu_B$</td>
</tr>
<tr>
<td>Bounded difference-</td>
<td></td>
</tr>
<tr>
<td>Bounded Sum</td>
<td>$\mu_{A \land B} = \max(0, \mu_A + \mu_B - 1)$</td>
</tr>
<tr>
<td>Probabilistic Sum</td>
<td></td>
</tr>
<tr>
<td>Disjoint Sum</td>
<td>$\mu_{A \land B} = \max(\mu_A, 1 - \mu_B), \min(1 - \mu_A, \mu_B)$</td>
</tr>
<tr>
<td>Einstein product -</td>
<td></td>
</tr>
<tr>
<td>Einstein sum</td>
<td>$\mu_{A \land B} = \frac{\mu_A \mu_B}{2 - (\mu_A + \mu_B + \mu_A \mu_B)}$</td>
</tr>
<tr>
<td>Hamacher product -</td>
<td></td>
</tr>
<tr>
<td>Hamacher sum</td>
<td>$\mu_{A \land B} = \frac{\mu_A \mu_B}{\mu_A + \mu_B - \mu_A \mu_B}$</td>
</tr>
</tbody>
</table>

**TABLE 4-2.** A set of and and or operator definitions.
Definition 4.13 t-norms are two-valued functions from \([0, 1] \times [0, 1] \rightarrow [0, 1]\) which satisfy the conditions:

1. \(t(0, 0) = 0\)
2. \(t(\mu_A(u), 1) = t(1, \mu_A(u)) = \mu_A(u)\)
3. \(t(\mu_A(u), \mu_B(u)) \leq t(\mu_C(u), \mu_D(u))\) if \(\mu_A(u) \leq \mu_C(u)\) and \(\mu_B(u) \leq \mu_D(u)\) (monotonicity)
4. \(t(\mu_A(u), \mu_B(u)) = t(\mu_B(u), \mu_A(u))\) (commutativity)
5. \(t(\mu_A(u), t(\mu_B(u), \mu_C(u)))) = t(t(\mu_A(u), \mu_B(u)), \mu_C(u))\) (associativity)

The and operators in TABLE 4-2. can be considered as t-norms, such that \(\mu_{A \text{ and } B} = t(\mu_A, \mu_B)\). The or operators in the table can then be considered as s-norms or t-conorms, such that \(\mu_{A \text{ or } B} = s(\mu_A, \mu_B)\) where the s-norm \(s(\mu_A, \mu_B)\) is defined as follows.

Definition 4.14 s-norms or t-conorms are two-valued functions from \([0, 1] \times [0, 1] \rightarrow [0, 1]\) which satisfy the conditions:

1. \(s(1, 1) = 1\)
2. \(s(\mu_A(u), 0) = s(0, \mu_A(u)) = \mu_A(u)\)
3. \(s(\mu_A(u), \mu_B(u)) \leq s(\mu_C(u), \mu_D(u))\) if \(\mu_A(u) \leq \mu_C(u)\) and \(\mu_B(u) \leq \mu_D(u)\) (monotonicity)
4. \(s(\mu_A(u), \mu_B(u)) = s(\mu_B(u), \mu_A(u))\) (commutativity)
5. \(s(\mu_A(u), s(\mu_B(u), \mu_C(u)))) = s(s(\mu_A(u), \mu_B(u)), \mu_C(u))\) (associativity)

Because of the associativity property the notation \(t(\mu_A(u), t(\mu_B(u), \mu_C(u))))\) can also be denoted as \(t(\mu_A(u), \mu_B(u), \mu_C(u))\). The same yields for the notation of the s-norm.

In FL t-norms and s-norms are used in a sense of logical duality [262]. For some of the definitions in TABLE 4-2. it is possible to state that

\[
t(\mu_A, \mu_B) = 1 - s(1 - \mu_A, 1 - \mu_B).
\]

(4-10)

In some publications a sort of negative logic reasoning is implemented, using inverse fuzzy sets \(\overline{A_i}\) such that \(\overline{A_i} = \text{not}(A_i)\) and using s-norms instead of t-norms and vice-versa. In this thesis the positive logic reasoning is adopted.

Definition 4.15 \(A_1 \otimes A_2 \otimes ... \otimes A_p\) denotes the combination of the fuzzy sets with s-norms. The combination of fuzzy sets with t-norms, is denoted as \(A_1 \oplus A_2 \oplus ... \oplus A_p\).
4.2.7 Fuzzy Inference

**Definition 4.16** The Inference Engine of the Fuzzy system links the premises in the If... Then structure to the consequences.

This thesis only considers inference schemes for the positive logic case. In that case the inference mechanism is usually some kind of s-norm implication between the premises and the consequences. This is done for each rule and the premise for each rule becomes

\[
\mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p(u_1^r, u_2^r, \ldots, u_p^r) = s(\mu_{A_1}(u_1^r), \mu_{A_2}(u_2^r), \ldots, \mu_{A_p}(u_p^r)).
\]  

(4-11)

Next, the premise is combined with the consequence. This also, can be done using an s-norm

\[
\mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p \otimes B_1(u_1^r, u_2^r, \ldots, u_p^r, y_1^r) = s(\mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p(u_1^r, u_2^r, \ldots, u_p^r), \mu_{B_1}(y_1^r))
\]  

(4-12)

The rules are then combined into one single membership function using a t-norm, such that

\[
\mu_{B_1}(y_1) = t(\mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p \otimes B_1(u_1^r, u_2^r, \ldots, u_p^r, y_1^r)),
\]

\[
\mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p \otimes B_1(u_1^2, u_2^2, \ldots, u_p^2, y_1^2), \ldots, \mu_{A_1} \otimes A_2 \otimes \ldots \otimes A_p \otimes B_1(u_1^r, u_2^r, \ldots, u_p^r, y_1^r))
\]  

(4-13)

with \( r \) the total number of rules \( R_j \) in the Rule Base. Note that until now only the MISO case is considered, resulting in a specific fuzzy set with membership function \( \mu_{B_1}(y_1) \). In the general MIMO case the described procedure is repeated for each output, resulting in the FS

\[
B_j = \{ (y_j, \mu_{B_1}(y_j)); y_j \in \psi \}
\]  

(4-14)

with \( j = 1, 2, \ldots, n \).

A special type of Fuzzy inference implements a *truth factor* for each rule. Each rule in the Rule Base is given a weight factor \( w_i \in [0, 1] \) that indicates how much the rule \( i \) should be effective. While this truth factor yields yet another set of parameters to be added to the parameter vector, some optimization schemes only tune (optimize) these weights, while keeping the Fuzzy Sets and rules constant. The Fuzzy Sets (4-14) then become

\[
B_j = \{ (y_j, w_i \mu_{B_1}(y_j)); y_j \in \psi \}.
\]  

(4-15)

Two inference schemes have become particularly popular in the past decades of FL research: the min-max and the product-sum inference. They can both be used with a truth factor. The min-max inference scheme uses the minimum operator for the s-norm and the maximum operator for the t-norm. For the product-sum inference these are the product and sum operators, respectively.
4.2.8 Defuzzification

The defuzzification translates the output fuzzy set (4-14) to a crisp output \( y_j \). Here also, a large number of defuzzification methods exist in the literature. Some of the defuzzification methods only operate on the overall membership functions (4-14). Other methods also include the final t-norm step, and calculate the crisp output directly from the individual rule outcomes (4-12). A list of possible defuzzification methods is shown in TABLE 4-3.

Although a large number of defuzzification methods are available, it can be concluded that some methods can seem awkward, if not illogical. A more in depth study on defuzzification methods was done by Hellendoorn and Thomas [88]. They ranked different methods, based on continuity, disambiguity, plausibility, computational complexity and weight counting properties.

\[ y_j = \int_{y_j \in \psi} \frac{y_j \cdot \mu_{B_j}(y_j) dy_j}{\int_{y_j \in \psi} \mu_{B_j}(y_j) dy_j} \]

\[ y_j = \int_{y_j \geq c} \frac{y_j \cdot \mu_{B_j}(y_j) dy_j}{\int_{y_j \geq c} \mu_{B_j}(y_j) dy_j} \text{ with } c \text{ a threshold value} \]

\[ y_j \in \arg \sup_{y_j} \{ \mu_{B_j}(y_j) \} \]

\[ y_j = \sum \arg \sup_{y_j} \mu_{B_j}(y_j) / N \text{ with } \mu_{B_j}(y_j) \text{ all local maxima of } \mu_{B_j}(y_j) \text{ and } N \text{ the total number of maxima.} \]

\[ y_j = \sum \arg \sup_{y_j} \mu_{B_j}(y_j) / N \text{ with } \mu_{B_j}(y_j) \text{ the leftmost, middle, or rightmost occurrence of local maxima in } \mu_{B_j}(y_j) \text{ and } N \text{ the number of local maxima.} \]
Based on their research, they concluded that the Centre-of-Sums defuzzification method provided superior performance on all criteria. The same conclusion was drawn by Matiá et al. [150].

4.2.9 Mamdani fuzzy logic systems

There isn’t just one basic FL model. From the brief discussion above, it can be concluded that a near endless combinations of Fuzzy sets, inference schemes, and defuzzification methods can be chosen that all carry the name Fuzzy Logic System. One of the much used models that made it out of this pool of combinations, is the Mamdani FLS.

The Fuzzy rules can have the form

\[
R_i; \text{If } (u_1^i = A_1^i) \text{ and } (u_2^i = A_2^i) \text{ and } ... \text{ and } (u_p^i = A_p^i) \text{ Then } (y_1^i = B_1^i) \text{ and } (y_2^i = B_2^i) \text{ and } ... \text{ and } (y_q^i = B_q^i) \tag{4-16}
\]

where the inputs \( u_1^i \) and outputs \( y_1^i \) follow the notational conventions of section 4.2.3. Note therefore, that the input \( u_1^i \) can denote any of the plant inputs: \( u_1^i \in \{ u_1, u_2, ..., u_m \} \). For the outputs this means that \( y_1^i \in \{ y_1, y_2, ..., y_n \} \).

The \textbf{and} operators in the consequence of the rule don’t have the same meaning as the operators in the premises. In practise they are just used to reduce the number of rules in the case of Multi Output systems. The rule (4-16) can actually be split in \( q \) rules of the form

\[
R_{i,j}; \text{If } (u_1^i = A_1^i) \text{ and } (u_2^i = A_2^i) \text{ and } ... \text{ and } (u_p^i = A_p^i) \text{ Then } (y_j^i = B_j^i) \tag{4-17}
\]

with \( j = 1, 2, ..., q \). It is therefore sufficient to consider the MISO problem (the same conclusion was drawn in [133]).

The input sets are mostly chosen as triangular or trapezoid, while the inference is mostly done with the min-max or product-sum operator combinations. Most experimenters choose the Centre of Gravity method for the defuzzification.

4.2.10 Takagi-Sugeno fuzzy logic systems

Takagi-Sugeno Fuzzy models can be considered as local-linear models. The use of a FLS then results in a smooth transition from one local linear operating point to the other. This becomes clear when analysing the model structure of a TS Fuzzy system:

\[
R_i; \text{If } (u_1^i = A_1^i) \text{ and } (u_2^i = A_2^i) \text{ and } ... \text{ and } (u_p^i = A_p^i) \text{ Then } (y^i = a_0^i + a_1^i u_1^i + a_2^i u_2^i + ... + a_p^i u_p^i). \tag{4-18}
\]
Only one output is considered here although MIMO configurations are possible, just as it was the case for the Mamdani model. The output is typically a linear function of the different inputs but can be any linear or nonlinear function \( f(u^1_1, u^1_2, ..., u^1_m) \). In this case the same variables were used as those used in the premise of the Fuzzy rule. Other configurations are possible, e.g. using variables in the consequence that are different from those in the premise of the rule.

An example of a Takagi-Sugeno system is shown in FIGURE 4-4. The system has three rules

\[
R_1: \text{If } (u = A^1_1) \text{Then } (y = a^1_0 + a^1_1 u) \quad (4-19a)
\]

\[
R_2: \text{If } (u = A^1_2) \text{Then } (y = a^2_0 + a^2_1 u) \quad (4-19b)
\]

\[
R_3: \text{If } (u = A^1_3) \text{Then } (y = a^3_0 + a^3_1 u) \quad (4-19c)
\]

The system uses triangular sets and a product-sum inference scheme. The premises for the three rules are simply the membership functions \( \mu_{A^1_1}(u) \), \( \mu_{A^1_2}(u) \) and \( \mu_{A^1_3}(u) \) respectively. The consequences can be calculated as

\[
R_1: \mu_{A^1_1} \otimes B^1(u, y) = s(\mu_{A^1_1}(u), a^0_0 + a^1_1 u) = \mu_{A^1_1}(u)(a^0_0 + a^1_1 u) \quad (4-20a)
\]

\[
R_2: \mu_{A^1_2} \otimes B^2(u, y) = s(\mu_{A^1_2}(u), a^2_0 + a^2_1 u) = \mu_{A^1_2}(u)(a^2_0 + a^2_1 u) \quad (4-20b)
\]

\[
R_3: \mu_{A^1_3} \otimes B^3(u, y) = s(\mu_{A^1_3}(u), a^3_0 + a^3_1 u) = \mu_{A^1_3}(u)(a^3_0 + a^3_1 u), \quad (4-20c)
\]

and finally the output of the FLS is calculated as

\[
f_{u_1}(u_1, u_2, ..., u_m) = \sum_{i=1}^{m} \mu_{A^1_i}(u) \cdot (a^0_i + a^1_i u)
\]
Note that the output didn’t need some specific defuzzification. The Takagi-Sugeno directly defines the consequences in a crisp way.

### 4.2.11 Singleton Fuzzy Systems

The singleton Fuzzy system makes use of rules of the form

\[
y = \sum_{i=1}^{3} \mu_{A_i}(u)(a_{0i} + a_{1i}u).
\]  

(4-21)

It is clear that the singleton Fuzzy Model can be considered as a special case of the Takagi-Sugeno model. A more popular approach considers the singleton Fuzzy system as a special case of the Mamdani model. The fuzzy output set \( B^i_1 \) in equation (4-16) on page 88 is then chosen as a singleton fuzzy set (see TABLE 4-1.).

### 4.2.12 Neurofuzzy systems

There are researchers who refuse to use neural networks because of their shortcoming of being "black boxes" [22]. In practise, no-one is going to dispute the use of a NN when a complicated task needs to be solved, as long as the network performs well. The main problem is that one can’t tell what is actually going wrong whenever a NN starts acting up [133].

While understanding the behaviour of NN is not that obvious, the optimization of the model parameters is well studied and well understood. Fuzzy systems, on the other hand, give a relatively easy overview of the rules that make up a controller. As an added value, this makes the initialization of fuzzy controllers and systems easier. Yet, no generally accepted and straightforward methodology exist for the choice and the optimization of the parameters.

The marriage between Fuzzy and Neural Networks, called neurofuzzy, promises to give nothing but advantages. Based on an analysis of the NN, Benitez [22] already proved that NN can be explained from a FL point of view, but due to the diversity in neural models this does not apply for all networks. The equivalence relationship between Fuzzy Logic and Neural Networks has been studied by more researchers (Chen et al. [37], Hornik [97], Jang [104]) but to prove the relationship, often an unbounded number of fuzzy rules is needed.

Recently, a number of neurofuzzy systems has been introduced, based on Neural Networks, where the neurons are configured to serve as Fuzzy Sets. Each layer of the NN operates as part of a Fuzzy model: the fuzzification, inference scheme, and defuzzification each use a layer in
the NN. The transfer functions in the neurons act as t-norms and s-norms. Neurofuzzy systems include the singleton FLS, Mamdani type and Takagi-Sugeno type Fuzzy Logic systems.

It lays beyond the scope of this thesis to give a detailed description of neurofuzzy systems. More detailed descriptions can be found in Horikawa et al. [96], Isermann et al. [102], the ANFIS model of Jang [104], Lin et al. [133] and Werbos [247].

4.3 Fuzzy Logic models for nonlinear identification

4.3.1 Introduction

Hellendoorn [89] states that it is generally agreed that the single most difficult step in identification is that of model structure selection. This is certainly true, taking into account the vast number of models available, and the sometimes large number of sub models in each of the fields. Both Neural Networks and Fuzzy Logic provide a proliferation of models.

Consider the case where a set of measurements is given that must be modelled using a FL model. Based on the scheme from Chapter 2, the first two basic decisions are then taken, and the optimization of the FLS parameters would involve three additional steps:

- Select the model parameters that must be optimized.
- Choose a cost function.
- Use one of the optimization routines that were discussed in Chapter 2.

Yet, the optimization of FLC is less trivial than is depicted in this simple scheme. Once the Fuzzy Logic model is chosen, the second most difficult step is undoubtfully the choice of the parameters that can be optimized, $\theta$, and the model parameters that should remain fixed, $\phi$. This is not only caused by overparametrizing. It is mainly caused by redundancy in the parameters.

Other models (NN, RBF, ...) usually restrict the free parameters to the size and position of the basis functions. Fuzzy models can alter size and position of the Fuzzy sets, the way how these sets are linked to each other in the rule base, the weights (truth factors) of the individual rules, and the parameters of the Fuzzy Sets or functions in the consequences of the Fuzzy rules. A lot of these parameters are redundant and this redundancy makes the optimization of the parameters a difficult task.

Not surprisingly a lot of research was done on the proper initialization of the parameters, rather than the optimization. A good initialization could even make the optimization unnecessary. This is certainly the case for Fuzzy controllers. Because of the property that FLC are quite
robust against model errors, they are easily used when large plant uncertainties exist. In that case, further optimization of the parameters is often not considered.

### 4.3.2 Choosing the model parameters

It is possible to give a near endless number of FL systems and derive all possible fixed and free parameters for each of these models. To fix the ideas, only one model is given here. Assume, therefore, a Takagi-Sugeno type of FLS with \( r \) rules of the form (4-18), given as

\[
R_i: \text{If } (u_{1i} = A_{1i}^j) \text{ and } (u_{2i} = A_{2i}^j) \text{ Then } (y^i = a_{0i}^j + a_{1i}^j u_{1i}^j + a_{2i}^j u_{2i}^j).
\]  

(4-23)

Assume, without loss of generality, that the model has two inputs and a single output. Each input is fuzzified using five triangular shaped FS, the first and last sets being open, and the FS for both inputs are normalized. In the case of a full and non-redundant Rule Base, the parameter vector \( \theta \) can have as much as 85 parameters and is build as

\[
\theta = [\theta_{u1}^T, \theta_{u2}^T, \theta_y^T, \theta_{RB}^T]^T
\]

(4-24)

with

- \( \theta_{u1} \) and \( \theta_{u2} \) the centres of the input triangular sets. Since the input sets are assumed triangular and normal, it can be shown that the only free parameters are the centres of each FS, thus reducing the number of parameters to five for each input.
- \( \theta_y \) contains three parameters \( a_{0i}^j, a_{1i}^j \) and \( a_{2i}^j \) for each rule in the rule base.
  - In the case of a full RB and no redundant rules, the number of rules equals \( p^m \) with \( m \) the number of inputs and \( p \) the number of FS per input.
  - In this example a full RB requires therefore 25 rules, leading to a total of 75 parameters in the \( \theta_y \) vector.
- \( \theta_{RB} \) defines the Rule Base parameters.

Note that in the case of a full rule base, the \( \theta_{RB} \) vector contains a combination of all FS of one input with the FS of the other input, e.g. \( \theta_{RB} = \{(1, 1), (1, 2), ..., (1, 5), (2, 1), ..., (5, 5)\} \). For a full and non-redundant RB, it is possible to say that \( \phi = \theta_{RB} \), i.e. the RB is fixed.

The immediate result of this approach is that the parameter vector \( \theta_y \) suffers from the curse of dimensionality: it grows exponential with the number of inputs and the number of FS per input. In practise one tries to define the FL system in the domain of interest only. The resulting problem is the choice of the combinations of input FS. It could very well be that choosing a different centre position for one of the input FS, results in a reduction of the number of needed rules in the RB.
On the other hand, examples can be given where different combinations of FS centre positions and output parameters, result in the same input-output behaviour for the complete FLS. In other words, the experimenter should expect redundancy in the parameters. Besides this, the RB in this example should be optimized in a different way from the optimization of e.g. the output parameters, since it links the different input sets in a discrete way.

Because of the complexity of the optimization of the Rule Base, and the limited success in practical applications, most experimenters decide to fix the RB and only optimize the Fuzzy Sets in the premises and consequences [36] [96] [133] [156] [247]. The choice of the parameters $\phi$ and $\theta$ is then done in three steps:

- The input sets are initialized, using e.g. clustering or linguistic modeling (explained in the next section). The input set parameters are then fixed, i.e. $\phi = [\theta_{u1}^T, \theta_{u2}^T]^T$ [64]. Wong [250] defines this part as structure identification.
- The Rule Base is modeled. This can be done using e.g. expert knowledge [66], or just by taking a full RB.
- The output parameters $\theta_y$ are optimized, using one of the optimization methods given in Chapter 2. This part is the true parameter identification [250].

Another method much used starts from a full RB where each rule is given a weight with low initial value. When optimizing, also the weights of the rules are updated when optimizing the output parameters. After convergence of the optimization routine, rules with weights below a user-defined threshold are taken out of the RB. Optionally a new optimization can be started with the reduced RB.

**Remark:** it should be noted that some references start from an initial Rule Base, for which the parameters are not changed during optimization. Instead, the optimization routine just adds new rules and assigns a weight to each added rule. The performance of the FLS with the added rule is then evaluated. This is done recursively until the model error is minimized. The resulting RB can have hundreds of rules, where a minimal RB would require e.g. 25 rules only.

In this thesis the modeling is considered as a means for compression of measurement data and the goal is the description of a measurement set with a low number of parameters. This type of modeling is therefore not considered in this thesis.
4.3.3 Initialization of Fuzzy Systems

The two best known methods for initialization of the input sets of a Fuzzy system are the use of *clustering* of the measurement data and the use of *linguistic modeling* of expert knowledge.

A. Fuzzy clustering

The final goal of clustering is the partitioning of the domain of interest into smaller clusters of measurement data. The number of clusters must be specified before clustering. The more clusters, the finer the approximation of the nonlinear system can be obtained, but also more parameters have to be estimated [89].

The reader should be aware that Fuzzy clustering only makes sense, if the measurement data contains distinct regions with concentrated measurement data. If clustering is performed on e.g. the input measurements only, while these measurements were done following an equidistant grid, then the results of the clustering don’t make much sense. In the best case, the measurements are split into two parts, denoting "high" and "low". Clustering should therefore not be used blindly, but it can be used whenever the inputs and outputs are based on a low number of discrete positions.

In recent years a large number of clustering methods have become known in the FL modeling field [19] [20] [89] [107] [250]. A brief list:

- C-means clustering [19] [250],
- Self-Organizing Maps (SOM) [107],
- Fuzzy learning vector quantization (FLVQ) [20],
- Fuzzy adaptive resonance theory (Fuzzy ART) [20] [133],
- growing neural gas (GNG) [20],
- fully self-organizing simplified adaptive resonance theory (FOSART),
- Gustafson-Kessel Fuzzy Clustering (GKFC) [89],
- LOLIMOT [64] [102] [161],
- The mountain method [253],
- Hybrid Clustering [250],
- Chen-Xi clustering [36].

It is beyond the scope of this thesis to give an overview of each of these methods. From this list, SOM and GKFC are generally considered as the best performing algorithms [20]. Still, a
severe drawback of both the SOM and the GKFC algorithms is that the final parameters are severely affected by the order of the input sequence [20].

Therefore, consider the following straightforward clustering method. It has been used by the author and showed to give good results at a low computational complexity. Clustering can be done on both inputs and outputs. This gives a rough model that is further optimized later [64]. To fix the ideas, the method is only explained for the clustering of the input measurements.

For simplicity of notation, it is assumed that all inputs are independent, such that the clusters can be derived for each input \( \mathbf{u}_i = [u_{i,1}, u_{i,2}, ..., u_{i,N}]^T \in \mathbb{R}^{N \times 1} \) separately. The expansion of the algorithm to multiple-input systems is straightforward.

The clustering algorithm explained here, is based upon the following premises:

- The Fuzzy Sets used, must be normal, such that

\[
\sum_{j=1}^{c} \mu_{i,j}(u_{i,k}) = 1 \quad \forall u_{i,k} \in \mathbb{R} \tag{4-25}
\]

with \( k \in \{1, 2, ..., N\} \), \( i \in \{1, 2, ..., m\} \) the index of the input and \( c \) the number of clusters. In practice, each cluster is covered by a FS.

- All measurements must be covered by a cluster, while no measurement may be covered by all clusters, or

\[
0 < \sum_{k=1}^{N} \mu_{i,j}(u_{i,k}) < N \quad j = 1, 2, ..., c. \tag{4-26}
\]

A collection of FS that satisfies equation (4-25), is given in FIGURE 4-5. The FS are triangular shaped sets, and the only free parameters are the centres of the different sets, \( c_1, c_2, ..., c_5 \). The slopes of each triangular shaped FS are completely determined by the preceding and the subsequent centres. Two centres are considered to be fixed at plus and minus infinity to guarantee open sets at both ends. The choice of a low number of free parameters, avoids redundancy in the parameters. Yet, the model provides enough freedom for clustering.

![FIGURE 4-5 A collection of normal Fuzzy Sets.](image-url)
The goal of the clustering is to position the cluster centres in such a way that all measurements are covered maximally by one of the sets. In other words, we seek to maximize the objective function

\[ J = \sum_{k=1}^{N} \sup_{j} \mu_{i,j}(u_{i,k}). \]  

(4-27)

Knowing that the supremum of each membership function must be smaller than one in all cases, the maximization of (4-27) can be changed to the minimization of the LS cost function

\[ C = \sum_{k=1}^{N} \left(1 - \max_{j} (\mu_{i,j}(u_{i,k}))\right)^2. \]  

(4-28)

Remark that this cost function is evaluated off-line and by nature, it is independent of the order how the measurements are presented to the cost function.

The selection of the number of clusters needed, is done roughly by matching the data to an elementary triangular FS. A single FS is used that covers the whole domain of interest from minus one to one as shown in FIGURE 4-6.

![FIGURE 4-6 Single FS, covering the domain of interest.](image)

The centre \( c_1 \) is swept from minus to plus one with a small step, and the performance index

\[ J = \frac{1}{2} \sum_{k=1}^{N} \mu_{i,1}(u_{i,k}) \]  

(4-29)

is evaluated on the whole data set. The performance shows local maxima where clustering of the measurement data can be expected. Each local maximum is selected as a possible cluster centre. The choice of this initial FS is rather arbitrary and a possible result is that some potential clusters are missed. If this is expected, the width of the FS can be made smaller. This typically leads to more initial clusters, such that pruning is necessary. Starting from these initial cluster centres, the iteration shown in Listing 4. is started.
The minimization of the cost \((4-28)\) can be done using a simple gradient descent algorithm, or any other optimization scheme that is described in Chapter 2. Because of the rather heuristic way how the initial clusters were chosen, the optimization can result in clusters that have
centre coordinates that come very close to each other. It is common practice to take those centres together. The pruning algorithm is therefore straightforward: if two centres lay within a user-defined distance (e.g. \( c_{i+1} - c_i < 0.05 \) in the example of FIGURE 4-5), one of the two centres is simply deleted, and a new optimization iteration is started.

To fix the ideas, consider the following example. An arbitrary measurement set \( y_k \), with \( k = 1, 2, ..., 700 \) and the \( y_k \) real values, is selected using 700 noisy data points, based on the centres \( \{-1, -0.8, -0.5, 0, 0.2, 0.7, 1\} \) which have been arbitrarily chosen. Three cases are considered with different noise levels \( \sigma^2 = 0.25 \), \( \sigma^2 = 0.01 \) and \( \sigma^2 = 10^{-4} \).

The results of the clustering are shown in FIGURE 4-7. The case \( \sigma^2 = 0.25 \) is shown on top, the case \( \sigma^2 = 0.01 \) in the middle while the bottom row shows the clustering for \( \sigma^2 = 10^{-4} \). The simulation data points are on the left side. The figures in the middle show the performance index. The detected local maxima are indicated by an asterisk. The figures on the right show the resulting fuzzy clusters after full convergence and pruning.

The figures show clearly that in the noisy case, only two levels ("high" and "low") are distinguished. This would also be the case if an equidistant measurement grid was chosen, or when the measurements would be equally distributed over the domain of interest. When less noise is used, the clustering algorithm succeeds in finding three clusters. From a first look on the data, it can be concluded that this is the most reasonable choice, indeed. At low noise levels, all clusters are found.

B. Linguistic modeling

Linguistic modeling is the name that is given to two different initialization operations on the FLS. In a first interpretation, linguistic modeling is used where expert knowledge is derived (e.g. by questionnaires) and translated into a Rule Base. A drawback of using linguistic modeling for initialization of the RB, is the danger for conflicting rules. It typically leads to flexible clustering instead of a grid-type clustering (see FIGURE 4-11 on page 105).

A second interpretation of linguistic modeling uses the expert’s interpretation of "high" and "low" to build the input and output Fuzzy Sets. This also, is done by questioning the expert. In practice this means that the cluster centres are placed such that the data is more or less covered. Although it is new to classic control theorists, linguistic modeling is a powerful tool for initializing FL systems [156] [181], prior to the optimization of the FL parameters. It should be compared with the initialization of e.g. NN systems, where the initial values are typically chosen randomly.

Linguistic modeling must not only be seen from the point of view of initialization of the FLS. It is also used to derive expert knowledge from the FLS after optimization of the FL parameters. This also is one of the reasons why FL researchers stress the importance of using
normal FS and minimal size Rule Bases. Assume again the example that was given earlier and where an extreme large number of rules is used in the RB. Not only does this lead to excessive calculations [36], it also makes a linguistic interpretation of the system impossible due to a large number of conflicting rules [64].

Note that the definition of linguistic modeling given here, may differ from the interpretation in some of the given references, e.g. Pedrycz [174] who describes Gustafson-Kessel clustering [89], calling it linguistic modeling and where clusters are replaced by granules. In this thesis the most straightforward definition is promoted, where linguistic modeling is based on linguistic variables only (see section 4.2.2 on page 77).

4.3.4 FLS parameter optimization

Once the model parameters $\theta$ and $\phi$ are chosen, the optimization is a matter of choosing the proper optimization scheme. Most methods make use of gradient (or backpropagation) methods for the optimization of the parameters [36] [96]. Note that the initial FL model or controller is already fairly well initialized by the user. The problem that gradient methods are limited and only reach a local minimum, is then of lesser importance and only plays a role for extreme nonlinear systems or noisy measurement data.

As an example, a Levenberg-Marquardt optimization scheme was implemented for the optimization of a Takagi-Sugeno Fuzzy system. To simplify the notations, a SISO system is considered here. The function $y = -\sin|2\pi u|$ is sampled using an equidistant grid on $u$ and modelled with the TS system given by equation (4-18) on page 88:

$$R_i: \text{If } (u^i_1 = A^i_1) \text{Then } (y^i = a^i_0 + a^i_1 u^i_1). \quad (4-30)$$

The input Fuzzy sets are chosen as in FIGURE 4-5 on page 95, such that the free parameters can be chosen as

$$\theta = [c^1, a^1_0, a^1_1, c^2, a^2_0, a^2_1, ... , c^r, a^r_0, a^r_1]^T \quad (4-31)$$

with the $c^i$ the centres of the input FS, and the $a^i_0$ and $a^i_1$ the parameters of the local linear systems. The initial values for the parameters are based on "expert" knowledge, and initialized as

$$[c^1, c^2, ... , c^r] = [-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1], \quad (4-32a)$$

$$[a^1_0, a^2_0, ... , a^r_0] = [0, 1, 0, -1, 0, -1, 0, 1, 0] \text{ and } (4-32b)$$
such that $r = 9$. The parameters are then optimized with the L-M optimization step (2-65) on page 42. The results of the modeling are shown in FIGURE 4-8. The initial model is shown on the left, while the model after optimization is shown on the right hand side. The upper figures show the Fuzzy input sets, and the transfer functions of the local-linear systems. The result of the modeling is given below, where the TS Fuzzy model is compared with the true function. It is clear that, despite the "reasonable" results of the expert modeling, the optimization is worth the effort.

For sake of completeness, the new parameter values are given below.

\[
[a^1_1, a^2_1, ..., a^r_1] = [1, 0, -1, 0, 1, 0, -1, 0, 1], \tag{4-32c}
\]

\[
[c^1, c^2, ..., c^r] = [-1.01, -0.76, -0.5, -0.24, 0, 0.25, 0.5, 0.77, 1.13], \tag{4-33a}
\]

\[
[a^1_0, a^2_0, ..., a^r_0] = [1.51, -0.24, -2.55, -1.62, -0.02, -1.65, -2.39, 0.04, 1.15] \tag{4-33b}
\]

\[
[a^1_1, a^2_1, ..., a^r_1] = [1.50, -1.69, -5.03, -2.25, 0.03, 2.34, 4.81, -2.12]. \tag{4-33c}
\]

## 4.4 Fuzzy Logic for control

### 4.4.1 Introduction

With Fuzzy control, the same controller configurations can be used as those that are used with NN control. Therefore the controller configurations that were given in section 3.3 on page 65, will not be repeated. Still, a few controller configurations can be considered as typical for FL control. These configurations are treated more in detail.

Some references [32] [36] [76] [103] [134] [156] try to optimize the FL parameters based on a look-and-feel approach, e.g. by using a second FL system to optimize the parameters of the main Fuzzy controller. The method is briefly explained in section 4.4.3 and commented.

### 4.4.2 Fuzzy Supervised Linear Control

In Fuzzy linear control, the actual control is performed by a linear controller, whose parameters are determined by a Fuzzy system. The Fuzzy system can be regarded as a supervisor that offers gain scheduling in an otherwise static linear system [64] [100] [103] [108] [170] [257] [260].

Consider the controller setup in FIGURE 4-9. A linear controller
In practise, the plant is only locally linear, e.g. caused by saturation effects. In linear control theory the model $H(s) = A(s)/B(s)$. $T_I$ and $T_D$ are the integrating and differentiating constants, $K$ is the controller gain and $T_1$ and $T_2$ are the time periods used for pole-placement.

In practise, the plant is only locally linear, e.g. caused by saturation effects. In linear control theory the model $H(s)$ is chosen such that it fits the actual plant at best, and the control parameters $T_I$, $T_1$, $T_2$ and $K$ are modeled such that an optimal control is provided for $H(s)$.
Instability can occur whenever the nonlinearities in the plant become too large such that the parameters don’t match the true plant any more.

The solution that is provided, chooses different operating points for the plant, and models the local linear plant transfer functions \( H_1(s), H_2(s), \ldots H_M(s) \). Each local linear plant has its own controller transfer function \( G_i(s) \) with \( i = 1, 2, \ldots, M \) and with its own parameters \( T_{l,i}, T_{D,i} \) and \( K_i \).

Based upon the different operating points, a FLS is set up that determines the controller parameters in each operating point. Using a linguistic rule (4-1), the FLS is typically chosen as a singleton Fuzzy system

\[
R_i: \text{If (Operating point is } A_i) \text{ Then } (T_I \text{ is } B_{1i}) \text{ and } (T_D \text{ is } B_{2i}) \text{ and } (K \text{ is } B_{3i}),
\]

or as a Takagi-Sugeno system

\[
R_i: \text{If (Operating point is } A_i) \text{ Then } G_i(s) = K_i \left( \frac{s^2 T_{l,i} T_{D,i} + s T_{l,i} + 1}{s T_{l,i}} \right).
\]

The modeling of processes using Fuzzy Linear Control offers a way to combine both FL and classic modeling techniques, taking the best of two methods.

### 4.4.3 Fuzzy Supervised Fuzzy Control

This type of FL controller is used in different references [32] [36] [76] [103] [134] [156]. The basic scheme is given in FIGURE 4-10. The linear or nonlinear plant is controlled by a FL Controller, which is initialized to perform e.g. a regular PID action. The controller is then adapted using a FL system which acts as an observer on the controlled system. In FIGURE 4-10, the closed loop system is monitored at three different places by the observer, but in practise some inputs (e.g. the monitoring of \( u_k \) or \( y_k \)) are omitted and only one or two inputs are considered.
The observer alters the FLC parameters based on its own FL rules. The remaining question is how the observer FLS should be designed such that the controller is well optimized. In practice this is often done based on expert knowledge, but it appears to be more correct to state that the observer is designed based on heuristic rules.

It is the author’s belief that, although this type of control is largely accepted in the FL domain, it should be avoided. Other optimization schemes available are known to perform excellent both for linear and nonlinear systems and offer adaptive optimization rates (e.g. Levenberg-Marquardt), robustness against local minima (e.g. using momentum terms) and guaranteed performance (e.g. gradient methods). The references given typically use the observer FLS as a substitute for a steepest descent optimization and offer little or no added optimization performance.

4.5 A scheme for building FL systems

4.5.1 Introduction

The previous sections gave a brief overview of a number of existing FL modeling techniques for FL systems and controllers. One of the complaints much heard in FL modeling, is the lack of a uniform modeling scheme [29] [66] [96] [100] [124] [156] [167]. Based on the insights derived from the discussions in the previous sections and in Chapter 2, however, it should be possible to exclude a number of FL modeling techniques and even provide a general modeling scheme.

4.5.2 Basic assumptions

Assumption 4.3 Regardless of the use of the FL system (model or controller), this thesis considers the case where input-output measurements are available to optimize the model parameters.
If Assumption 4.3 doesn’t apply, the experimenter would find himself in the position that he hasn’t any objective means of evaluating the model or controller. The modeling is then straightforward: the experimenter can pick any FL model, and initialize the parameters based on linguistic modeling (see section 4.3.3 on page 94) based on expert knowledge on the system. This would end the modeling. For further optimization, the availability of measurement data seems a reasonable assumption.

**Assumption 4.4** The input Fuzzy Sets cover the whole domain of interest and are normal.

The demand that the FS cover the whole domain of interest is based on *observability* demands. If certain parts of the domain of interest would not need to be observable, it is clear that the domain of interest is just not correctly defined.

The demand for normalization is less obvious and is more based on empirical reasons. Assume a FL system that makes use of not-normal input sets. Then there is a potential risk that certain input measurements are not very well covered by at least one Fuzzy input set. This lack of coverage at the input can lead to a range of problems, e.g. when determining the output sets or when stability issues come into play. For normal sets, it is possible to state that any input is fully covered by at least one FS. For not-normal sets, the coverage could be from one to practically zero, so that no sufficient coverage can be guaranteed.

The demand for using normal FSs practically comes down to the use of triangular and trapezoidal shaped sets, such as those displayed in figures 4-5 and 4-7. Some authors prefer the use of e.g. Gaussian shaped FS instead of triangular shaped sets, because Gaussian shaped sets have finite higher order derivatives. The following three properties are highly appreciated in search for the ideal Fuzzy Set:

- It should be normal, to guarantee observability of the input measurements.
- It should have finite higher order derivatives [247] such that gradient methods can be used for optimization of the model parameters.
- The transfer function of the FS must be analytical, for various reasons: the derivatives are easier to calculate, lesser memory is needed to store the FS in memory, and a higher calculation precision is reached.

To date, no function exists that complies with all three of these demands. Triangular and trapezoidal shaped sets do not guarantee finite higher order derivatives. Gaussian shaped sets can be expressed analytical and have finite higher order derivatives, but can’t offer normal FS. The Gustafson-Kessel algorithm results in normal sets with finite higher order derivatives, but they can’t be expressed analytically and are calculated for a fixed number of point in the domain of interest.
Because of the need for finite higher order derivatives, e.g. for parameter optimization or stability issues, some authors relax the normalization demand to the inequality

\[ c \leq \sum_{j=1}^{r_i} \mu_{A_j}(u_i) \leq 1 \]  

(4-37)

with \( c \) a user defined parameter (e.g. \( c = 0.9 \)) such that it is possible to use Gaussian shaped FS.

Accepting Assumption 4.4 automatically leads to the use of grid-type Fuzzy clustering, which is shown in FIGURE 4-11 for a SISO Mamdani type FLS. The advantage of the grid-type clustering is an easier optimization. It is observed that Flexible hypercube clustering usually needs far less rules in the Rule Base, but the optimization of the clusters is more demanding [133] and easily suffers from redundancy in the parameters.

Assumption 4.5 Within the domain of interest a full Rule Base is used.

This assumption is based upon controllability demands. It makes sense to demand that a rule exists in the Rule Base for each possible combination of Fuzzy input sets that is likely to occur. If not, the model or the controller could reach an undefined state. Remark that this assumption is restricted to the domain of interest in order to reduce the curse of dimensionality effects.

Assumption 4.6 The Fuzzy Inference is done using the product-sum t-norm and s-norm.

In other words: although a large number of inference mechanisms could be used (see TABLE 4-2. on page 84), the product-sum combination is favoured. Here also, it would be possible to use the reasoning that universal approximation is guaranteed in all cases shown in the table. Moreover, different references [87] [90] [170] report that the difference with other inference schemes are only marginal, such that the product-sum combination is indeed sufficient for
Fuzzy modeling. Product-Sum inference has a low complexity, leading to more insight in the operation of the model. The best reason for using product-sum, however, is that it makes the calculation of the parameter optimization steps simpler.

**Corollary 4.1** Under Assumption 4.6, it is always possible to combine redundant rules in the Rule Base into one single Fuzzy rule, such that the Rule Base is always minimal.

**Proof:** see APPENDIX A.

**Assumption 4.7** The defuzzification of the output Fuzzy Set is done using the Centre-of-Sums method.

Based on the research of Hellendoorn and Thomas [88], and Matiá et al. [150] (see section 4.2.8) it can be concluded that CoS provides superior performance compared with other defuzzification methods. A demand for using CoS is that the sum-operator should be used in the Inference Engine. This demand is met with Assumption 4.6.

**Theorem 4.1** Under assumptions 4.6 and 4.7, any Mamdani FL system that makes use of Fuzzy output sets with a constant area, can be replaced by the Takagi-Sugeno FL system

\[
y = \sum_{i=1}^{r} \mu_{A_i}(u_1)\mu_{A_i}(u_2)\ldots\mu_{A_i}(u_m) \frac{b_i}{f_i(u_1, u_2, \ldots, u_m)}
\]

where the function \(f_i(u_1, u_2, \ldots, u_m)\) is a polynomial or a nonlinear function with constant parameters for each rule and \(r\) is the number of rules in the RB.

**Proof:** see APPENDIX A.

One of the choices that must be made in the design of a FL system, is the choice between a Mamdani type or a Takagi-Sugeno type system. From Theorem 4.1 it follows that the most commonly used Mamdani Fuzzy system can be replaced, using a TS system with hyperbolic transfer functions.

It is our experience that the use of hyperbolic transfer functions easily leads to a large number of local minima in the cost function. The optimization of the parameters then becomes a difficult task. If a nonlinear transfer function is demanded for each rule, a better choice could be the use of polynomial or hyperbolic tangent transfer functions. These have shown to give excellent results when used in Neural Networks, or Neurofuzzy systems. We therefore propose to use TS Fuzzy systems of the type
A scheme for building FL systems

\[ y = \sum_{i=1}^{r} \mu_{A_i}(u_1)\mu_{A_i}(u_2)\ldots\mu_{A_i}(u_m)w_if_i(u_1, u_2, \ldots, u_m) \]  (4-39)

where \( w_i \) is a weighting factor, and \( f_i(u_1, u_2, \ldots, u_m) \) can be a polynomial function, or e.g. a NN.

4.5.3 The modeling scheme

If the assumptions given in the previous section are accepted, the FL modeling can be summarized as follows.

1. Perform the necessary Input-Output measurements for the model or the controller.
2. Choose a cost function, based on the properties given in Chapter 2. Mostly a LS cost function is selected.
3. The FL model proposed by the author is a Takagi-Sugeno Fuzzy system, with normal triangular and trapezoid transfer functions for the input FS, and an input-output relationship given by equation (4-39).

A first choice for the local functions should be

\[ f_i(u_1, u_2, \ldots, u_m) = 1, \]  (4-40)

such that the model becomes a simple singleton FL system based on the \( w_i \) parameter. If the performance doesn’t satisfy the demands, a more complex transfer function could be chosen, such as the linear system

\[ f_i(u_1, u_2, \ldots, u_m) = \alpha_0^i + \alpha_1^i u_1 + \alpha_2^i u_2 + \ldots + \alpha_m^i u_m \]  (4-41)

with parameters \( \alpha_0^i, \alpha_1^i, \ldots, \alpha_m^i \). For severe nonlinear systems, even more complex transfer functions are possible, such as NN \( f_i(u_1, u_2, \ldots, u_m) = f_{NN_i}(u_1, u_2, \ldots, u_m) \) or RBF \( f_i(u_1, u_2, \ldots, u_m) = f_{RBF_i}(u_1, u_2, \ldots, u_m) \) functions.

4. Initialize the input FSs using clustering or linguistic modeling (section 4.3.3.).
5. Based on controllability demands, we know that a full Rule Base is needed (Assumption 4.5), while Corollary 4.1 shows that a minimal Rule Base is sufficient. The Rule Base is therefore built as a combination of the input Fuzzy Sets of all inputs, within the domain of interest \( \Psi \) (see the definition of the domain of interest on page 16).
6. Optimize the TS parameters, e.g. the \( w_i \) and \( \alpha_0^i, \alpha_1^i, \ldots, \alpha_m^i \) parameters if a linear transfer function was chosen. Since the input FS are fixed, and a full Rule Base is used, the choice of a singleton or linear transfer function results in the optimization of local linear systems. Therefore, the use of gradient methods or a L-M optimization seems sufficient.
4.6 Conclusions

This chapter gave an overview of existing Fuzzy Logic modeling techniques for system identification and control. The overview shows that the name "Fuzzy Logic" covers a large number of possible transfer functions based on different definitions for the input and output sets, different interpretations of the modus ponens reasoning, different transformations between crisp values and Fuzzy values, and even different model topologies.

Because of this broad definition of what is Fuzzy Logic, there is a demand for a clear and uniform approach for the design of FL systems. This chapter has tried to provide such an approach, based on a number of reasonable assumptions and the knowledge that a general identification scheme exists for nonlinear system modeling.

The question may arise why it is needed to use Fuzzy Logic after all. The reason is that FL provides a tremendous look-and-feel added value, mainly for automatic control. Initial control systems can be set up based upon human intuition, rather than using complex math. This might sound as if the control research is set back for twenty years, but it must be clear that there is a demand for such systems from the industry.

It is not only this look-and-feel that speaks in favour of FL systems. Besides that, Fuzzy Logic is also a very powerful modeling technique that can be combined with existing techniques, such as linear or Neural Network modeling. Because of that, any linear control system can be replaced by a Fuzzy Logic Controller, under mild conditions regarding the FLC parameters [89] [107] [165]. It is therefore very appealing to use a FLC instead of a classic controller, and FL gains interest in the practical control domain [1].
Abstract: This chapter studies the absolute stability of Fuzzy Logic control systems. It is shown that SISO Fuzzy systems can be written as $NL_q$ systems. This is illustrated on a simple SISO Mamdani-type controller. The stability theorems that come with $NL_q$ theory, therefore, also apply for Fuzzy systems. The theory is applied on a simple control system that is set up with FL control and for which the parameters are optimized, based on guaranteed stability.

In this thesis $NL_q$ theory is extended to $NL_p$ theory for parallel nonlinear systems, in order to include MIMO Fuzzy systems. A number of stability theorems are formulated for $NL_p$ systems, and it is shown that MIMO Fuzzy systems can be written as $NL_q$ systems.
5.1 Introduction

The stability of linear systems has been studied for decades. To date, both robustness and stability constraints can be put on systems that are linear in the parameters. The stability analysis techniques not only provide absolute stability, but also relative stability. In the latter case limitations can be imposed on the control system regarding overshoot and settling time properties.

The story is different for nonlinear systems. A variety of analysis techniques exist, most of which are based on Lyapunov theory [42] [51] [171]. Lyapunov theory provides a theorem for global asymptotic stability (GAS) only. Moreover it is a sufficient proof for stability and not an necessary one. The Lyapunov criterion can be rather conservative, leading to control systems that are too slow, dependent on the chosen Lyapunov function.

The stability theorems that are proven in NLq theory [205] [207] are based on Lyapunov theory too, but also provide robust local stability criteria, next to global asymptotic stability. This is a relaxation on the constraints that are put on the parameters. In practise, this robust local stability is often sufficient, even for larger input variations (semi-global stabilization).

In section 5.2, NLq is briefly introduced, and then applied on SISO systems in section 5.3. Based on the optimization theory of the previous chapter, a FL controller is optimized in section 5.4 with guaranteed global asymptotic stability of closed loop systems. The stability constraint is imposed on the system within the cost function, where a dynamic balance is made between the FLC performance and its stability.

NLp theory is explained in section 5.5 and applied on MIMO Fuzzy systems in section 5.6. A number of proofs are given for the stability of NLp systems.

5.2 NLq theory for stability analysis

This section gives an introduction to NLq theory and briefly describes the internal stability and I/O stability theorems that come with the theory.

5.2.1 Definition of NLq systems

Definition 5.1 Suykens [205] defines an NLq system as a concatenation of q nonlinear and linear discrete time subsystems, denoted in state space form as

\[
\begin{align*}
    p_{k+1} &= \Gamma_1(V_1 \Gamma_2(V_2 \ldots \Gamma_q(V_q p_k + B_q w_k)) \ldots + B_1 w_k) \\
    e_k &= \Lambda_1(W_1 \Lambda_2(W_2 \ldots \Lambda_q(W_q p_k + D_q w_k)) \ldots + D_1 w_k)
\end{align*}
\]
and which relates to a recurrent network of the form

\[
\begin{align*}
\begin{cases}
p_{k+1} &= \sigma_1(V_1 \sigma_2(V_2 \ldots \sigma_q(V_{q} p_k + B_q w_k)) \ldots + B_1 w_k) \\
e_k &= \sigma_1(W_1 \sigma_2(W_2 \ldots \sigma_q(W_{q} p_k + D_q w_k)) \ldots + D_1 w_k).
\end{cases}
\end{align*}
\] (5-2)

The \(V_i, W_i, B_i, D_i\) matrices denote the linear part of the \(\text{NL}_q\) system. \(\sigma_i\) is a sector bounded linear or nonlinear function on the sector \([0, 1]\), and with the property

\[
0 \leq \frac{\sigma_i(\omega)}{\omega} \leq 1.
\] (5-3)

\(\sigma_i\) applied on a vector or matrix, is taken elements wise. \(\Gamma\) and \(\Lambda\) are diagonal matrices with diagonal elements \(\sigma_i(\omega)/\omega\) such that \(\|\Gamma\|_\infty \leq 1\) and \(\|\Lambda\|_\infty \leq 1\) with \(\|\cdot\|_\infty\) the 1, 2 or infinity norm. \(p_k\) is the state space vector of the \(\text{NL}_q\) system \(w_k\) the exogeneous input and \(e_k\) the regulated output (standard plant).

Useable functions for \(\sigma\) are the linear function \(\text{lin}(\omega) = \omega\), the \(\text{tanh}(\omega)\) function used in many neural networks and the \(\text{sat}(\omega)\) function, defined as

\[
sat(\omega) = \begin{cases} 
\omega & |\omega| \leq 1 \\
1 & \omega > 1 \\
-1 & \omega < -1
\end{cases}
\]

\[
\begin{array}{c}
sat(\omega)
\end{array}
\]

\[
\begin{array}{c}
\omega
\end{array}
\]

\[
\begin{array}{c}
-1
\end{array}
\]

\[
\begin{array}{c}
1
\end{array}
\]

\[
\begin{array}{c}
0
\end{array}
\]

Definition 5.2 An equivalent representation for an \(\text{NL}_q\) system is [205]

\[
p_{k+1, e} = \left( \prod_{i=1}^{q} \Omega_i R_i \right) \begin{bmatrix} p_k \\ w_k \end{bmatrix}
\] (5-5)

with

\[
\begin{bmatrix}
p_{k+1} \\
e_{k}^{ext} \\
w_k
\end{bmatrix} = \begin{bmatrix} p_{k+1} \\
e_{k}^{ext} \\
w_k
\end{bmatrix}.
\] (5-6)

\(p_k\) and \(w_k\) are defined above, and \(e_{k}^{ext}\) is defined such that its dimension is equal to the dimension of \(w_k\).
Equation (5-5) is indeed an equivalent representation of (5-1). This can be proven by straightforward calculation and defining the \( \mathbf{\Omega}_i \) and \( \mathbf{R}_i \) matrices as

\[
\mathbf{\Omega}_i = \begin{bmatrix}
\mathbf{\Gamma}_i & \\
\mathbf{\Lambda}_i & \\
\mathbf{J}
\end{bmatrix},
\]

(5-7)

\[
\mathbf{R}_i = \begin{bmatrix}
V_i & 0 & B_i \\
0 & W_i & D_i \\
0 & 0 & I
\end{bmatrix}
\]

for \( i = 1, 2, \ldots, q-1 \) and \( \mathbf{R}_q = \begin{bmatrix}
V_q & B_q \\
W_q & D_q \\
0 & I
\end{bmatrix} \). (5-8)

Remark that \( \|\mathbf{\Omega}_i\|_i = 1 \) in the particular cases \( l = 1 \) and \( l = \infty \), while in the general case \( \|\mathbf{\Omega}_i\|_i \leq 1 \) because \( \|\mathbf{\Gamma}_i\|_i \leq 1 \) and \( \|\mathbf{\Lambda}_i\|_i \leq 1 \).

5.2.2 NLq stability theorems

Theorem 5.1 A sufficient condition for global asymptotic stability of the autonomous NLq system

\[
p_{k+1} = \sigma_1(V_1 \sigma_2(\ldots \sigma_{q-1}(V_{q-1} \sigma_q(V_q p_k))\ldots)))
\]

(5-9)

with \( V_i \in \mathbb{R}^{n_i \times n_i+1} \) \( (n_1 = n_{q+1} = n) \) and \( \sigma_i \) a sector bounded function that is applied elements wise, is that a diagonal matrix \( \mathbf{\Delta} \) can be found such that

\[
\|\mathbf{\Delta} V_{tot} \mathbf{\Delta}^{-1}\|_2 = \beta(\mathbf{\Delta}) < 1 \text{ with } V_{tot} = \begin{bmatrix}
0 & V_2 & 0 \\
0 & V_3 & \ddots \\
& & \ddots \\
0 & V_q & 0 \\
V_1 & 0 & \ldots
\end{bmatrix}.
\]

(5-10)

Proof: See Suykens et al. [205].

Theorem 5.2 A sufficient condition for I/O stability of the non-autonomous system (5-1) is that a diagonal matrix \( \mathbf{\Delta} \) can be found, such that

\[
\|\mathbf{\Delta} V_{tot} \mathbf{\Delta}^{-1}\|_2 = \beta(\mathbf{\Delta}) < 1
\]

(5-11)

with
NLq theory for stability analysis

\[ V_{\text{tot}} = \begin{bmatrix} 0 & V_2' & 0 \\ 0 & V_3' & \vdots \\ \vdots & \ddots & \vdots \\ 0 & V_q' & 0 \end{bmatrix} \quad \text{and} \quad V_i' = \begin{bmatrix} V_i' & B_i' \\ 0 & 1 \end{bmatrix}. \quad (5-12) \]

**Proof:** See Suykens et al. [205].

The above stability theorems only make use of a diagonal matrix \( \Delta \), which guarantees global asymptotic stability for the autonomous system, and I/O stability with finite \( l_2 \) gain in the non-autonomous case. In practice these theorems are too conservative.

Suykens et al. [205] give less conservative versions of the theorems, making use of matrices of full rank instead of the diagonal matrices. From these less conservative theorems, the condition number factor theorem is given here.

**Definition 5.3** The condition number \( \kappa(P) \) is by definition equal to

\[ \kappa(P) = \|P\|_2 \|P^{-1}\|_2. \quad (5-13) \]

**Theorem 5.3** A sufficient condition for global asymptotic stability of the autonomous NLq system is that matrices \( P_i \) of full rank can be found such that

\[ \prod_{i=1}^{q} \kappa(P_i) \|P_{\text{tot}}V_{\text{tot}}P_{\text{tot}}^{-1}\|_2^2 = \beta(P_{\text{tot}}) < 1 \quad (5-14) \]

with \( P_i \in \mathbb{R}^{n_i \times n_{i+1}} \) \( (n_1 = n_{q+1} = n) \), \( P_{\text{tot}} = \text{blockdiag}\{P_2, P_3, \ldots, P_q, P_1\} \) and \( V_{\text{tot}} \) given in (5-10).

**Proof:** See Suykens et al. [205].

An analogous theorem exists for the I/O stability of the non-autonomous system, making use of the \( V_i \) matrices instead of the \( V_i' \) matrices.
Condition relaxation

The condition of Theorem 5.1 can be relaxed to a local stability criterion [204] such that a sufficient condition for stability of an NLq system is reached if the condition number product

\[ \varphi(P_1, P_2, \ldots, P_q) = \prod_{i=1}^{q} \kappa(P_i) \]  \hspace{1cm} (5-15)

is minimized, subject to the condition

\[ \|P_{tot}V_{tot}P_{tot}^{-1}\|_2 = \beta(P_{tot}) < 1. \]  \hspace{1cm} (5-16)

In practice, this condition is much easier reached. \( \varphi \) is related to the attraction zone where the NLq is stable, and should be as small as possible. From (5-14) it can easily be seen that the system is globally asymptotically stable whenever \( \varphi(P_1, P_2, \ldots, P_q) \leq 1 \).

In section 5.4.3 a more detailed example is given, illustrating the use of the condition number product. It is shown that the use of the relaxed condition turns out to be the only option, even for control loops of low complexity.

5.3 Representation of SISO Fuzzy Logic Control systems within NLq theory

5.3.1 Introduction

Lately, much effort has been put to overcome the major drawback of FL systems: how to find an easy to use stability theorem? Although Fuzzy Systems are locally linear, the global transfer function is of a nonlinear nature.

Although stability of FL systems has been addressed in many papers, most of these papers focus on the local-linear properties of Takagi-Sugeno controllers [109] [211] [256], while here we study the global asymptotic stability and I/O stability of Mamdani FLC, with a few assumptions on normalization of the input and output vectors as it was also done in [32], [120] and [212].

The stability criterion is based on FL controllers that use triangular and trapezoidal shaped Fuzzy Sets. It assumes that systems can be described with either a linear model or a FL model [36] [209], or a combination of both. It is shown that a class of FL systems can be described as a combination of linear activations functions \( \sigma_i(x) = \text{lin}(x) \) and saturating activation functions \( \sigma_i(x) = \text{sat}(x) \).
This section gives an application of \( \text{NL}_q \) theory on SISO Fuzzy Logic Controllers. The criterion that is given, provides a sufficient proof for the stability of Fuzzy Additive Systems (further denoted as FAS [119]). The given \( \text{NL}_q \) stability criterion proves to be easy to use, once the state space representation of the controlled system is derived. For this, it is assumed that a plant can be accurately described (Certainty Equivalence Principle) using a linear or nonlinear model, in this case a FL system.

5.3.2 Formal representation of a FAS with trapezoidal shaped Fuzzy Sets

Consider a SISO Fuzzy Additive System \( F: \mathbb{R} \rightarrow \mathbb{R} \) that stores \( r \) rules of the form

\[
R_j: \text{If } x = A_j \text{ then } y = B_j, \quad j = 1, 2, \ldots, r. \tag{5-17}
\]

The input fuzzy sets \( A_j \) and output fuzzy sets \( B_j \) are defined as triangular or trapezoidal shaped sets, as shown in FIGURE 5-1. All "then" parts \( B_j \) are summed to give the output set \( B(y) \), defined as

\[
B(y) = \sum_{j=1}^{r} \mu_j(x)B_j(y). \tag{5-18}
\]

The membership function \( \mu_j(x) \) with \( \mu_j: \mathbb{R} \rightarrow [0, 1] \) measures the degree to which the input \( x \) belongs to the fuzzy set \( A_j \). The translation from \( x \) to \( \mu_j(x) \) is the fuzzification part of the FLS.

**Assumption 5.1** The system inputs and outputs are normalized, such that \( x \in [-1, 1] \) and \( y \in [-1, 1] \).

**Assumption 5.2** The input sets are normal, such that the following equation holds:

\[
\text{FIGURE 5-1} \quad \text{[LEFT]} \text{ Normal fuzzy input sets. [RIGHT] Output sets with a constant area.}
\]
Typical Fuzzy Sets that comply with this assumption, are the combination of triangular and trapezoidal sets as shown in FIGURE 5-1[LEFT]. The first and last sets of the FLS are chosen as open FSs. This not only guarantees normal operation. It also guarantees that inputs beyond the normalized region are allowed and no "black holes" exist in the FL controller, i.e. regions that are not covered by the FL controller such that the plant remains in an undefined state.

**Assumption 5.3** All output sets $B_j$ have the same area $b_j$ with

$$ b_j = \int B_j(y) dy = b. $$(5-20)

This assumption can easily be achieved by choosing the same base set for each output set $B_j$, as shown in FIGURE 5-1[RIGHT].

Remark that this almost always leads to not-normal output sets since the only remaining tuneable parameter is the centre of the Fuzzy Set. The centres $c_j^B$ of each output set are defined as

$$ c_j^B = \frac{\int yB_j(y) dy}{\int B_j(y) dy} = \frac{1}{b} \int yB_j(y) dy. $$

The generalized output sets $B_j$ have an output function $B_j(y): \mathbb{R} \rightarrow [0, 1]$, such that the rules $R_j$ form an arbitrary linear or nonlinear mapping $R_j: \mathbb{R} \rightarrow \mathbb{R}$.

A major concern when choosing all outputs with the same width is the question whether the generalization and approximation capabilities of the Fuzzy System are not harmed. Indeed, when looking at the output sets of FIGURE 5-1[RIGHT] one can easily find a scenario where gaps arise in between the fuzzy sets and the output fuzzy sets seemingly don’t cover the whole output space anymore.
This problem was also mentioned in Chen et al. [38] with respect to RBF networks, where it is shown that the width of a RBF node can be fixed to provide a simpler training strategy based on the centres of the nodes only.

**Assumption 5.4** The rule base is full and non-redundant within the domain of interest.

The demand that the rule base is full means that all possible inputs should be covered by the rule base. For a SISO system, this implies that each input set is mapped to exactly one rule and the number of rules equal the number of input sets \((r = R)\). It is, however, not necessary that all output sets are used for the inference mechanism.

The assumption that the rule-base is non-redundant implies that no contradictory rules are used for the same input set. This seems a rather trivial demand. Yet, FL learning schemes exist that just keep adding rules with a given weight factor, where just a few rules could be sufficient for the same input-output relationship.

**Assumption 5.5** The crisp output value is calculated using the Centre of Sum defuzzification operator.

The FL system described is defined as a Standard Additive Model (SAM) [119] with transfer function

\[
FLS(x) = \text{Centroid} \left( \sum_{j=1}^{r} \mu_j(x)B_j(y) \right) = \frac{\int_{\mathbb{R}} y \sum_{j=1}^{r} \mu_j(x)B_j dy}{\int_{\mathbb{R}} \sum_{j=1}^{r} \mu_j(x)B_j dy}.
\]  

(5-22)

With assumption 5.3 this can be written as

\[
FLS(x) = \sum_{j=1}^{r} \frac{\mu_j(x)b_j c_j^B}{\sum_{j=1}^{r} \mu_j(x)b_j} = \sum_{j=1}^{r} \frac{\mu_j(x)c_j^B}{\sum_{j=1}^{r} \mu_j(x)}
\]  

(5-23)

and with assumptions 5.2 and 5.4

\[
FLS(x) = \sum_{j=1}^{r} \mu_j(x)c_j^B.
\]  

(5-24)
This equation describes the fuzzy relationship from a given input $x$ to a given output $y$. It is static, although for controllers sometimes an integrating relationship is preferred. In the sequel a more general SISO representation of the FL system is used, based on the concatenation of a linear system and a FL system, as shown in FIGURE 5-2.

The representation of this FL system is then written in state space form as

$$
\begin{align*}
    s_{k+1} &= As_k + Bx_k \\
    y_k &= FLS(Cs_k + Dx_k) = \sum_{j=1}^{r} \mu_j(Cs_k + Dx_k)c_j^B.
\end{align*}
$$

It is clear that equation (5-25) reduces to (5-24) by setting the matrices $A$, $B$ and $C$ to zero and $D = 1$. Remark that with this notation it is possible to handle inputs $x_k$ that have more than one element and this system could be regarded as a MISO configuration. Yet, the FL system itself should remain SISO.

### 5.3.3 Fuzzy Systems as NLq’s

Based on the $sat(\omega)$ function it is possible to define a fuzzy trapezoidal set as given in FIGURE 5-3. These sets can be formulated as

$$
\mu_j(x) = \mu_{(x = A_i)} = \frac{1}{2} sat\left(2 \frac{x - c_i + k_i^L}{\sigma_i^L} + 1\right) + \frac{1}{2} sat\left(-2 \frac{x - c_i - k_i^R}{\sigma_i^R} + 1\right).
$$

Triangular shaped sets are in fact also trapezoidal shaped sets, with zero kernel, such that $k_i^L$ and $k_i^R$ equal zero.

**Definition 5.4** Define the positive definite shift vectors

$$
K^R = [k_1^R, k_2^R, \ldots, k_r^R]^T \text{ and } K^L = [k_1^L, k_2^L, \ldots, k_r^L]^T.
$$

**Definition 5.5** Define the slope vectors.
Definition 5.6 Define the centre vectors
\[
e^A = [c^A_1, c^A_2, \ldots, c^A_r]^T \quad \text{and} \quad e^B = [c^B_1, c^B_2, \ldots, c^B_r]^T
\]
in which the centres for the fuzzy input sets \(c^A_i\) are defined as in FIGURE 5-3, while the centres for the output sets \(c^B_i\) are defined in (5-21). The indices 1, 2, ..., \(r\) correspond with the \(r\) rules used for the fuzzy inference. It is not necessary that the FL rules are ordered in any way.

\[S_L = \left[ \frac{1}{\sigma^L_1}, \frac{1}{\sigma^L_2}, \ldots, \frac{1}{\sigma^L_r} \right]^T \quad \text{and} \quad S_R = \left[ \frac{1}{\sigma^R_1}, \frac{1}{\sigma^R_2}, \ldots, \frac{1}{\sigma^R_r} \right]^T.\]  

\(\text{(5-28)}\)

Definition 5.7 The operator \(\odot\) denotes the Hadamard-Shur product, also written as “.*” in different mathematical programs. The operator
\[
P \prod_{j=1}^{r} \xi^{[j]}\]
\(\text{(5-30)}\)
denotes the elements wise product of the \(\xi^{[j]}\) matrices.

Lemma 5.1 equation (5-26) can be written in the form
\[
\Phi(x) = \frac{1}{2} \left[ I_r \cdot I_r \right] \text{sat} \left( 2 \begin{bmatrix} S_L^T \end{bmatrix} x + 1 \begin{bmatrix} (K_L - c^A) \odot S_L^T \\ (K_R + c^A) \odot S_R^T \end{bmatrix} \right)
\]
\(\text{(5-31)}\)

with \(\Phi = [\mu_1, \mu_2, \ldots, \mu_r]^T\) and \(I_r\) an identity matrix of size \(r\).

Proof: This can be proven with straightforward calculation.
Lemma 5.2 The FL system (5-25) can be written as an NL₂ system.

Proof: See APPENDIX A. □

Based upon the knowledge that a SAM Fuzzy Logic system can be represented as an NLₚ, all stability theorems that come with NLₚ theory, can be used. An example is given in the next section.

5.3.4 Application

Assume the concatenation of a linear system and a known nonlinearity that can be modeled using a FL system. The system is controlled with a FL controller. The closed loop scheme is given in FIGURE 5-4

The elements of the closed loop system of FIGURE 5-4 are given by

\[
\begin{align*}
\text{FLC:} & \quad \begin{cases}
z(k + 1) = E^c z(k) + F^c e(k) \\ u(k) = \text{FLC}(G^c z(k) + H^c e(k))
\end{cases} \\
\text{M1:} & \quad \begin{cases}
x^1(k + 1) = A^1 x^1(k) + B^1 u(k) \\
r^1(k) = C^1 x^1(k) + D^1 u(k)
\end{cases} \\
\text{FLS:} & \quad \begin{cases}
\nu(k + 1) = E^s \nu(k) + F^s r^1(k) \\
r^2(k) = \text{FLS}(G^s \nu(k))
\end{cases} \\
\text{M2:} & \quad \begin{cases}
x^2(k + 1) = A^2 x^2(k) + B^2 r^2(k) \\
y(k) = C^2 x^2(k) + D^2 r^2(k)
\end{cases}
\end{align*}
\]  

(5-32) (5-33) (5-34) (5-35)

This closed loop system can be written as an NL₃ system. To show this, define the closed loop system state space vector \( \mathbf{p}(k) = [z(k); x^1(k); \nu(k); x^2(k)] \) and the exogeneous input \( \mathbf{w}(k) = [d(k) \ 1]^T \). To simplify notations, define the following FLC matrices
\[
\alpha^c = \begin{bmatrix} 2S^L, c \\ -2S^R, c \end{bmatrix}, \quad \beta^c = \begin{bmatrix} 1 + 2(K^L, c - c^L, c) \odot S^L, c \\ 1 + 2(K^R, c + c^L, c) \odot S^R, c \end{bmatrix}, \quad \gamma^c = (c^{R, c})^T \frac{1}{2} \begin{bmatrix} I_c \\ I_c \end{bmatrix} (5-36)
\]

and similarly the FLS parameters \( \alpha^s, \beta^s \) and \( \gamma^s \). The state space representation of the closed loop system can then be written as

\[
\begin{cases}
z(k+1) = E^c z(k) + F^c d(k) - F^c C^2 x^2(k) - F^c D^2 \gamma^s sat(\alpha^s G^s v(k) + \beta^s) \\
x^1(k+1) = A^1 x^1(k) + B^1 \gamma^s sat(\alpha^s G^s z(k) + \alpha^c H^c d(k) - \alpha^c H^c y(k) + \beta^c) \\
v(k+1) = E^s v(k) + F^s C^1 x^1(k) + F^s D^1 \gamma^s sat(\alpha^s G^s z(k) + \alpha^c H^c d(k) - \alpha^c H^c y(k) + \beta^c) \\
x^2(k+1) = A^2 x^2(k) + B^2 \gamma^s sat(\alpha^s G^s v(k) + \beta^s).
\end{cases} (5-37)
\]

Define the intermediate values

\[
t^1(k) = \begin{bmatrix} \begin{bmatrix} I_c \\ I_{x^1} \\ I_v \\ 0 \end{bmatrix} p(k) + \begin{bmatrix} \alpha^c G^c \\ \alpha^c H^c C^2 \\ \alpha^c H^c D^2 \gamma \end{bmatrix} w(k) \\ \begin{bmatrix} I_c \\ I_{x^1} \\ I_v \\ 0 \end{bmatrix} p(k) + \begin{bmatrix} \alpha^c G^c \\ \alpha^c H^c C^2 \\ \alpha^c H^c D^2 \gamma \end{bmatrix} w(k) \end{bmatrix} \end{bmatrix} \end{align*}
\]

where \( I_z \) is an identity matrix of size \( n \times n \) if \( z \) is of size \( n \times m \). Further, define

\[
t^2(k) = \begin{bmatrix} \begin{bmatrix} I_c \\ I_{x^1} \\ I_v \\ 0 \end{bmatrix} t^1(k) + \begin{bmatrix} \alpha^c G^c \\ \alpha^c H^c C^2 \\ \alpha^c H^c D^2 \gamma \end{bmatrix} w(k) \\ \begin{bmatrix} I_c \\ I_{x^1} \\ I_v \\ 0 \end{bmatrix} t^1(k) + \begin{bmatrix} \alpha^c G^c \\ \alpha^c H^c C^2 \\ \alpha^c H^c D^2 \gamma \end{bmatrix} w(k) \end{bmatrix} \end{align*}
\]

then the closed loop system in FIGURE 5-4 can be written as
NLqθ stability theory with application on Fuzzy Logic Systems

\[ p(k + 1) = \text{lin} \left( \begin{bmatrix} E^0 & 0 & 0 & -F^0C^2 & -F^0D^2 & 0 \\ 0 & A^1 & 0 & 0 & 0 & B^1\gamma^c \\ 0 & F^0C^1 & E^s & 0 & 0 & F^0D^1\gamma^c \\ 0 & 0 & 0 & A^2 & B^2 & 0 \end{bmatrix} t^2(k) + \begin{bmatrix} F^0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} w(k) \right) \] (5-40)

This is an NL_3 system

\[ p(k + 1) = \sigma_1(V_1\sigma_2(V_2\sigma_3(V_3p(k) + B_3w(k)) + B_2w(k)) + B_1w(k)) \] (5-41)

with

\[ V_1 = \begin{bmatrix} E^0 & 0 & 0 & -F^0C^2 & -F^0D^2 & 0 \\ 0 & A^1 & 0 & 0 & 0 & B^1\gamma^c \\ 0 & F^0C^1 & E^s & 0 & 0 & F^0D^1\gamma^c \\ 0 & 0 & 0 & A^2 & B^2 & 0 \end{bmatrix} \]

\[ V_2 = \begin{bmatrix} I_z & 0 \\ I_{x^1} & 0 \\ I_y & 0 \\ \gamma^c & 0 \\ \alpha^cG^c & 0 & 0 & -\alpha^cH^cC^2 & -\alpha^cH^cD^2\gamma^c \end{bmatrix} \] (5-42)

and

\[ V_3 = \begin{bmatrix} I_z & 0 & 0 & 0 \\ 0 & I_{x^1} & 0 & 0 \\ 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & I_{x^2} \\ 0 & 0 & \alpha^cG^c & 0 \end{bmatrix} \]

\[ B_1 = \begin{bmatrix} F^0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \]

\[ B_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \]

\[ B_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \alpha^cH^c \beta^c \end{bmatrix} \] (5-43)

Remark that in the case that the FL controller is chosen as a strictly proper system \((H^c = 0)\), the closed loop reduces to an NL_2 system. The stability of the autonomous and non-autonomous closed loop system can now be analysed with theorems 5.1 to 5.3. For the autonomous case this means that a sufficient proof that the hybrid system of FIGURE 5-4 is globally asymptotically stable, is given when a diagonal matrix \(\Delta\) can be found such that

\[ \Delta = \begin{bmatrix} 0 & V_2 & 0 \\ 0 & 0 & V_3 \\ V_1 & 0 & 0 \end{bmatrix} \]

\[ \Delta^{-1} \]

\[ \| \Delta^{-1} \|_2 = \| \beta(\Delta) \| < 1 \] (5-44)
or when matrices $P_i$ of full rank can be found such that
\[
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}
\begin{bmatrix}
0 & V_2 & 0 \\
0 & 0 & V_3 \\
V_1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}^{-1}
\prod_{i=1}^{3} \kappa(P_i) = \beta(P) < 1. \tag{5-45}
\]

In the non-autonomous case, the hybrid system is I/O stable if a diagonal matrix or matrices of full rank $P_i$ can be found such that
\[
\begin{bmatrix}
0 & V_2 & 0 \\
0 & 0 & V_3 \\
V_1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\Delta \\
\Delta^{-1}
\end{bmatrix}
\|_2 = \beta(\Delta) < 1 \tag{5-46}
\]
or
\[
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}
\begin{bmatrix}
0 & V_2 & 0 \\
0 & 0 & V_3 \\
V_1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}^{-1}
\prod_{i=1}^{3} \kappa(P_i) = \beta(P) < 1. \tag{5-47}
\]

When local stability is sufficient, the stability condition is reduced to a constrained minimization problem, where the condition number product
\[
\wp = \prod_{i=1}^{3} \kappa(P_i) \tag{5-48}
\]
is minimized, subject to the condition
\[
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}
\begin{bmatrix}
0 & V_2 & 0 \\
0 & 0 & V_3 \\
V_1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
P_2 & 0 & 0 \\
0 & P_3 & 0 \\
0 & 0 & P_1
\end{bmatrix}^{-1}
\|_2 < 1. \tag{5-49}
\]

**Remark:** In the case that the $V_i$ or $V'_i$ matrices are fixed, finding the $\Delta$ and $P_i$ matrices is a convex problem that can be solved in polynomial time with classic optimization tools that are available in standard mathematical programs (e.g. the constr() function in the Matlab mathematical framework). The analysis of a given FL system therefore becomes straightforward. This is not the case for the synthesis of stable FL controllers, since also the $V_i$ matrices need to be optimized. The synthesis of FL controllers is treated in the next section.
5.4 Learning SISO Fuzzy Systems with guaranteed NLq stability

5.4.1 Introduction

The introduction of stability constraints while optimizing the FL controller, is done by adding extra terms in the cost function. First consider the case of a MIMO system that is controlled by an FLC, as shown in FIGURE 5-5. The goal of the controller is to minimize the error $e(k)$ between the plant output $y(k) \in \mathbb{R}^n$ and a reference signal $d(k) \in \mathbb{R}^n$. Assume that the performance of the controller is measured using a Least Squares cost function

$$C_{LS}(d, y, \Theta) = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} (d_i(k) - y_i(k))^2$$  \hspace{1cm} (5-50)

with $N$ the number of measurement samples, taken in a certain time period $T > 0$, $d(k) = [d_1(k), d_2(k), ..., d_n(k)]^T$ and $y(k) = [y_1(k), y_2(k), ..., y_n(k)]^T$.

In order to force stability constraints, a term is added to the cost (5-50), which results in the new cost function

$$C_R(d, y, \Theta) = C_{LS}(d, y, \Theta) + \delta C_S(\Theta)$$  \hspace{1cm} (5-51)

where $\Theta$ denotes the FLC parameters (for the choice of the FL parameters see also section 4.3.2 on page 92). The cost $C_S(\Theta)$ equals

$$C_S(\Theta) = (\beta(P(\Theta)) - \varsigma)^2$$  \hspace{1cm} (5-52)

with $\beta(P(\Theta))$ defined in (5-14) on page 113 for autonomous systems, and

$$C_S(\Theta) = (\beta'(P(\Theta)) - \varsigma)^2$$  \hspace{1cm} (5-53)

with $\beta'(P)$ defined in (5-15) for non-autonomous systems. $\varsigma$ is a constant term that guarantees that $\beta(P)$ and $\beta'(P)$ are smaller than one, e.g. $\varsigma = 0.98$. The regularization term $\delta$ is
initialized at a low value, such that the Fuzzy controller is minimized for optimal performance. δ is then gradually enlarged during optimization, such that the optimization is pulled away from the best performing solution and the stability constraint

\[ C_S(\Theta) = 0 \]  

is satisfied, with \( C_S(\Theta) \) taken as in (5-52) or (5-53). The resulting control system should be optimal, with a guaranteed stability.

### 5.4.2 Illustrative example on a linear system

To fix the ideas, consider the SISO case where a linear plant is controlled using an FLC. The FLC is amplitude limited, such that the actuator output is guaranteed to remain below a given maximal value. The linear system is defined by the transfer function

\[ \text{Plant}(s) = \frac{1}{s(0.2s + 1)(0.5s + 1)}. \]  

(5-55)

The control is done in discrete time steps, with a sampling period \( \Delta T = 20 ms \). The state space representation of the plant, with \( x(k) \) the internal states of the plant, then equals

\[
\begin{aligned}
    x(k+1) &= \begin{bmatrix} 0.86 & -0.2 & 0 \\ 0.02 & 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0.02 \\ 0 \end{bmatrix} u(k) \\
    y(k) &= \begin{bmatrix} 0 & 0 & 10 \end{bmatrix} x(k).
\end{aligned}
\]  

(5-56)

The plant output is required to follow a trajectory

\[ d(t) = \frac{1}{2} \cos^3(t^2) \]  

(5-57)

that is initiated at time step \( t = 0 \) and where the output \( d(k+1) \) is then evaluated at discrete timesteps for five seconds. The actuator for the plant is known to be limited in amplitude, such that a classic linear controller could drive the actuator into saturation and classic stability theories don’t apply. For that reason the actuator is modelled using a FL controller. The controller is based on the strictly proper case of equation (5-25) on page 118 and is given by

\[
\begin{aligned}
    s(k+1) &= Es(k) + Fe(k) \\
    u(k) &= \text{FLS}(Gs(k)) = \sum_{j=1}^{r} K_{out}^{\mu_j} (GK_{in}s(k)) e_j^B
\end{aligned}
\]  

(5-58)
with \( s(k) \) the internal states of the controller, \( u(k) \) the controller output and \( e(k) \) the plant error, evaluated as \( e(k) = d(k) - y(k) \).

Since this is a SISO system, the number of rules, \( r \), equal the number of input FSs. For simplicity, the controller is set up with a proportional action, such that \( E = 0 \), \( F = 1 \) and \( G = 1 \). The FL parameters are the centres of the normal input FSs, the centres of the output

**FIGURE 5-6** [TOP] Initial, [MIDDLE] optimal and [BOTTOM] stable FL control of a linear system.
FS with a constant area, and the sensor and actuator gains $K_{in}$ and $K_{out}$ respectively. In order to guarantee normalization of the controller, the actuator gain is hard limited to the boundary [0, 1].

The initial Fuzzy parameters were chosen, based on a first guess, as

$$
\begin{align*}
K_{in} &= 0.7, \quad K_{out} = 0.5, \quad (5-59) \\
c^A &= [-1, -0.4, 0, 0.4, 1]^T \quad \text{and} \quad c^B = [-1, -0.7, 0, 0.7, 1]^T. \quad (5-60)
\end{align*}
$$

The first and last FSs of $c^A$ are open sets that remain fixed at plus and minus one, i.e. are not further optimized. The other parameters were optimized by minimizing the LS cost function (5-50) on page 124 where $n = 1$. After minimization, the resulting optimal FLC parameters are

$$
\begin{align*}
K_{in} &= 1.15, \quad K_{out} = 0.980, \quad (5-61) \\
c^A_{opt} &= [-1, -0.33, -0.01, 0.31, 1]^T \quad \text{and} \quad c^B_{opt} = [-1, -0.83, -0.01, 0.79, 1]^T. \quad (5-62)
\end{align*}
$$

The control performance is shown in FIGURE 5-6. The top row shows the initial FL control. The optimal control is given in the middle, while the stable control is at the bottom of the figure. On the left the input Fuzzy Sets are shown in the region $[-1.5, 1.5]$ and in the middle the output FS are shown for the same region. The input and output gains $K_{in}$ and $K_{out}$ are already included when calculating the FSs.

The tracking shows a good performance, even at higher frequencies on the right hand side of the tracking results figure. Yet, the step response in the beginning of the tracking indicates that the closed loop system is less damped. From the linear control theory point of view, this means that the poles are closer to the imaginary axis and the system is less stable.

The controller was further optimized, starting from the optimal FLC parameters and with guaranteed stability. During each iteration the stability matrices in (5-14) on page 113 are set up and the cost (5-53) on page 124 is minimized with the constraint $\zeta = 0.98$. The condition factors were chosen as $K_i = 1$, such that only semi-global stability is imposed. The resulting FLC parameters are

$$
\begin{align*}
K_{in} &= 0.695, \quad K_{out} = 1, \quad (5-63) \\
c^A_{stable} &= [-1, -0.53, 0, 0.32, 1]^T \quad \text{and} \quad c^B_{stable} = [-1, -0.82, 0, 0.82, 1]^T. \quad (5-64)
\end{align*}
$$

The performance shows that the system output in the first stage of the tracking is more damped, but also that the performance at higher frequencies is reduced.
It should be noted that, despite the good results in this example, there is an increased risk for the stability criterion to become far too conservative. It has been observed that it is hard to let systems comply with the stability criterion in the case of a large number of inputs \((n \gg 1)\) or with many input and output FSs, even if the control action \(K_{out}\) is set to practically zero.

### 5.4.3 Effects of conservatism on the condition number product

Even for FL control of simple linear systems, and when using the matrices of full rank \(P_i\), the full stability criterion (5-14) on page 113 can easily become too conservative for practical use. This can be partially solved by using the relaxed condition (5-16) such that \(\beta' < 1\), and maximize the attraction zone within which the stability condition applies, i.e. minimize \(\wp\) in equation (5-15).

<table>
<thead>
<tr>
<th>Order</th>
<th>Plant(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-st order</td>
<td>(Plant(s) = \frac{1}{(0.1s + 1)})</td>
</tr>
<tr>
<td>2-nd order</td>
<td>(Plant(s) = \frac{1}{(0.1s + 1)(0.5s + 1)})</td>
</tr>
<tr>
<td>3-rd order</td>
<td>(Plant(s) = \frac{1}{(0.1s + 1)(0.2s + 1)(0.5s + 1)})</td>
</tr>
<tr>
<td>4-th order (imaginary poles)</td>
<td>(Plant(s) = \frac{1}{(0.1s + 1)(0.166s + 1)(1.7s^2 + 1.25s + 1)})</td>
</tr>
</tbody>
</table>

**TABLE 5-1.** Linear system transfer functions

To show the effects of the model complexity on the condition number product \(\wp\), the simulations were repeated with increasing model order of the controlled system and using the control loop configuration of FIGURE 5-5. The different linear systems were chosen to have the transfer functions, given in TABLE 5-1. A constrained optimization was started where \(\wp = \prod_{i=1}^{q} \kappa(P_i)\) was minimized, with constraint \(\|P_{tot} V_{tot} P_{tot}^{-1}\|_2 = \beta' < 0.98\). The minimization was done using the constr() function within the Matlab framework.

The results of the simulations are shown in TABLE 5-2. It can be seen that the constraint is never reached, although \(\beta'\) is smaller than one in all cases, which means that the system is at least locally stable. The reason for not reaching the constraint \(\beta' < 0.98\) is possibly due to conservatism of the stability constraint.
Moreover, the condition number product increases very fast with the controlled system model order. Note that in none of the simulations global asymptotic stability of the closed loop system can be proven [205], since $\beta^2 \varphi > 1$ in all cases.

<table>
<thead>
<tr>
<th>order</th>
<th>$\beta'$</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-st order</td>
<td>0.9803</td>
<td>1.939</td>
</tr>
<tr>
<td>2-nd order</td>
<td>0.983</td>
<td>42.10</td>
</tr>
<tr>
<td>3-rd order</td>
<td>0.997</td>
<td>2468</td>
</tr>
<tr>
<td>4-th order (imaginary poles)</td>
<td>0.999</td>
<td>5519</td>
</tr>
</tbody>
</table>

**TABLE 5-2. Simulation Results**

When using the FL controller on a fifth order linear system, no stability could be proven, i.e. $\beta'$ remained larger than one. Yet, the manually initialized controller clearly showed to be stable. Even when the optimization routine was allowed to change the Fuzzy Sets in the FLC to obtain a better stability, it wasn’t possible to reach the criterion (5-14).

It must be stressed that all examples given make use of extremely low output gains in the FL controller, leading to very slow controller responses. This indicates that the stability constraint is still too conservative for complex systems, even if it may be of practical use for everyday simple systems.

### 5.5 NLqp theory

#### 5.5.1 Definition of NLqp systems

NL$_q^p$ theory can be seen as an extension to NL$_q$ theory and is useable with complex nonlinear systems. Further an example on multiple-Input Fuzzy Logic systems is given. The stability of complex nonlinear systems is in many publications studied with the use of a Lyapunov function. For an arbitrary system, this leaves the tedious task of finding a suitable Lyapunov function for each controller. In the sequel a stability criterion is given that can be used for NL$_q^p$ systems in general.

Assume an NL$_q$ system

$$ p_{k+1} = \Gamma_1(V_1 \Gamma_2(V_2 \ldots \Gamma_q(V_q p_k + B_q w_k)) \ldots + B_1 w_k) $$

(5-65)
which corresponds to a recurrent nonlinear network of the form

\[ p_{k+1} = \sigma_1(V_1 \sigma_2(V_2 \ldots \sigma_q(V_q p_k + B_q w_k)) \ldots + B_i w_k) \]  \hspace{1cm} (5-66)

**Definition 5.8** An \( \text{NL}_q^p \) system is the parallel concatenation of \( p \) \( \text{NL}_q \) subsystems, and is denoted as

\[ p_{k+1} = V \bigoplus_{j=1}^p \{ \text{NL}_q^j \} \]  \hspace{1cm} (5-67)

with \( \text{NL}_q^j = \text{NL}_q^j(p_k) \) the system (5-65) with elements \( \Gamma_i^j, V_i^j \) and \( B_i^j \).

The \( \text{NL}_q^p \) system relates to a recurrent network of the form

\[ p_{k+1} = V \prod_{j=1}^p \{ f_q^j(p_k, w_k) \} \]  \hspace{1cm} (5-68)

for scalar functions \( f_q^j \), or

\[ p_{k+1} = V \bigoplus_{j=1}^p \{ F_q^j(p_k, w_k) \} \]  \hspace{1cm} (5-69)

with

\[ F_q^j(p_k, w_k) = \sigma_1(V_i^j \sigma_2(V_2^j \ldots (\sigma_q(V_q^j p_k + B_q^j w_k)) \ldots B_i^j w_k) \]  \hspace{1cm} (5-70)

for MIMO systems.

**Lemma 5.3** The \( \text{NL}_q^p \) system (5-67) with exogeneous input can be written as

\[
\begin{bmatrix}
p_{k+1} \\
w_{k+1}
\end{bmatrix} = T \bigoplus_{j=1}^p \left\{ \Omega_i^j \left( T_i^j \Omega_q^j \left( T_q^j \ldots \Omega_q^j \left( T_q^j \left( p_k \right) \right) \ldots \right) \right) \right\}
\]  \hspace{1cm} (5-71)

with

\[ T_i^j = \begin{bmatrix} V_i^j & B_i^j \\ 0 & I \end{bmatrix}, T = \begin{bmatrix} V & 0 \\ 0 & X \end{bmatrix} \]  \hspace{1cm} (5-72)

and \( V_i^j \) and \( B_i^j \) given in (5-70). The \( X \) matrix describes the trajectory of the exogeneous input. For the \( \Omega_i^j \) matrices the following applies
Proof: Put (5-65) in equation (5-67). Equation (5-71) follows by straightforward calculation, using the matrices (5-72). The matrix \( \mathbf{X} \) describes the trajectory of the exogeneous input. Since \( \mathbf{w}_k \) is an external input, there is usually no need for the calculation of \( \mathbf{w}_{k+1} \), such that \( \mathbf{X} \) can be chosen as \( \mathbf{X} = \mathbf{0} \).

The matrices \( \Omega_i^{[j]} \) are calculated as

\[
\Omega_i^{[j]} = \begin{bmatrix} \Gamma_i^{[j]} & 0 \\ 0 & I \end{bmatrix}
\]

for which, by the definition of the 2-norm and using the definition of \( \Gamma_i^{[j]} \), applies that

\[
\|\Omega_i^{[j]}\|_2 \leq 1.
\]

\section{5.5.2 Global asymptotic stability of autonomous NLqp systems}

\textbf{Lemma 5.4} Consider a matrix \( \mathbf{A} \in \mathbb{R}^{n \times n} \) that is build as \( \mathbf{A} = \text{blockdiag}(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_m) \).

Then the following inequality applies:

\[
\|\mathbf{A}\|_2^m \geq \prod_{i=1}^{m} \|\mathbf{A}_i\|_2.
\]

\textbf{Proof:} See APPENDIX A.

\textbf{Remark:} In the proof it is shown that Lemma 5.4 also applies if two rows or two columns, containing the matrices \( \mathbf{A}_i \) and \( \mathbf{A}_j \) with \( 1 \leq i, j \leq m \), are randomly swapped.

\textbf{Assumption 5.6} The initial state space vectors \( \mathbf{p}_0 \) and exogeneous inputs \( \mathbf{w}_0 \) are normalized, such that it is possible to find a nonzero diagonal matrix \( \Delta \) with

\[
\|\Delta \mathbf{p}_0\|_2 \leq 1.
\]

It is always possible to find a matrix \( \Delta \) that complies with this assumption, by simply taking the inverse of the elements of \( \mathbf{p}_0 \) as the diagonal elements for \( \Delta \). If the initial states are already normalized, \( \Delta \) can be chosen as the identity matrix.
Definition 5.9 Consider the matrices $V_{i,j}^{[j]} \in \mathbb{R}^{n_{i,j} \times n_{i+1,j}}$ of size $n_{i,j} \times n_{i+1,j}$ with $n_{1,j} = n_{q+1,j} = n$ and the matrix $V \in \mathbb{R}^{n \times n}$ that correspond with the matrices of the NL$_p$ system (5-67) to (5-70). Then the square diagonal matrix $\Delta_{tot}$ is defined as

$$\Delta_{tot} = \text{diag}(\text{diag}(I_{n}, \Delta_{1}^{[1]}, \Delta_{2}^{[1]}, \ldots, \Delta_{q}^{[1]}, \Delta), \ldots, \text{diag}(I_{n}, \Delta_{1}^{[j]}, \Delta_{2}^{[j]}, \ldots, \Delta_{q}^{[j]}, \Delta), \ldots) \quad (5-77)$$

with $j = 1, 2, \ldots, p$ and where the $\Delta_{i}^{[j]}$ diagonal matrices of size $n_{i,j} \times n_{i,j}$ and $\Delta$ of size $n \times n$.

Definition 5.10 Consider the matrices $V_{i,j}^{[j]} \in \mathbb{R}^{n_{i,j} \times n_{i+1,j}}$ of size $n_{i,j} \times n_{i+1,j}$ with $n_{1,j} = n_{q+1,j} = n$ and the matrix $V \in \mathbb{R}^{n \times n}$ that correspond with the matrices of the NL$_p$ system (5-67) to (5-70). The square matrix $V_{tot}$ is defined as

$$V_{tot} = \begin{bmatrix} 0 & V^{[1]} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & V^{[2]} & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & V^{[3]} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & V^{[p]} \\ V & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (5-78)$$

where the $V^{[j]}$ are defined as

$$V^{[j]} = \text{blockdiag}(V_{1,j}^{[j]}, V_{2,j}^{[j]}, \ldots, V_{q,j}^{[j]}) \quad j = 1, 2, \ldots, p \quad (5-79)$$

Theorem 5.4 [Diagonal Scaling] Under Assumption 5.6, a sufficient condition for the global asymptotic stability of the autonomous NL$_p$ system (5-67) is that nonzero diagonal matrices $\Delta_{i}^{[j]}$ and $\Delta$ can be found, such that

$$\|\Delta_{tot} V_{tot} \Delta_{tot}^{-1}\|_2^{q+1} = \zeta < 1 \quad (5-80)$$

where $\Delta_{tot}$ and $V_{tot}$ are given in the definitions 5.9 and 5.10.

Proof: See APPENDIX A. □

In practice the condition (5-80) can be simplified to the condition
\[ \| \Delta_{tot} V_{tot} \Delta_{tot}^{-1} \|_2 < 1. \] (5-81)

**Definition 5.11** The square matrix \( P_{tot} \) is defined as

\[ P_{tot} = \text{blockdiag}(\text{blockdiag}(I_m, P_2^{[1]}, P_3^{[1]}, \ldots, P_q^{[1]}, P), \ldots, \text{blockdiag}(I_m, P_2^{[j]}, P_3^{[j]}, \ldots, P_q^{[j]}, P), \ldots, \text{blockdiag}(I_m, P_2^{[p]}, P_3^{[p]}, \ldots, P_q^{[p]}, P)) \] (5-82)

in which the positive definite matrices \( P \) and \( P^{[j]} \) of full rank have the same sizes as the \( \Delta \) and \( \Delta^{[j]} \) matrices of Definition 5.9 respectively.

**Assumption 5.7** It is possible to find a positive definite matrix \( P \) of full rank, such that

\[ \| P P_0 \|_2 \leq 1. \] (5-83)

Again, it is always possible to find such a matrix, e.g. by choosing \( P = \Delta \) while \( \Delta \) was already defined in Assumption 5.6.

**Theorem 5.5** [Condition Number Factor] Under Assumption 5.7, a sufficient condition for the global asymptotic stability of the autonomous NLqp system (5-67) is that nonzero positive definite matrices \( P \) and \( P^{[j]} \) can be found, such that

\[ \| P_{tot} V_{tot} P_{tot}^{-1} \|_2^{pq+1} \prod_{j=1}^{p} \prod_{i=2}^{q} [\kappa(P^{[j]}_i)] = \zeta < 1 \] (5-84)

where \( V_{tot} \) and \( P_{tot} \) are given in the definitions 5.10 and 5.11.

**Proof:** See APPENDIX A. \( \Box \)

### 5.5.3 I/O stability of NLqp systems with an exogeneous input

**Assumption 5.8** The initial state space vectors \( p_0 \) and exogeneous inputs \( w_0 \) of the NLqp system with exogeneous input (5-71) and (5-72) are normalized, such that it is possible to find a nonzero diagonal matrix \( \Delta \) with

\[ \| \begin{bmatrix} \Delta P_0 \\ \Delta w_0 \end{bmatrix} \|_2 \leq 1. \] (5-85)
It is always possible to find such a matrix $\Delta$ by taking the inverse of the elements of $p_0$ and $w_0$ as the diagonal elements for $\Delta$. If the initial states and the exogeneous inputs are already normalized, $\Delta$ can be chosen as the identity matrix.

**Definition 5.12** Consider the matrices $T^{[j]}_i \in \mathbb{R}^{n_{i,j} \times n_{i+1,j}}$ of size $n_{i,j} \times n_{i+1,j}$ with $n_{1,j} = n_{q+1,j} = n$ that correspond with the definition of the $NL^p_q$ system with exogeneous input (5-71) and (5-72). The square diagonal matrix $\Delta_{tot}$ is then defined as

$$\Delta_{tot} = \text{diag}[\text{diag}(I_{n_i}, \Delta^{[1]}_2, \Delta^{[1]}_3, ..., \Delta^{[1]}_q, \Delta), ..., \text{diag}(I_{n_i}, \Delta^{[j]}_2, \Delta^{[j]}_3, ..., \Delta^{[j]}_q, \Delta), ...]$$

with $j = 1, 2, ..., p$ and where the $\Delta^{[j]}_i$ are diagonal matrices of size $n_{i,j} \times n_{i,j}$ and $\Delta$ of size $n \times n$.

**Definition 5.13** Consider the matrices $T^{[j]}_i \in \mathbb{R}^{n_{i,j} \times n_{i+1,j}}$ of size $n_{i,j} \times n_{i+1,j}$ with $n_{1,j} = n_{q+1,j} = n$ that correspond with the definition of the $NL^p_q$ with exogeneous input (5-71) and (5-72). The diagonal matrix $T_{tot}$ is defined as

$$T_{tot} = \begin{bmatrix}
0 & T^{[1]} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & T^{[2]} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & T^{[3]} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & T^{[j]} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & T^{[j]} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & T^{[j]} \\
T & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}$$

(5-87)

where the $T^{[j]}$ are defined as

$$T^{[j]} = \text{blockdiag}(T^{[j]}_1, T^{[j]}_2, ..., T^{[j]}_q)$$

(5-88)

with $j = 1, 2, ..., p$.

**Theorem 5.6** [Diagonal Scaling] Under Assumption 5.8, a sufficient condition for the I/O stability of non-autonomous $NL^p_q$ systems is that nonzero diagonal matrices $\Delta^{[j]}_i$ and $\Delta$ can be found, such that

$$\|\Delta_{tot} T_{tot} \Delta_{tot}^{-1}\|_{p+1} = \zeta < 1$$

(5-89)
with $\Delta_{tot}$ and $T_{tot}$ given in the definitions 5.12 and 5.13.

**Proof:** See APPENDIX A. \hfill \Box

**Definition 5.14** The square matrix $P_{tot}$ is defined as

$$P_{tot} = \text{blockdiag}(\text{blockdiag}(I_n, P^{[1]}_1, P^{[1]}_3, ..., P^{[1]}_q, P), ...,$$

$$..., \text{blockdiag}(I_n, P^{[j]}_2, P^{[j]}_3, ..., P^{[j]}_q, P), ..., \text{blockdiag}(I_n, P^{[p]}_1, P^{[p]}_3, ..., P^{[p]}_q, P))$$  \hfill (5-90)

in which the positive definite matrices $P$ and $P^{[j]}$ of full rank have the same sizes as the $\Delta$ and $\Delta^{[j]}$ matrices of Definition 5.12.

**Assumption 5.9** It is possible to find a square positive definite matrix $P$ of full rank, such that

$$\left\| P \begin{bmatrix} p_0 \\ w_0 \end{bmatrix} \right\|_2 \leq 1 .$$  \hfill (5-91)

This assumption follows the reasoning in assumptions 5.7 and 5.8.

**Theorem 5.7** [Condition Number Factor] Under Assumption 5.9, a sufficient condition for the I/O stability of the non-autonomous $NL_{qp}$ system is that nonzero positive definite matrices $P$ and $P^{[j]}$ can be found, such that

$$\left\| P_{tot} T_{tot} P_{tot}^{-1} \right\|_{2^{q+1}} \prod_{j=1}^{p} \prod_{i=2}^{q} \kappa(P^{[j]}_i) < 1$$  \hfill (5-92)

where $T_{tot}$ and $P_{tot}$ are given in the definitions 5.13 and 5.14.

**Proof:** See APPENDIX A. \hfill \Box

**Condition relaxation**

Although it is not proven in this thesis, a relaxation should be possible on the stability criterions (5-84) and (5-92), similarly to the relaxation (5-15) and (5-16). A local I/O stability of the non-autonomous system can therefore be reached if the condition number product...
is minimized, subject to the condition

$$\|P_{tot}T_{tot}P_{tot}^{-1}\|_2 < 1$$

(5-94)

where $P_{tot}$ and $T_{tot}$ are given in the definitions 5.13 and 5.14. A similar relaxation should be possible for the autonomous stability criterion (5-84).

### 5.6 Representation of MIMO Fuzzy Logic Control systems within NLqp theory

#### 5.6.1 Formal representation of a MIMO Fuzzy Additive System

Assume a Mamdani fuzzy controller with trapezoidal and triangular Fuzzy Sets, such as the sets that were defined in detail in section 5.3.2 on page 115. We also adopt assumptions 5.1, 5.2 and 5.3 while Assumption 5.4 is formulated for MISO and MIMO systems as follows.

**Assumption 5.10** The rule base is full and non-redundant within the input domain. For a MIMO system this implies that the number of rules equals $r = R^p$ with $p$ the number of inputs and $R$ the number of fuzzy sets used for the inputs. Each combination of input sets is used exactly once in the rule base, while the output sets can be used more than once and not all output sets must be used.

This assumption suffers from the so called curse of dimensionality. A system with 4 inputs and 5 Fuzzy Sets per input, would need 625 rules. In order to prevent the curse of dimensionality, it is possible to relax the conditions given in Assumption 5.10 by limiting the rules within the domain of interest $\Psi \subseteq \mathbb{R}^p$ that covers the whole work space of the plant. Assumption 5.10 can still be considered true, as long as the system inputs remain in a subspace in which it can be guaranteed that

$$\sum_{i=1}^{R} \mu_i^A(x) = 1 \quad \forall x \in \Psi.$$  

(5-95)

This is the case e.g. for systems where some combinations of the inputs can impossibly be reached. Yet, it remains a good practise to monitor the condition (5-95) during control when stability must be guaranteed.
**Definition 5.15** A MISO Fuzzy Additive System is defined as a function \( f : \mathbb{R}^p \to \mathbb{R} \) that stores \( r \) rules of the form

\[
R_j: \text{If } (x_1 = A_{1,j}) \text{ and } (x_2 = A_{2,j}) \text{ and } \ldots \text{ and } (x_p = A_{p,j}) \text{ then } y = B_j. \quad (5-96)
\]

The antecedent is interpreted as a fuzzy set with a membership function

\[
\mu^A_i = \mu^A_{(x_1 = A_{1,i}) \text{ and } (x_2 = A_{2,i}) \text{ and } \ldots \text{ and } (x_p = A_{p,i})} \quad (5-97)
\]

and the consequent is calculated as

\[
FLS(x) = \text{Centroid} \left( \sum_{i=1}^{r} \mu^A_i B_j(y) \right) \quad (5-98)
\]

\[
= \text{Centroid} \left( \sum_{i=1}^{r} \left( \prod_{j=1}^{p} \mu^A_{(x_j = A_{j,i})} \right) B_j(y) \right)
\]

with \( x = (x_1, x_2, \ldots, x_p) \).

Following the lines of section 5.3.2 on page 115, it can be found that the calculation of the consequent reduces to

\[
FLS(x) = \sum_{i=1}^{r} \left( \prod_{j=1}^{p} \mu^A_{(x_j = A_{j,i})} \right) c_i^B \quad (5-99)
\]

with \( c_i^B \) the centres of the output sets.

**Theorem 5.8** The MISO Fuzzy Additive System (5-96) to (5-99) is an NL\(_2^p\) system.

**Proof:** See APPENDIX A.

**Assumption 5.11** Within the MIMO Fuzzy System, the consequents of the rules are not applied to the antecedents of other rules.
Lemma 5.5 Under Assumption 5.11 a MIMO Fuzzy System can be considered as a collection of MISO Fuzzy Systems. Without loss of generality, it is sufficient to discuss only MISO systems and apply the conclusions to MIMO systems.

Proof: See Lee [124] and Lin et al. [133].

5.6.2 Example

Consider a FL controller applied on a linear system, as shown in FIGURE 5-7

The state space representation of the linear system is

\[
\begin{align*}
    x_{k+1} &= Ax_k + Bu_k \\
    y_k &= Cx_k
\end{align*}
\]  

(5-100)

and the state space representation of the FLC can be written as

\[
\begin{align*}
    z_{k+1} &= E z_k + F (d_k - y_k) \\
    u_k &= FLC(G z_k + H(d_k - y_k))
\end{align*}
\]  

(5-101)

with

\[
E = 0, \quad F = 1, \quad G = \begin{bmatrix} 0 \\ -1/\Delta t \end{bmatrix} \quad \text{and} \quad H = \begin{bmatrix} 1 \\ 1/\Delta t \end{bmatrix}.
\]  

(5-102)

The sampling period of the discrete time system is denoted by $\Delta t$. The FLC is a MISO system with two inputs, and a rule base that contains $r$ rules

\[
R_j: \text{If } (x_1 = A_{1,j}) \text{ and } (x_2 = A_{2,j}) \text{ then } (y = B_j).
\]  

(5-103)
The FLC can be written in the form

\[
FLC(\phi) = (e^B)^T \bigotimes_{j=1}^{p} \frac{1}{2} \begin{bmatrix} I_r & I_z \end{bmatrix} \text{sat} \left( 2 \begin{bmatrix} S_{L,j}^T & 0 \\ -S_{R,j}^T & 0 \end{bmatrix} \phi_j + 1 + 2 \begin{bmatrix} (K_{L,j}^c - c^{A,j}) \otimes S_{L,j}^T \\ (K_{R,j}^c + c^{A,j}) \otimes S_{R,j}^T \end{bmatrix} \right)
\]

(5-104)

with \( \phi = [\phi_1 \ \phi_2]^T \). To determine the stability of the closed loop system of FIGURE 5-7, it must be written as an NLp^q. Therefore, define the state space vector

\[
\xi_k = [x_k^T \ z_k^T]^T
\]

(5-105)

and the exogeneous input

\[
w_k = [d_k^T \ 1]^T.
\]

(5-106)

Define the matrices

\[
\alpha^1 = 2 \begin{bmatrix} S_{L,1}^T & 0 \\ -S_{R,1}^T & 0 \end{bmatrix}, \quad \alpha^2 = 2 \begin{bmatrix} 0 & S_{L,2}^T \\ 0 & -S_{R,2}^T \end{bmatrix}
\]

(5-107)

and

\[
\beta^j = \begin{bmatrix} 1 + 2(K_{L,j}^c - c^{A,j}) \otimes S_{L,j}^T \\ 1 + 2(K_{R,j}^c + c^{A,j}) \otimes S_{R,j}^T \end{bmatrix}
\]

(5-108)

with \( j \in \{1, 2\} \). Define

\[
T = \begin{bmatrix} A & 0 & 0 & B(e^B)^T \\ -FC & E & F & 0 \end{bmatrix},
\]

(5-109)

\[
T_1^{[1]} = \text{blockdiag}(I_a, I_z, I_d, 1_{r \times 1}),
\]

(5-110)

and

\[
T_2^{[1]} = \text{blockdiag}(I_a, I_z, I_d, 1).
\]

(5-111)

\( I_a \) is an identical matrix of size \( n \times n \) if \( x \) has a size \( n \times 1 \). Further, define
and

\[
T_2^{[j]} = \begin{bmatrix}
0 \\
-\alpha^{-1}HC \\ \alpha^{-1}G \\ \alpha^{-1}H \\ \beta^{-1}
\end{bmatrix} \cdot \begin{bmatrix}
I/2 \\
I/2
\end{bmatrix}, \quad j = 2, 3.
\] (5-113)

with \(1_x\) a vector filled with ones and of size \(n \times 1\) if the vector \(x\) has a size \(n \times 1\). The closed loop system of FIGURE 5-7 can then be written as an NL system

\[
\begin{bmatrix}
x_{k+1} \\
z_{k+1}
\end{bmatrix} = T_2 \left( \bigotimes_{j=1}^{3} T_2^{[j]} \Omega_2^{[j]} \left( T_2^{[j]} \begin{bmatrix} \xi_k \\ w_k \end{bmatrix} \right) \right).
\] (5-114)

Remark that Theorem 5.7 can be used to examine the stability and the closed loop system is locally stable if matrices \(P\) and \(P^{[j]}\) of full rank can be found, such that

\[
\prod_{j=1}^{3} \prod_{i=2}^{2} [\kappa(P^{[j]})]
\] (5-115)

is minimized, subject to the constraints

\[
\|P_{tot}T_{tot}P_{tot}^{-1}\|_2 < 1
\] (5-116)

and

\[
\left\| P \begin{bmatrix} x_0 \\ z_0 \end{bmatrix} \right\|_2 \leq 1.
\] (5-117)

Here, \(P_{tot}\) and \(T_{tot}\) are defined as

\[
P_{tot} = \text{blockdiag}(I, P_2^{[1]}, P_1, I, P_2^{[2]}, P)
\] (5-118)
The stability criterion results in a constrained minimization problem of (5-115) with constraints (5-116) and (5-117). The minimization can be done using SQP programming techniques, e.g. with the "constr()" function within the Matlab mathematical toolbox. If the minimization of (5-115) results in a value that is smaller or equal to one, then the system is globally asymptotically stable.

\[
T_{tot} = \begin{bmatrix}
0 & T_1^{[1]} & 0 & 0 & 0 & 0 \\
0 & 0 & T_2^{[1]} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & T_1^{[2]} & 0 \\
0 & 0 & 0 & 0 & 0 & T_2^{[2]} \\
T & 0 & 0 & 0 & 0 & 0
\end{bmatrix}. \tag{5-119}
\]
Abstract: Although the Errors-In-Variables cost function was originally defined within the theory of econometry, it is better known in linear system identification theory. Based on the general identification scheme, it is clear that the cost function should also be useable for nonlinear transfer functions. For Wiener-Hammerstein systems, his has been done by Vandersteen [221]. This chapter introduces the EIV cost function for BB systems in general, but NN are used as an illustration of the theory.
6.1 Introduction

To date, much has been done to introduce different models and different learning schemes in the NN domain. However, more work is required on choosing the proper cost function which in turn determines completely the stochastic properties (noise sensitivity) of the NN. Currently, most learning algorithms for NN modeling are based on the Output Error approach, using a LS cost function, and special care must be taken when training the network with noisy input data, or when both inputs and outputs contain noise [233].

Sometimes avoiding input and output noise on the measurements is just not possible and other ways must be found to avoid biasing effects. Noise on the outputs has already been considered within the theory of Bayesian inference, introduced to NN by Green and MacKay [80] and Bishop [26]. This section proposes the use of the EIV cost function within BB modeling. The EIV cost function is already in use for identification of linear [53] [189] [196] and nonlinear [221] models and is defined in detail in section 2.6.2. We start with the introduction of the EIV cost function for BB and discuss the optimization scheme for optimizing the parameters of a Neural Network. Next, the need for a dedicated stopping algorithm and a proper initialization is discussed. Finally some examples are given to illustrate the improved NN performance with the EIV cost function when compared with the LS cost function. It must be stressed that EIV does not prevent biasing but merely reduces the biasing effects for nonlinear systems.

6.2 Problem statement

Although the following theory also applies for MIMO systems, SISO systems are discussed for simplicity of notations. Consider therefore a SISO linear or nonlinear system, as shown in FIGURE 6-1
For this system we seek a BB model $y_k^{[i]}(\theta) = f_{NN}(u_k^{[i]}, \theta)$ with $\theta$ the NN parameters, $u_k^{[i]}$ the measured inputs and $y_k^{[i]}(\theta)$ the estimated outputs of the measured outputs $y_k^{[i]}$. The network parameters are optimized with a set of $N \times M$ input-output samples, in which each measurement $k$, $k = 1, 2, \ldots, N$ is repeated $M$ times. The input and output matrices are denoted by $U$ and $Y$ and defined as

$$U \in \mathbb{R}^{N \times M}, U = \begin{bmatrix} u_1^{[1]} & u_1^{[2]} & \ldots & u_1^{[M]} \\ u_2^{[1]} & u_2^{[2]} & \ldots & u_2^{[M]} \\ \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & u_N^{[M]} \end{bmatrix}$$

and

$$Y \in \mathbb{R}^{N \times M}, Y = \begin{bmatrix} y_1^{[1]} & y_1^{[2]} & \ldots & y_1^{[M]} \\ y_2^{[1]} & y_2^{[2]} & \ldots & y_2^{[M]} \\ \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & y_N^{[M]} \end{bmatrix}.$$  (6-1)

The measurement pairs $(u_k^{[i]}, y_k^{[i]})$ will further be called the training set. For each measurement $k$ the sample variances of the inputs, $\hat{\sigma}_{u,k}^2$, and outputs, $\hat{\sigma}_{y,k}^2$, the sample covariance matrix $\hat{\sigma}_{uy,k}^2$ and the mean values $\hat{u}_k$ and $\hat{y}_k$ are determined as

$$\hat{u}_k = \frac{1}{M} \sum_{i=1}^{M} u_k^{[i]} \quad \hat{y}_k = \frac{1}{M} \sum_{i=1}^{M} y_k^{[i]}$$

$$\hat{\sigma}_{u,k}^2 = \frac{1}{M-1} \sum_{i=1}^{M} (u_k^{[i]} - \hat{u}_k)^2 \quad \hat{\sigma}_{y,k}^2 = \frac{1}{M-1} \sum_{i=1}^{M} (y_k^{[i]} - \hat{y}_k)^2$$

$$\hat{\sigma}_{uy,k}^2 = \frac{1}{M-1} \sum_{i=1}^{M} (u_k^{[i]} - \hat{u}_k)(y_k^{[i]} - \hat{y}_k)$$

where the $\hat{\cdot}$ denotes the estimated values. A special case is when $M = 1$ for which the variance cannot be calculated, but should be given a priori on the basis of the error interval of the measurement device, for example.
Assumption 6.1 The process noise and measurement noise sources $n_{p,k}^{[i]}$, $n_{m,u,k}^{[i]}$ and $n_{m,y,k}^{[i]}$ are considered to be stationary, independent, mutually uncorrelated and zero mean random variables with finite fourth order moments, and $\sigma_{u,y,k} = 0; \forall k$.

The means of the measured values (6-3) can then be written as

$$\hat{u}_k = u_k^* + n_{u,k}$$
$$\hat{y}_k = y_k^* + n_{y,k}$$

with

$$n_{u,k} = \frac{1}{M}\sum_{i=1}^{M}n_{u,k}^{[i]}$$
$$n_{y,k} = \frac{1}{M}\sum_{i=1}^{M}n_{y,k}^{[i]}$$

in which the $u_k^*$ and $y_k^*$ are the true but unknown values, $n_{u,k} = n_{m,u,k}$ and $n_{y,k} = n_{m,y,k} + n_{p,k}$ are the noise contributions, and $\sigma_{u,y,k}$ is the noise. Since by assumption the repeated measurements $i = 1, 2, ..., M$ are independent and the noise is stationary with finite variance, the sample mean and sample variances converge strongly to their true values [200], viz:

$$\text{a.s.lim}_{M \to \infty} (\hat{u}_k) = u_k^* n$$
$$\text{a.s.lim}_{M \to \infty} (\hat{y}_k) = y_k^*$$

and

$$\text{a.s.lim}_{M \to \infty} (\hat{\sigma}_{u,k}^2) = \sigma_{u,k}^2$$
$$\text{a.s.lim}_{M \to \infty} (\hat{\sigma}_{y,k}^2) = \sigma_{y,k}^2$$

in which a.s.lim stands for almost sure limit [142].

6.3 The Least-Squares cost function for NN

Most current NN programs make use of the Output Error (or: LS) cost function to fit the model to the data. In Van Gorp et al. [236] it is shown that using the LS cost function with noisy output samples asymptotically ($N \to \infty$) gives the true NN parameters, while optimizing the NN parameters with noisy input data in general leads to faulty NN parameters.
Assumption 6.2 The input is persistently exciting the system, such that the LS cost function

\[ C_{LS}(u_k^*, y_k^*) = \frac{1}{N} \sum_{k=1}^{N} (y_k^* - f_{NN}(u_k^*, \theta)) \]  

(6-10)

has a unique global minimum

\[ \theta^* = \arg\min_{\theta} \ (C_{LS}(u_k^*, y_k^*)) \]  

(6-11)

satisfying

\[ y_k^* = f_{NN}(u_k^*, \theta^*) \]  

(6-12)

when using noiseless measurement data pairs \((u_k^*, y_k^*)\) and to within any given precision and for any \(N\), including infinity.

This assumption is based on the fact that neural networks are universal approximators [73] [74] [97]. It must be well understood that \(u_k^*\) and \(y_k^*\) are not known (unless noiseless measurements are available), and that the cost \(C_{LS}(u_k^*, y_k^*)\) can’t be calculated in practise. The assumption only states that it is possible to find the unique true parameters \(\theta^*\) with the NN, given the true input and output measurements.

In practise it is possible that more global minima exist, e.g. by swapping two neurons within a network layer. If all the connected neurons in other layers are swapped with those neurons, the overall mapping of the NN remains the same [201]. This non-uniqueness problem can be solved by the use of parameters sets that lead to the same input-output relation, e.g. it is possible to demand that all neurons in a layer are sorted according to the weight and neurons in the other layers are swapped accordingly. This would lead to a unique solution of the NN parameters, without affecting the overall mapping.

For simplicity this thesis doesn’t deal with the problem where the global minimum is not unique, since this does not directly contribute to the idea that is presented in this chapter.

Knowing that the true parameters \(\theta^*\) can be found, we will now first focus on what happens if the NN parameters are optimized in the presence of noisy input measurements. Then the case of noisy input and output measurements will be considered.

Assumption 6.3 The LS cost function \(C_{LS}\) is a continuous function with respect to the parameters \(\theta\).

This assumption is needed in the proofs of the following theorems.

The Least-Squares cost function for NN
If the LS cost function is used with noisy inputs measurements and noiseless output measurements, the LS cost function for NN is written as

$$C_{LS}(\hat{u}_k, y_k^*) = \frac{1}{N} \sum_{k=1}^{N} (y_k^* - f_{NN}(\hat{u}_k, \theta))^2$$  \hspace{1cm} (6-13)

with $N$ the number of measurement samples, $y_k^*$ the true output samples and $\hat{u}_k$ the means of the noisy input samples. Replace the noisy input samples by their equivalents $\hat{u}_k = u_k^* + n_{u,k}$ (6-6) such that

$$C_{LS}(\hat{u}_k, y_k^*) = \frac{1}{N} \sum_{k=1}^{N} (y_k^* - f_{NN}(u_k^* + n_{u,k}, \theta))^2 .$$  \hspace{1cm} (6-14)

First consider the case where the noise contributions $n_{u,k}$ are small, so that the following first order Taylor series approximation of $f_{NN}$ can be made:

$$f_{NN}(u_k^* + n_{u,k}, \theta) \equiv f_{NN}(u_k^*, \theta) + n_{u,k} \frac{\partial (f_{NN}(u_k^*, \theta))}{\partial u_k} .$$  \hspace{1cm} (6-15)

**Definition 6.1** The linearised LS cost function is defined as

$$\overline{C}_{LS}(\hat{u}_k, y_k^*) = \frac{1}{N} \sum_{k=1}^{N} \left( y_k^* - f_{NN}(u_k^*, \theta) - n_{u,k} \frac{\partial (f_{NN}(u_k^*, \theta))}{\partial u_k} \right)^2 .$$  \hspace{1cm} (6-16)

**Theorem 6.1** If a NN is trained with noisy input measurements using the linearised LS cost function (6-16), the estimated NN parameters are in general inconsistent, or

$$\lim_{N \to \infty} \{ \text{arg min}_\theta (\overline{C}_{LS}(\hat{u}_k, y_k^*)) \} \neq \theta^*$$  \hspace{1cm} (6-17)

**Proof:** see APPENDIX A.

This theorem is a more formal representation of Theorem 7.1 on page 187, applied on NN and using a linearised form of the cost function. It can be seen that even with a simplified (linearised) transfer function, biasing remains. In section 6.4 it is proven that the use of the EIV cost function avoids this bias in the linearised case.

The previous theorem was formulated for the case of input noise only. The remaining question is if Theorem 6.1 still applies if both the inputs and outputs contain noise when optimizing the
NN parameters. For the case of noisy input and output measurements, the LS cost function is written as

$$C_{LS}(\hat{u}_k, \hat{y}_k) = \frac{1}{N} \sum_{k=1}^{N} (\hat{y}_k - f_{NN}(\hat{u}_k, \theta))^2$$  \hspace{1cm} (6-18)

with $$\hat{y}_k = y_k^* + n_{y,k}$$ the noisy output samples, and $$\hat{u}_k, \theta$$ and $$N$$ defined as in (6-10). The linearized cost function becomes

$$C_{LS}(\hat{u}_k, \hat{y}_k) = \frac{1}{N} \sum_{k=1}^{N} \left( \hat{y}_k - f_{NN}(u_k^*, \theta) - n_{u,k} \frac{\partial (f_{NN}(u_k^*, \theta))}{\partial u_k} \right)^2.$$  \hspace{1cm} (6-19)

The case for noisy inputs and outputs is then treated in the following theorem.

Theorem 6.2 If a NN is trained with noisy input and output measurements using the linearized cost function (6-19), the estimated NN parameters are in general inconsistent, or

$$\text{a.s. lim}_{N \to \infty} \arg \min_{\theta} \left( \bar{C}_{LS}(\hat{u}_k, \hat{y}_k) \right) \neq \theta^*$$ \hspace{1cm} (6-20)

Proof: see APPENDIX A. \hfill \square

6.4 The Errors-In-Variables cost function for NN

The extra knowledge of the variances on the inputs and the outputs allows the use of the EIV cost function

$$C_{EIV}(\hat{u}_k, \hat{y}_k) = \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{(\hat{y}_k - f_{NN}(u_k, \theta))^2}{\sigma_{y,k}^2} + \frac{(\hat{u}_k - u_k)^2}{\sigma_{u,k}^2} \right].$$  \hspace{1cm} (6-21)

$$u_k$$ and $$y_k$$ parametrize the true but unknown input and output values which must be estimated. It will first be shown that the EIV estimates are consistent with the linearised NN, such that only higher order noise terms contribute to the inconsistency (biasing effects) of the NN. Next it is shown that for Gaussian errors the strong consistency for the linearised NN remains if the samples variances $$\hat{\sigma}_{u,k}^2$$ and $$\hat{\sigma}_{y,k}^2$$ are used instead of the true variances, where at least six repeated measurements are made.

Assumption 6.4 The input is persistently exciting the system, such that the EIV cost function
The EIV cost function for nonlinear systems

\[
C_{EIV}(u_k^*, y_k^*) = \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{(y_k^* - f_{NN}(u_k^*, \theta))^2}{\sigma_{\gamma,k}^2} + \frac{(u_k^* - u_k)^2}{\sigma_{u,k}^2} \right] \tag{6-22}
\]

has a unique global minimum

\[
(\theta^*, u_k^*) = \arg \min_{\theta, u_k} (C_{EIV}(u_k^*, y_k^*)) \tag{6-23}
\]

satisfying

\[
y_k^* = f_{NN}(u_k^*, \theta^*) \tag{6-24}
\]

when using noiseless measurement data pairs \((u_k^*, y_k^*)\) and to within any given precision and for any \(N\), including infinity.

**Assumption 6.5** The EIV cost function \(C_{EIV}\) is a continuous function with respect to the parameters \(\theta\).

**Definition 6.2** The linearised form of the EIV cost function (6-21) is denoted as

\[
\bar{C}_{EIV}(\hat{u}_k, \hat{y}_k) = \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{1}{\sigma_{\gamma,k}^2} \left( \hat{y}_k - f_{NN}(u_k^*, \theta) - \varepsilon_{u,k} \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} \right)^2 + \frac{1}{\sigma_{u,k}^2} (\hat{u}_k - u_k)^2 \right] \tag{6-25}
\]

in which the error in the estimated input values is defined as

\[
\varepsilon_{u,k} = u_k - u_k^* \tag{6-26}
\]

and the NN is replaced by its first order approximation

\[
f_{NN}(u_k, \theta) \approx f_{NN}(u_k^*, \theta) + \varepsilon_{u,k} \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*}. \tag{6-27}
\]

**Remark:** Note that a clear distinction exists between the noise on the measurements, \(n_{u,k} = \hat{u}_k - u_k^*\), and the model errors, \(\varepsilon_{u,k} = u_k - u_k^*\).
Lemma 6.1 Under assumptions 6.1, 6.4 and 6.5, and when a NN is trained with noisy input and output measurements using the linearised EIV cost function (6-25), the minimizer of the expected value of the cost function, equals the true NN parameters \( \theta^* \), or

\[
\theta^* = \arg\min_{\hat{\theta}} \{ E\{ \tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k) \} \} \tag{6-28}
\]

**Proof:** see APPENDIX A. \( \square \)

Theorem 6.3 Under assumptions 6.1, 6.4 and 6.5, and if a NN is trained with noisy input and output measurements using the linearised EIV cost function (6-25), the estimated NN parameters \( \hat{\theta} \) converge strongly to the true NN parameters \( \theta^* \), or

\[
\text{a.s.lim}_{N \to \infty} \{ \arg\min_{\theta} (\tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k)) \} = \theta^* \tag{6-29}
\]

**Proof:** see APPENDIX A. \( \square \)

In practice it is not possible to use the linearised EIV cost function because it needs the true input values \( u_k^* \) which, of course, are not known. Consequently the EIV cost function (6-21) must be used. The effects on the NN parameters is described in the following theorem.

**Definition 6.3** The *bias* of the NN parameters is defined as

\[
\beta_{\theta} = E\{ \hat{\theta} - \theta^* \} \tag{6-30}
\]

with

\[
\hat{\theta} = \arg\min_{\theta} (\hat{C}(\hat{u}_k, \hat{y}_k)) \tag{6-31}
\]

and where \( \hat{C}(\hat{u}_k, \hat{y}_k) \) can be the EIV or the LS cost function.

**Definition 6.4** The *variances* of the NN parameters are defined as the diagonal elements of the covariance matrix

\[
\text{cov}(\theta) = E\{ (\hat{\theta} - \theta^*) (\hat{\theta} - \theta^*)^T \} \tag{6-32}
\]

with \( \hat{\theta} \) defined as in Definition 6.3.
**Definition 6.5** The *mean square error* (MSE) of a BB mapping is denoted as

\[
MSE = \sqrt{\frac{1}{N} \sum_{k=1}^{N} \|f_{BB}(u_k, \hat{\theta}) - f_{BB}(u_k, \theta^*)\|^2_2}
\]  

(6-33)

with \( \hat{\theta} \) defined as in Definition 6.3 and \( N \) the total number of measurements.

**Theorem 6.4** Under assumptions 6.1 and 6.2, and if a NN is optimized with noisy input and output measurements and for sufficiently large SNR (\( \text{SNR} > 0 \)), the NN parameters based on the EIV cost function (6-21) have a smaller bias (see Definition 7.5 on page 182) than the NN parameters based on the LS cost function.

**Proof:** When training with noisy inputs Theorem 6.1 shows the inconsistency of the linearised LS estimator, while Theorem 6.3 proves the consistency of the linearised EIV estimator. If larger noise levels are used, higher order terms come into effect and both estimators become inconsistent. For the EIV, however, only higher order moments of the noise contributes to the inconsistency, which makes EIV more robust against input noise. □

We can conclude that the generally used Least Squares cost function leads to biased NN parameters when training with noisy input measurements. Therefore it is better to learn the NN parameters with the EIV cost function, thus reducing the bias on the parameters. Note that the bias doesn’t completely disappear.

Moreover, in [176] it is shown that for the linearised case and for sufficiently large SNR, the EIV cost function approaches the Cramér-Rao bound [164] asymptotically when modeling with noisy inputs. This is not the case for the LS cost function.

As a result the EIV qualifies an efficient estimator for the linearised NN. The variance on the NN parameters will be less when using the EIV cost function than when using the LS cost function. Also the MSE of the resulting NN is lower compared with the MSE of the NN that are estimated with the LS cost function.

The examples given below, show that the advantage of the EIV is mainly in improving mapping of the details of a transfer function, and a smaller RMS error when the NN model is compared with the true model.
6.5 Condition relaxation on the variances

For large data sets with many repeated measurements ($M$ large) the variances can be well estimated and equation (6-8) applies. In practise, the true variances of the input and output measurements are usually not known, in which case the sample variances (6-3) must be used. If the noise is Gaussian distributed, we already know that even for small data sets ($M \geq 6$) [189]

$$E\{\hat{C}_{EIV}\} = \frac{M-1}{M-3} E\{C_{EIV}\}$$  \hspace{1cm} (6-34)$$

with

$$\hat{C}_{EIV} = \frac{M}{2N} \sum_{k=1}^{N} \left[ \hat{y}_k - f_{NN}(u_k, \theta) \right]^T \begin{bmatrix} \hat{\sigma}^2_{y,k} & \hat{\sigma}_{yu,k} & \hat{\sigma}^2_{uy,k} \\ \hat{\sigma}_{uy,k} & \hat{\sigma}^2_{u,k} \\ \hat{\sigma}_{u,k} \end{bmatrix}^{-1} \begin{bmatrix} \hat{y}_k - f_{NN}(u_k, \theta) \\ \hat{u}_k - u_k \end{bmatrix}.$$  \hspace{1cm} (6-35)$$

Thus, even for a small number of repeated measurements it is possible to replace the true variances and means by the sample variances and means. This has no effect on the minimization of the cost function with respect to $\theta$. Note that the sample covariances $\hat{\sigma}_{yu,k}$ and $\hat{\sigma}_{uy,k}$ must be included although it is assumed that the input and output errors are uncorrelated.

Assume that all noise terms are uncorrelated. Then for a very large number of repeated measurements ($M \to \infty$), or in the case that the variances are exactly known, the variance matrix only contains diagonal elements. The estimated EIV cost function (6-35) can then be simplified to

$$\hat{C}_{EIV} = \frac{M}{2N} \sum_{k=1}^{N} \frac{(\hat{y}_k - f_{NN}(u_k; \theta))^2}{\hat{\sigma}^2_{y,k}} + \frac{(\hat{u}_k - u_k)^2}{\hat{\sigma}^2_{u,k}}.$$  \hspace{1cm} (6-36)$$

In the particular case that $M = 1$ and the user can give good estimates for the variances, the EIV estimator still proves to be useful. The extra knowledge on the variances allows for a better learning of the NN parameters, and outliers due to noise on the inputs will be moved towards the NN mapping.
6.6 EIV cost function learning algorithms for large data sets (M large)

It is observed that the EIV cost function, used on NN, leads to an increased risk of being trapped in a local minimum during optimization. This was also observed for other nonlinear systems in [221]. The fact that estimated values of the input measurements are used for modeling the NN, could be a possible reason for the increase in local minima. Each time one of the estimated input values is updated, the NN parameters must again be optimized to a local minimum. Initially, when large updates are used for the estimated input values, this could cause the cost function to hop from one local minimum to the other for the NN, thus seemingly increasing the number of local minima.

Because of the increased risk for local minima, it is advisable to use the output parameters from the LS method as the starting values for the EIV cost function. This increases the robustness, and decreases the number of needed learning steps. EIV can be regarded as a postprocessing tool to improve the accuracy of the NN parameters after OE learning. The EIV cost function can actually be used on a "no cure, no pay" basis.

In this section the case of a large number of repeated measurements is considered (M large) and the noise is considered to be fully uncorrelated. The covariance matrix is then reduced to a diagonal matrix, and the definition (6-36) applies for the EIV cost function.

In the sequel two learning methods are discussed: gradient methods for large number of different measurement samples (N large), and the more robust Levenberg-Marquardt method for smaller number of samples (N small). The decision line for calling N large or small depends on the available amount of internal memory and processing power.

6.6.1 Gradient methods (Backpropagation)

Define the errors

\[ e_{f,k} = \hat{y}_k - f_{NN}(u_k, \Theta) \]  (6-37)

and

\[ e_{u,k} = \hat{u}_k - u_k \]  (6-38)

such that \( e_f = [e_{f,k}] \) and \( e_u = [e_{u,k}] \) are column vectors with length N. Define the error vector \( e \) as

\[ e = \begin{bmatrix} e_f^T & e_u^T \end{bmatrix}^T \]  (6-39)
and the vector of parameters $p$

$$p = \left[ \theta^T u^T \right]^T.$$  

(6-40)

For backpropagation the parameter update vector

$$\Delta p = \left[ \Delta \theta^T \Delta u^T \right]^T$$

(6-41)

of the simplified EIV cost function (6-36) is calculated using

$$\Delta p = -\eta J^T \Phi e$$

(6-42)

in which $\eta$ is a small arbitrary positive value, called the learning rate. Methods exist to make $\eta$ adaptive or to include a momentum term [71], but these are discussed in detail in section 2.7.2. The $\Phi$ matrix is a diagonal matrix of size $2N \times 2N$ based on the variances, or

$$\Phi = \begin{bmatrix} \Phi_y & 0 \\ 0 & \Phi_u \end{bmatrix}$$

(6-43)

with

$$\Phi_y = \begin{bmatrix} 1/\hat{\sigma}_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\hat{\sigma}_N^2 \end{bmatrix} \quad \text{and} \quad \Phi_u = \begin{bmatrix} 1/\hat{\sigma}_{u,1}^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\hat{\sigma}_{u,N}^2 \end{bmatrix}.$$  

(6-44)

In (6-43) $J$ is the Jacobian matrix, defined as $J_{ij} = \partial e_i / \partial p_j$ which will be denoted as $J = \partial e / \partial p$. The Jacobian equals

$$J = \begin{bmatrix} A & B \\ 0_{N \times P} & I_N \end{bmatrix}.$$  

(6-45)

with $P$ the number of NN parameters in $\theta$. $A \in \mathbb{R}^{N \times P}$ and $B \in \mathbb{R}^{N \times N}$ are defined as

$$A = \frac{\partial f_{NN}(\theta, u)}{\partial \theta} \quad \text{and} \quad B = \frac{\partial f_{NN}(\theta, u)}{\partial u}.$$  

(6-46)

The derivation of the Jacobian matrix poses a special problem. While for the OE method only $A$ must be calculated, EIV also demands for the calculation of the $B$ matrix, which has a size of $N \times N$. Making use of the property that $B$ is a sparse matrix, it is possible to simplify the parameter updates, such that
The EIV cost function for nonlinear systems

\[
\Delta p = \eta \begin{bmatrix} A^T \Phi_y e_f \\ B^T \Phi_y e_f + \Phi_u e_u \end{bmatrix}
\]  \hspace{1cm} (6-47)

and finally

\[
\begin{align*}
\Delta \theta_j &= \frac{\eta M}{2N} \sum_{k=1}^N \frac{e_{f,k} \partial f_{NN}(u_k, \theta) \partial \theta_j}{\hat{\sigma}_{y,k}^2} \\
\Delta u_k &= \frac{\eta M}{2} \left[ \frac{e_{f,k} \partial f_{NN}(u_k, \theta)}{\hat{\sigma}_{u,k}^2} + \frac{e_{u,k}}{\hat{\sigma}_{u,k}^2} \right].
\end{align*}
\]  \hspace{1cm} (6-48)

It can be seen that it is no longer necessary to store the \(B\) matrix in memory, although it still needs to be calculated. This indicates that the EIV method needs more calculations than the OE method and that the method is slower, with the added advantage that the method produces a more correct NN model.

### 6.6.2 Levenberg-Marquardt

Here, the optimization step for the simplified cost function (6-36) is defined as

\[
\Delta p = -(J^T \Phi J + \mu I)^{-1} (J^T \Phi e)
\]  \hspace{1cm} (6-49)

with \(J\), \(\Phi\) and \(e\) defined as in the previous paragraph. Note that the terms containing second order derivatives of the NN function \(f_{NN}(u_k, \theta)\) with respect to the parameters are negligible compared to the first order terms.

In contrast to gradient methods, Levenberg-Marquardt optimization demands that the full Jacobian is stored in memory. Moreover a very large matrix must be inverted in order to calculate the optimization step. It is possible to reduce the number of calculations, based on the knowledge of \(J\), as given in (6-45). The calculation of the parameter update vector becomes

\[
\Delta p = -\eta \begin{bmatrix} A^T \Phi_y A & A^T \Phi_y B \\ B^T \Phi_y A & B^T \Phi_y B + \Phi_u \end{bmatrix}^{-1} \begin{bmatrix} A^T \Phi_y e_f \\ B^T \Phi_y e_f + \Phi_u e_u \end{bmatrix}
\]  \hspace{1cm} (6-50)

and now needs little less memory during the calculation. In practise LM proves to be very robust, and convergence is reached faster than with gradient methods. The only restriction is the limited amount of memory.
6.7 EIV cost function learning algorithms for small data sets (M small)

In this section the case of a small number of repeated measurements is considered ($M$ small) such that the full covariance matrix must be taken into account for the calculation of the EIV cost function.

6.7.1 Gradient methods (Backpropagation)

Define the column vector errors $e_f = [e_{f,1}, ..., e_{f,N}]$ and $e_u = [e_{u,1}, ..., e_{u,N}]$ as in section 6.6.1 on page 154. Define the error vector $e \in \mathbb{R}^{2N \times 1}$ as

$$e = \begin{bmatrix} e_{f,1}, e_{u,1}, e_{f,2}, e_{u,2}, ..., e_{f,N}, e_{u,N} \end{bmatrix}^T.$$

(6-51)

The covariance matrices are symmetric positive definite matrices that can be decomposed using singular value decomposition as

$$\begin{bmatrix} \hat{\sigma}^2_{y,k} & \hat{\sigma}_{yu,k} \\ \hat{\sigma}_{uy,k} & \hat{\sigma}^2_{u,k} \end{bmatrix} = U_k \Sigma_k U_k^T.$$

(6-52)

Now define the inverse matrices $C_k = U_k \Sigma_k^{-1/2} U_k^T$ such that

$$C_k C_k = C_k^T C_k = \begin{bmatrix} \hat{\sigma}^2_{y,k} & \hat{\sigma}_{yu,k} \\ \hat{\sigma}_{uy,k} & \hat{\sigma}^2_{u,k} \end{bmatrix}^{-1} = \begin{bmatrix} a_k & \beta_k \\ \beta_k & \gamma_k \end{bmatrix}$$

(6-53)

and define the matrix $C = \text{blockdiag}\{ C_1, C_2, ..., C_N \}$. The full EIV cost function (6-35) on page 153 can be written as

$$\hat{C}_{EIV} = \frac{M}{2N} e^T C C e = \frac{M}{2N} e^T C^T C e.$$

(6-54)

Define the vector of parameters $p$ and the parameter update vector $\Delta p$ as

$$p = \begin{bmatrix} \theta^T \hspace{2pt} u^T \end{bmatrix}^T \text{ and } \Delta p = \begin{bmatrix} \Delta \theta^T \hspace{2pt} \Delta u^T \end{bmatrix}^T.$$

(6-55)

Recall the matrices, given by (6-46), as

$$A = \frac{\partial f_{NN}(\theta, u)}{\partial \theta} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots & \ddots \end{bmatrix}$$

(6-56)
The EIV cost function for nonlinear systems

and

\[ B = \frac{\partial f_{NN}(\theta, u)}{\partial u} = \begin{bmatrix} b_{1,1} & b_{1,2} & \ldots \\ b_{2,1} & b_{2,2} \\ \vdots & \vdots \end{bmatrix}. \]  

(6-57)

then the Jacobian \( J = \frac{\partial e}{\partial \theta} \in \mathbb{R}^{2N \times (N + P)} \) is defined as

\[ J = - \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,N} & b_{1,1} & b_{1,2} & \ldots & b_{1,N} \\ 0 & 0 & \ldots & 0 & 1 & 0 & 0 & \ldots & 0 \\ a_{2,1} & a_{2,2} & \ldots & a_{2,N} & b_{2,1} & b_{2,2} & \ldots & b_{2,N} \\ 0 & 0 & \ldots & 0 & 0 & 1 & 0 & \ldots & 0 \\ a_{3,1} & a_{3,2} & \ldots & a_{3,N} & b_{3,1} & b_{3,2} & \ldots & b_{3,N} \\ 0 & 0 & \ldots & 0 & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{N,1} & a_{N,2} & \ldots & a_{N,N} & b_{N,1} & b_{N,2} & \ldots & b_{N,N} \\ 0 & 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 1 \end{bmatrix}. \]  

(6-58)

and the backpropagation parameter update vector of the full EIV cost function (6-54) is calculated as

\[ \Delta p = -\eta J^T C^T e \]  

(6-59)

in which \( \eta \) is the learning rate. Making use of the property that \( J \) is a sparse matrix, the individual parameter updates can be calculated as

\[
\begin{cases}
\Delta \theta_j = \eta \frac{M}{2N} \sum_{k=1}^{N} \left[ (\alpha_k e_{f,k} + \beta_k e_{u,k}) \frac{\partial f_{NN}(u_k, \theta)}{\partial \theta_j} \right] \\
\Delta u_k = \eta \left[ (\alpha_k e_{f,k} + \beta_k e_{u,k}) \frac{\partial f_{NN}(u_k, \theta)}{\partial u_k} + \beta_k e_{f,k} + \gamma_k e_{u,k} \right].
\end{cases}
\]  

(6-60)

6.7.2 Levenberg-Marquardt

The optimization step for the full cost function (6-54) is calculated as

\[ \Delta p = -(J^T C^T C J + \mu I)^{-1} (J^T C^T Ce) \]  

(6-61)

In this case, the full Jacobian is used for the inversion of the matrix and no straightforward simplification is possible. The only restriction, therefore, is the limited amount of memory.
6.8 The need for early stopping

EIV identification shapes both inputs and outputs according to the confidence level, indicated by the variances $\sigma_u^2$ and $\sigma_y^2$. As a result, EIV is very prone to overfitting. In the case of large variances, the measurements are typically grouped together related to the horizontal axis. In between the measurement groups, gaps arise. This potentially leads to the case of sparse data and an inaccurate mapping.

The effects of this overfitting are shown in FIGURE 6-2. In the figure, an arbitrary nonlinear function

$$y = \tanh(10u + 4) - \tanh(10u + 3) + \tanh(10u - 4) - \tanh(10u - 3) \quad (6-62)$$

with $u \in [-1, 1]$ is chosen and sampled 100 times. Gaussian, zero mean noise is added to both inputs and outputs with $\sigma_{u,k}^2 = \sigma_u^2 = 13\text{dB}$ and $\sigma_{y,k}^2 = \sigma_y^2 = 13\text{dB}$. The measurements were not repeated ($M = 1$). The samples are shown as crosses on the figure. The circles are the estimated $u_k$, after mapping the samples using a NN with one hidden layer of 10 tan-sigmoid perceptrons.

The figure shows how the estimates are drawn towards the approximated curve: must $u_k$ lie close to the EIV approximation of the original function. However, no $u_k$ points remain in e.g. the region $u \in [0.081, 0.217]$ (enlarged section) and the NN approximation of the curve shows an overfitting in this region. Within the uncertainty bounds of the $u_k^{[l]}$ values, the
measurements in this region are shifted away over the horizontal axis, such that the neural network is allowed to take any arbitrary form in the resulting gap.

In the case of EIV, this overfitting can successfully be prevented by the use of a dedicated early-stopping algorithm. This is done with a validation set which is built in the same way as the training set and consists of \( N_v \) measurements with:

\[
\begin{align*}
  u_l &= u_l^* + n_{u,l} & l = 1 \ldots N_v \times M_v \\
  y_l &= y_l^* + n_{y,l}
\end{align*}
\]

(6-63)

in which the index \( \cdot_v \) denotes the validation set. Note that repeated measurements \( (M_v > 1) \) can be used to calculate the variances on the inputs and outputs, but aren’t treated differently from other measurements when evaluating the early stopping criterion.

The easiest way to implement a stopping criterion is by monitoring the LS cost function

\[
C = \sum_{l=1}^{N_v \times M_v} (y_l - f_{NN}(u_l, \Theta))^2
\]

(6-64)

applied on the validation set and stop learning whenever this cost function starts increasing again. In the case of EIV, the cost function (6-35) should be used with fixed NN parameters \( \Theta \), in which case the true input values \( u_l^* \) must be estimated prior to the evaluation of the cost function. This calls for an extra learning sequence each time validation is performed. It is evident that this results in a very slow learning process since during each loop of the optimization routine, a second optimization must be performed.

It is possible to normalize (6-64) using a linearised form of the NN function, similar to the approximation used in the proof of Lemma 6.1. The early stopping criterion \( C_{ES} \) then takes the form

\[
C_{ES} = \sum_{l=1}^{N_v \times M_v} \left[ \frac{1}{\sigma_{y,l}^2 + \sigma_{u,l}^2 \left( \frac{\partial f_{NN}(u_l^*, \Theta)}{\partial u_l^*} \right)^2} \right] \left( y_l - f_{NN}(u_l, \Theta) \right)^2
\]

(6-65)

**Remark:** Note that the early stopping criterion is applied on the measured values only, without estimating the true \( u_l^* \) values.
Assumption 6.6 Equation (6-65) makes uses of the derivative of the NN function for the true $u_k^*$ values. In practise these values are unknown. Therefore, the following approximation is made:

$$\frac{\partial f_{NN}(u_l^*, \theta)}{\partial u_l^*} \approx \frac{\partial f_{NN}(u_l, \theta)}{\partial u_l}.$$  

(6-66)

This means that within the uncertainty bounds of the measurements, the NN is expected to have about the same gradient, i.e. the NN mapping should be smooth.

If this assumption cannot be met, the only solution is the use of the OE cost function for early stopping. If the assumption is met, the normalized cost function used for early stopping, is defined as

$$C_{ES} = \sum_{l=1}^{N_v \times M_v} \frac{(y_l - f_{NN}(u_l, \theta))^2}{\hat{\sigma}_{y,l}^2 + \left(\frac{\partial f_{NN}(u_l, \theta)}{\partial u_l}\right)^2 \hat{\sigma}_{u,l}^2 - 2 \hat{\sigma}_{u,y,l} \frac{\partial f_{NN}(u_l, \theta)}{\partial u_l} \frac{\partial f_{NN}(u_l, \theta)}{\partial u_l}}$$

(6-67)

in which the parameters $\theta$ have been determined by one of the previously given optimization routines.

Here also, it can be assumed that all noise terms are uncorrelated. Then for a very large number of repeated measurements ($M \to \infty$), or in the case that the variances are exactly known, the variance matrix only contains diagonal elements. The normalized cost function for early stopping then reduces to

$$C_{ES} = \sum_{l=1}^{N_v \times M_v} \frac{(y_l - f_{NN}(u_l, \theta))^2}{\hat{\sigma}_{y,l}^2 + \left(\frac{\partial f_{NN}(u_l, \theta)}{\partial u_l}\right)^2 \hat{\sigma}_{u,l}^2}.$$  

(6-68)

6.9 Modeling scheme

Earlier it was stated that EIV is a rather slow method. Moreover, simulations showed that EIV demands good starting values in order to converge. Since convergence is easily reached with the OE method currently used in most mathematical packages, the following procedure is used, which showed to be very robust:

1. With common OE methods, based on a LS cost function, the starting values for the NN parameters are calculated using
The EIV cost function for nonlinear systems

\[ \theta_{OE} = \arg \min_{\theta} \left( \frac{1}{N} \sum_{k=1}^{N} (\hat{y}_k - f_{NN}(\hat{u}_k, \theta))^2 \right) \]  

(6-69)

2. Choose the proper value for \( \mu \) or \( \eta \), based on the chosen optimization routine. The initial values for \( \hat{u} \) and \( \hat{\sigma}^2 \) are chosen according to equation (6-3). The initial NN parameters are \( \theta^{(0)} = \theta_{OE} \) and the starting value for \( u \) is taken as \( u^{(0)} = \hat{u} \).

3. From these initial values, start the following iteration

3a. Choose a \( \Delta p \) (one of the equations (6-48), (6-50), (6-60) or (6-61)) and calculate the new parameter vector \( p^{(k+1)} = p^{(k)} + \Delta p^{(k)} \).

3b. With the new \( \theta^{(k+1)} \) values, check the cost function \( C_{ES} \) (6-67) or (6-68) on the validation data. If the cost starts rising, stop the iteration process, else proceed with 3a.

6.10 Simulation results

6.10.1 Curve Fitting

The given method was simulated on the function \( y_k = \sin(9 \sin^3(u_k + 1.8) - 1) \) with 500 measurements \( u_k \) taken in the region \( u_k \in [e^0, e^1] - 1.8 \). \( M \) was chosen to be 30 and \(-5.2\text{dB}\) noise was added to get a training set with \( 500 \times 30 \) simulation pairs \( (u_k^{(l)}, y_k^{(l)}) \). The validation set was built with a second set of 15,000 pairs. \( M \) was considered to be large, such that the simplified cost function (6-36) was used for the calculation of the parameter updates.

The neural network used was a two layer perceptron with 10 neurons with a hyperbolic tangent transfer function and bias. The topology of the network is shown in FIGURE 6-3 in which the NN parameters equal \( \theta = \left[ W_1^T B_1^T W_2^T B_2^T \right]^T \). The dashed square encloses one layer of the NN which is denoted as \( f_{ts}(W_1 u_k^{(l)} + B_1) \) with \( f_{ts} \) a tan-sigmoid activation function, and

![FIGURE 6-3 Neural Network topology](image-url)
which is applied on each of the 15000 measurement samples. The $a \times b$ values that are given above the matrices and the paths in the NN, denote the sizes of the matrices and information paths.

The simulations were repeated 4,000 times following the procedure of the previous section and using the Levenberg-Marquardt optimization step (6-50). After the simulation, the relative RMS error between the true function $f(u)$ and the NN mapping $f_{NN}(u)$ was calculated as

$$E_{RMS} = \frac{\sum_{i=1}^{N_G} [f(u_i) - f_{NN}(u_i)]^2}{\sum_{i=1}^{N_G} [f(u_i)]^2} \quad (6-70)$$

with $u_i \in [e^0 \ e^1] - 1.8$, using a fine grid ($N_G$ very large). From the simulations we selected 3,930 simulations that had less than 20% relative RMS error between the data and the NN mapping after performing the OE optimization. I.e. 70 simulations were considered to have a bad convergence. The mean result is shown in FIGURE 6-4.

Note that different $\hat{u}_k$ data points are taken for each simulation, dependent on the added noise. The sample means shown in the figure are only those used in the last simulation.

![FIGURE 6-4 Simulation results of EIV compared to OE.](image)

The mean number of iterations for OE to reach convergence in this example (using early stopping), was 22. The mean computing time to reach this convergence was approximately 6.2 seconds. The mean number of epoch needed for EIV, starting from the OE result, was 3.6. This
took another 75 seconds of computing time. In 89% of the cases EIV needed less than 5 epoch to reach the new convergence point for the parameters.

The Least Squares output error on the mean of the 3,930 results, compared with the original function, dropped from 6.2% for OE to 0.8% for EIV when using the Neural Network with noiseless input data. The NN parameters were also compared using a test set of 2000 input-output pairs in which noise was added to the inputs with $\sigma_u^2 = 13$ dB. The Least Squares output error, compared with the true outputs, dropped from 14% in the OE case to 7.3% in the EIV case.

FIGURE 6-5 shows a histogram of the 3,930 RMS errors of the NN mappings compared with the true functional. From the figure it follows that more simulations had a low RMS error after using the EIV cost function than when using the LS cost function: on the average the RMS error drops from 7.5% to 5.1%. The EIV mapping isn’t guaranteed to perform better in all cases. In this example the error of the EIV mapping was lower than the error of the OE mapping in 81% of the cases.

### 6.10.2 Circle fitting

To demonstrate the biasing effects of input noise when performing a nonlinear mapping, Amemiya [11] proposes the fitting of a circle. For the simulations a SIMO (Single-Input Multiple-Output) version of the EIV cost function is used. One output maps the upper part of a circle shaped waveform, another output maps the lower part.

This example is repeated four times, such that a comparison can be made between the simplified EIV cost function and the full EIV cost function. The circle is sampled twice with a
low number of measurement points (e.g. \( N = 16 \) and \( M = 6 \)) and twice with a high number of measurement points (e.g. \( N = 180 \) and \( M = 200 \)).

In all cases the simulations were done with a large amount of noise on the measured samples. Since the simulations were done with different machines and different programs, the results are not fully comparable. Yet, a brief comparison is made at the end of this section.

A. **simplified EIV cost function with sparse data**

The optimization of the NN parameters was done using backpropagation with an adaptive learning rate \( \eta \). In the first example the circle is sampled 12 times at 16 points with \(-7\text{dB}\) noise at the inputs and \(-40\text{dB}\) noise at the outputs. The data are then split in a learning set and a validation set, with the learning set of size \( 16 \times 6 \) with \( M = 6 \) and the validation set containing 96 samples.

Note that the choice for \( M \) is the minimal value required by equation (6-35) with sample means and sample variances. In order to compare OE with EIV, both are trained with the 16 sample means, and both use the same validation set for early stopping. Both methods use a two-layer perceptron NN architecture with 4 neurons in the hidden layer.

The simulations were repeated 5000 times. 70 simulations had more than 20% residual error after training with the OE cost function, and were skipped. The mean epoch for OE to reach the point of early stopping was 5, which took about 0.4 seconds. The postprocessing with EIV needed 33 epoch on the average, which took another 2.6 seconds for each simulation.

The mappings of both NN are compared with the original circular signal. The relative RMS error on the mappings drops from 9.8% in the OE case to 8.9% in the EIV case. The mean results of the 5000 simulations are shown in FIGURE 6-6. It is clear that the large noise levels...
have caused the mappings to be severely biased. From the figure it can be seen that the EIV mapping is slightly less biased than the OE mapping.

The EIV fit on the data outperforms the OE mapping in 69% of the 5,000 simulations and in the mean the mapping resembles better the true plant characteristics. FIGURE 6-7 shows a histogram of the RMS errors of both mappings compared with the true function. The figure shows that there are more EIV mappings with a lower RMS error, while more OE mappings have an RMS error that is above 20%.

B. simplified EIV cost function with large data set

A second circle mapping used 3200 samples with \( M = 1 \), \(-15\text{dB}\) noise was put on the inputs and the data was split in to two sets of 1600 samples. Since more samples were available, a two-layer perceptron network with 20 neurons was chosen both for OE and for EIV. The OE cost function was minimized with a Levenberg-Marquardt optimization step. On a total of 30 simulations the mean epoch was 7.7, which took 2 minutes.

For the EIV postprocessing a backpropagation scheme with adaptive learning rate was needed due to the large number of samples. The mean epoch to reach the new NN parameters was 1308, needing approximately 6 hours. The error of the mappings, compared with the true function dropped from 10.3% to 4.4%. The mean result of the simulations is shown in FIGURE 6-8.

The larger number of samples result in an significant improvement during the postprocessing. The EIV fit on the data outperforms the OE mapping in 93% of the cases. It is very likely that due to the large number of samples the EIV mapping is more accurate than the OE mapping.
The histogram of all simulations (FIGURE 6-9) again shows that in general the EIV mappings have a lower RMS error when compared with the true function.

**C. Full EIV cost function with sparse data**

The simulations that were done in section A. were repeated using the full EIV cost function (6-59) on page 158. The LM optimization step (6-61) was used to learn the NN parameters.

The circle is sampled 12 times at 16 points with $-7\text{dB}$ noise at the inputs and $-40\text{dB}$ noise at the outputs. The data are then split in a learning set and a validation set of equal size. The NN used is a two-layer perceptron NN architecture with 4 neurons in the hidden layer as it was the case in section A.
The simulations were repeated 5000 times. The simulations that had more than 20% residual error after training with the OE cost function were skipped. The mean epoch for OE to reach the point of early stopping was 6.8, which took about 0.27 seconds. The postprocessing with EIV needed 8 epoch on the average, which took one more second. Note that a different type of computer was used for these simulations, such that the processing times shown here must not be compared with the times in the previous sections.

The mappings of both NN are compared with the original circular signal. The relative RMS error on the mappings drops from 25.6% in the OE case to 22.8% in the EIV case. The mean
Simulation results

results of the 5000 simulations are shown in FIGURE 6-10. From the figure it can be seen that the EIV mapping is again less biased than the OE mapping.

The EIV fit on the data outperforms the OE mapping in 64% of the 5,000 simulations and in the mean the mapping resembles better the true plant characteristics. FIGURE 6-11 shows the histogram of the RMS errors of both mappings compared with the true function.

D. Full EIV cost function with large data set

The simulations that were done in section B. make use of not-repeated measurements ($M = 1$). In this simulation a larger number of repeated measurements is taken ($M = 200$), combined with a large number of different measurements ($N = 181$). It is now possible to calculate the full crossvariances, needed for the full LM optimization step (6-61) on page 158.

In this simulation a very high level of noisy was used. The circle is sampled with +9dB noise at the inputs and –3dB noise at the outputs. These levels of noise are possible because of the higher number of repeated measurements. The NN used is a two-layer perceptron NN architecture with 20 neurons in the hidden layer as it was the case in section B.

The simulations were repeated 3010 times. From these simulations 268 (8.9%) had more than 20 % residual error after training with the OE cost function, and were skipped. Even using the LM optimization step, the mean epoch for OE to reach the point of early stopping was 7.5, which took 1.5 seconds. The postprocessing with EIV and an LM optimization step needed 8.4 epoch on the average, which took 90 seconds.
The relative RMS error on the mappings drops from 17% in the OE case to 13% in the EIV case. The mean results are shown in FIGURE 6-12. The EIV fit on the data outperforms the OE mapping in 94% of the 3000 simulations. This is shown in the histogram of FIGURE 6-13. Although a very large number of the simulations perform better when using EIV, this doesn’t really show in FIGURE 6-12. The better performance can easier be seen in the histogram.

**E. Comparison**

In all four examples the EIV cost function outperforms the OE cost function. The increase in performance is typically 10%. When comparing the performance of the full EIV cost function (part C.) with that of the simplified cost function (part A.) for the case of a small number of repeated measurements, a marginal increase of performance is reached. The performance increase goes from 9% (decrease in error from 9.8% to 8.9%) in the simplified EIV case to 11% (decrease in error from 25.6% to 22.8%) when using the full EIV cost function.

It seems therefore that the use of the full EIV cost function is justified for a small number of repeated measurements. Yet, the number of cases where EIV outperforms the OE solution, drops from 69% (simplified EIV) to 64% (full EIV). Due to the very high amount of noise on the simulation samples, there is still a high variance on the simulation results, despite the large number of simulations. Since the EIV cost function should be especially used with high input noise, this variance is hard to avoid.

The conclusion is that the use of the full EIV cost function instead of the simplified EIV cost might give an added performance, but this can not be shown in the examples. When compared with the OE cost function, an increase in performance is very likely.
6.10.3 Plant modeling

The quality of the steel in a steel converter is directly dependent on the end temperature of the steel in the melt. This temperature is controlled with the amount of oxygen that is blown into the melt where 20 input parameters are expected to influence this temperature. These parameters are given in table 6-1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>[Min / Max] value</th>
<th>$\sigma_u$</th>
<th>Unit</th>
<th>(Input / Output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converter age</td>
<td>[1 / 3000]</td>
<td>0</td>
<td>days</td>
<td>(I)</td>
</tr>
<tr>
<td>Measure time</td>
<td>[1 / 4]</td>
<td>1</td>
<td>min</td>
<td>(I)</td>
</tr>
<tr>
<td>Batch type</td>
<td>[30 / 40]</td>
<td>0</td>
<td></td>
<td>(I)</td>
</tr>
<tr>
<td>Raw Iron</td>
<td>[95 / 120]</td>
<td>0.36</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Added Steel 1</td>
<td>[15 / 45]</td>
<td>1.22</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Added Steel 2</td>
<td>[0 / 19]</td>
<td>0.99</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Added Steel 3</td>
<td>[0 / 11]</td>
<td>0.57</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Starting Value C</td>
<td>[4 / 4.7]</td>
<td>2%</td>
<td>%</td>
<td>(I)</td>
</tr>
<tr>
<td>Goal temperature</td>
<td>[1650 / 1690]</td>
<td>0</td>
<td>°K</td>
<td>(I)</td>
</tr>
<tr>
<td>Oxygen</td>
<td>[7000 / 8500]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Feed Type 1</td>
<td>[800 / 1800]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
</tr>
<tr>
<td>Feed Type 2</td>
<td>[500 / 2000]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
</tr>
<tr>
<td>Feed Type 3</td>
<td>[-150 / 150]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
</tr>
<tr>
<td>$Mn$</td>
<td>[0.4 / 1.4]</td>
<td>1%</td>
<td>%</td>
<td>(I)</td>
</tr>
<tr>
<td>$Si$</td>
<td>[0.3 / 1.7]</td>
<td>2%</td>
<td>%</td>
<td>(I)</td>
</tr>
<tr>
<td>$N_2$</td>
<td>[0 / 40]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>$Ar$</td>
<td>[0 / 60]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>$CaO_2$</td>
<td>[5 / 12]</td>
<td>0.05</td>
<td>ton</td>
<td>(I)</td>
</tr>
<tr>
<td>Additive 1</td>
<td>[0 / 500]</td>
<td>10</td>
<td>kg</td>
<td>(I)</td>
</tr>
<tr>
<td>Phase 1 temperature</td>
<td>[1503 / 1683]</td>
<td>4</td>
<td>°K</td>
<td>(I)</td>
</tr>
<tr>
<td>Steel end temperature</td>
<td>[1913 / 1983]</td>
<td>4</td>
<td>°K</td>
<td>(O)</td>
</tr>
</tbody>
</table>

**TABLE 6-1.** Main steel converter parameters

Some of the parameters have a nonlinear relationship to the steel temperature, and a white box identification of the plant is hardly possible. In a first stage, the goal of the NN mapping is to provide a plant model in order to predict the steel temperature of the plant. In a second stage an
inversion of the black box model is used in order to predict the needed amount of pure oxygen. Since no repeated measurements are available, $M$ equals one.

The variances on the measured inputs are given (estimated) by the steel company, but during simulations the dynamic range between the largest and the smallest variances was limited to 20 dB. Since no repeated measurements were made, no calculation of the covariance matrices is possible, and the simplified EIV cost function (6-36) on page 153 was used.

2091 measurements were available for the mapping. From these measurements, one fifth was used for the validation set, and one seventh was used as a test set. The other samples were used for learning the NN parameters. The choice of the set sizes was arbitrarily, keeping in mind that a maximum of samples should be reserved for learning.

The NN topology used has 20 inputs, one output and 5 sigmoidal neurons in the hidden layer. The OE cost function is minimized with a Levenberg-Marquardt optimization step, while the EIV postprocessing is done using backpropagation with an adaptive learning rate. The simulations were repeated 5,000 times, picking different learning, validation and test sets out of the measurements on a random basis. The performance of the NN mapping is measured by the prediction of the output steel temperature. A hit is recorded if the prediction lays within an interval $[-10,+15]$ degrees Kelvin from the true end temperature. To date the hit rate of the steel plant operators is 64%.

After the OE mappings all NN that had a hit rate below 50% were considered to have a bad convergence, such that 80 NN mappings were skipped. The mean hit rate of the OE mappings on the test set was 68.9%. The minimum on the validation set was reached in a mean of 15 iterations, which took about 12 seconds. The EIV postprocessing spent another 35 seconds in order to reach the new minimum on the validation set in 121 iterations, using backpropagation.

The mean hit rate for the EIV mapping was 69.7%. One could argue that the gain in performance (0.8%) is marginal, but it should kept in mind that the test set is also built from noisy data. In the first example was already shown that the performance decreases when using noisy samples, while the gain in performance of the EIV mapping is mainly due to a better modeling of the plant.

The first example in this section also shows how the difference between OE and EIV becomes less with very low signal-to-noise ratios. Yet, even in this example the small effort of using the EIV postprocessing leads to a direct improvement of the steel quality.

For sake of completeness, the histogram of the RMS errors is given in FIGURE 6-14. One should keep in mind that this histogram is not a comparison of the model to the true plant, though based on noisy measurements. Yet it can be seen that there is a slight improvement of the RMS error when using EIV. The EIV postprocessing lead to a better hit rate of the NN model in 71% of the simulations.
6.10.4 Plane mapping

The goal of this example is the control of a flexible manipulator. Two motors control the length of two pulling cables that bent a spring into a far nonlinear state. The spring is then used as a highly flexible robot arm. The measurement of the end tip of the spring within 4 square feet, is carried out with a magnetic inductor, which suffers from induced noise.

A white box identification of the robot setup was proposed but proved to be non-inversable [224]. The goal of this example is to examine the usefulness of a NN mapping for the plant. The NN should give the position $z$ of each motor for every given position of the end tip of the manipulator in a plane $(x, y)$.

In order to test the method, this example makes use of an arbitrary exactly known surface, rather than the robot data. A nonlinear surface is built upon the transfer function

$$z(x, y) = \sin(-3x)\cos(-5y)$$

and sampled at 820 instances with $-20$dB noise on both inputs $x$ and $y$ and the output $z$. $M$ is chosen as one, and the EIV cost function was trained with the true variances (20 dB). The covariance terms $\sigma_{u, k}^2$ and $\sigma_{yu, k}^2$ are known to be zero, and equation (6-36) on page 153 is used for the calculation of the cost function.

The neural network used was a two layer perceptron with 10 neurons in the hidden layer. Early stopping was used, based on a second set of 820 samples. The OE cost function is minimized with Levenberg-Marquardt, while the EIV cost function uses backpropagation.
The simulations were repeated 700 times and 634 samples had a residual error of less than 20% after the OE mapping. The mean result is shown in FIGURE 6-15. The error compared with the true function dropped from 4.8% when using the OE cost function (15 seconds for 71 epoch) to 3.5% when using EIV (99 seconds for 447 epoch).

A histogram of the RMS error over all 700 simulations is shown in FIGURE 6-16 The figure shows that more EIV mappings have a lower RMS error. In 84% of the simulations the EIV mapping performed better than the OE mapping.

FIGURE 6-15 Mapping of a plane

FIGURE 6-16 Histogram of RMS errors.
6.11 Reduced Models

Consider the case where only the output measurements contain noise. This means that all inputs are well known, or $u_k^i \equiv u_k \equiv u_k^*$, thus eliminating the second term in the cost function (6-21). The EIV algorithm is then reduced to Bayesian NN learning [26].

When the variances are not known, it is possible to set all $\sigma_{y_k}^2 \equiv 1$. In this case the EIV cost function is reduced to the OE cost function. The OE method is, therefore, a special case of the EIV method.

Another possibility is when only the input measurements contain noise. An example where this can happen is when the data are used to invert a plant transfer function, based on a number of measurements [224]. The optimization reduces to a constrained nonlinear minimization based on a cost

$$C^* = \frac{1}{N} \sum_{k=1}^{N} \frac{(\hat{u}_k - u_k)^2}{\hat{\sigma}_{u_k}^2}$$ (6-72)

with constraints

$$y_k = f_{NN}(u_k, \theta).$$ (6-73)

Instead of seeking a new optimization routine, the existing routines can be used by setting the $\hat{\sigma}_{y_k}^2$ in equation (6-35) to a very small value.

6.12 Conclusions

Using the Errors-In-Variables cost function with Black Box systems is less trivial than using it with systems that are linear in the observations. EIV allows that the measurements are shifted on all axes, taking the variance of the measurements in respect. For BB modeling this increases the risk of overfitting significantly. An example of this was given, using NN modeling.

It must be stressed that the NN parameters obtained from noisy input and output measurements with the EIV cost function are meant for use with noiseless input data, as is the case for inversing control and simulation purposes. However, simulations also indicate an improved performance when using the EIV NN parameters with noisy input data.

This chapter showed how the overfitting behaviour can be taken care for with the use of a dedicated early stopping algorithm. From that point on, the use of the EIV cost function results in a better BB mapping of noisy measurement data, compared with the LS cost function. This
also was demonstrated with NN modeling with an optimization scheme on the implementation of EIV using Backpropagation and Levenberg-Marquardt. Moreover, different examples were given to illustrate the optimization.

The extra cost to perform the identification is the required knowledge of the measured data variances. The gain of the algorithm is the improved performance of the identification when both the input data and output data contain noise or when only the input data contain noise. The drawback of the method is an increased demand for computing time and memory needs.

In order to optimize the BB parameters with the EIV cost function it is advised to start from the parameters that were found using the OE cost function. It is possible to turn this drawback into an advantage, using EIV as a postprocessing tool for a classic OE mapping. This way an experimenter can still rely on the existing BB modeling techniques, while accepting the EIV results only whenever these prove to give a better mapping of the measurement data.
Abstract: Within the domain of NN identification, there is a belief that creating new data samples by simply adding existing measurements with additive noise, will help to obtain a better generalization of the NN model. This is, however, not guaranteed. Usually the NN models are trained using a simple Least Squares cost function. In that case adding input noise indeed results in a smoother NN model. In this section it will be proven that adding noise on the input measurements causes biasing in the parameters and a loss in performance of nonlinear BB models in general, i.e. the BB model doesn’t map the true function anymore.


7.1 Introduction

7.1.1 Chapter outline

This chapter discusses the popular technique of adding noisy samples to the existing measurements. The introduction gives a few definitions and explains the method used more in detail.

To clarify the idea that is brought in this chapter, section 7.2 starts with the simple SISO case and shows why adding noise leads to errors in the model performance. The analysis is based on a higher order Taylor series expansion of the cost function. It will be shown that adding noise increases the probability that higher order derivatives of a BB model are suppressed.

A better solution than adding noise, is the use of interpolation for the case where one needs a better generalization of the BB model. This is explained in detail in section 7.4. The best method is the use of interpolation techniques to create artificial measurements [230] [233]. Examples will be given that clearly show the improved performance over the technique of adding noise.

Finally, In section 7.5 the proofs that are given in section 7.2 are repeated for the MIMO case.

7.1.2 About generalization

Despite of the many BB models used in the literature, a model is basically a compression algorithm. The goal is the exact description of a nonlinear surface or hypersurface with a small number of parameters $\theta$.

As a rule of thumb it is said that at least ten to twenty times more measurements must be taken than the number of parameters used in the model. For BB models one must take into account that some or all of the model parameters only operate on a restricted region of the domain of interest. It is possible that only a limited number of measurements was drawn from a given region, such that the actual optimization of the parameters is done with a limited part of the measurements. In general it can therefore be stated that a larger number of measurements is needed than what is the case for classic linear modeling.

Assume the particular case where an experimenter expects his BB model to perform bad in a certain region of the domain of interest, e.g. caused by a lack of measurements available in that region. Additional measurements are not possible but still the experimenter wants to impose a given behaviour in that region. Usually it is demanded that the model should have a good generalization (see Definition 2.1 on page 16), but in practise the demand is reduced to the requirement that the model should have a smooth behaviour in between measurement points.
Two causes are pinpointed for a bad generalization. The first is due to overfitting of noisy samples, the second being bias effects caused by noise at the input of a nonlinear model. The best way to guarantee a good generalization, therefore, is to obtain enough noiseless measurements. The system modeled must be measured in the whole domain of interest with a sufficient density such that detailed information is extracted from the modeled hyperplane.

To explain some of the concepts in the sequel of this chapter, it is necessary to make a distinction between variance and variability of a model. Consider a plant with a true linear or nonlinear transfer function \( y^* = f_{BB}(u^*, \theta^*) \) with \( \theta^* \) the true parameters of a BB mapping, based on the noiseless input-output measurements \( u^* \in \mathbb{R}^m \) and \( y^* \in \mathbb{R}^n \). The plant is measured \( N \) times with noisy measurements \( u_k = u_k^* + n_{u,k} \) and \( y_k = y_k^* + n_{y,k} \) with \( u_k \in \mathbb{R}^m, y_k \in \mathbb{R}^n \) and \( k = 1, 2, ..., N \). A large number of BB models \( f_{BB}(u, \hat{\theta}_i(u_k, y_k)) \) is mapped on the measurement data, with \( \hat{\theta}_i(u_k, y_k) \) the estimated BB parameters based on the noisy measurements and starting from different initial values for \( \theta, i = 1, 2, ..., K \) and \( K \) the total number of BB models that were mapped.

The different model parameters are obtained using different initial values for the optimization vector. Two cases are considered where the mapping results in BB parameters that are different from the true parameters \( \theta^* \). The first case is where a nonperfect mapping is caused by noisy measurement data. The second case is where a nonperfect mapping is caused by unwanted behaviour of the optimization algorithm, such that it gets stuck in a local minimum instead of reaching the global minimum, or such that early stopping is initiated at the wrong moment.

A. The parameter variance case

A high variance can result when a BB model with a high number of parameters is mapped on sparse data. Some parameters possibly only operate on a region where no measurements were taken, leading to unpredicted behaviour of the BB model in these regions. Another cause for high variance is the use of an early stopping algorithm to avoid overfitting on noisy data. If many different BB models were mapped starting from random initial parameters, it is highly unlikely that the optimization of the models leads to the same input-output behaviour in all cases.

All of these examples result in a variability on the BB model that can be independent from the possibility of noise on the measurement data. Consider therefore the following definitions which are given for the case of noiseless measurement data.

**Definition 7.1** Consider a BB model that is mapped on a set of noiseless measurements \((u^*, y^*)\). The mapping is repeated \( K \) times, starting from different initial BB parameters. The *mean BB mapping* on the measurement data, is defined as
On generalization by adding noise

\[ f_{BB, \theta}(x) = \lim_{K \to \infty} \left( \frac{1}{K} \sum_{i=1}^{K} f_{BB}(x, \hat{\theta}_i(u^*, y^*)) \right) \bigg|_{x \in \psi} \]  \hspace{1cm} (7-1)

with \( \hat{\theta}_i(u^*, y^*) \) the \( K \) estimated BB parameters, based on the same input-output measurements \( (u^*, y^*) \), \( \psi \) the domain of interest, defined in section 2.4, and \( x \in \mathbb{R}^m \) any possible input from the domain of interest. Note that \( f_{BB, \theta}(x) \) is a vector in the MIMO case.

**Remark:** The use of \( x \) instead of \( u \) is preferred here to stress that the input \( x \) is chosen independently from the measurements.

**Definition 7.2** The variability of a BB model is the variance of an infinite number of BB models that are mapped on the measurement data, and which is caused by parameter uncertainty due to the minimization algorithm that is initiated with different starting values. It is defined as

\[ v_{f_{BB}}^2(x) = E_{\xi}\{ \| f_{BB}(x, \hat{\Theta}(\xi)) - E_{\xi}\{ f_{BB}(x, \hat{\Theta}(\xi)) \|_2^2 \} \bigg|_{x \in \psi} \} \]  \hspace{1cm} (7-2)

where \( E_{\xi}\{\cdot\} \) denotes the expected value with respect to different starting values for \( \Theta \) and the results of e.g. an early stopping algorithm, and \( \hat{\Theta}(\xi) \) are the estimated parameters for different BB mappings on the same data.

The variability of the BB mapping can in practise be calculated as

\[ v_{f_{BB}}^2(x) = \lim_{K \to \infty} \left( \frac{1}{K} \sum_{i=1}^{K} \| f_{BB}(x, \hat{\theta}_i(u^*, y^*)) - \bar{f}_{BB, \theta}(x) \|_2^2 \right) \bigg|_{x \in \psi} \]  \hspace{1cm} (7-3)

where the input \( x \) is chosen by the experimenter, e.g. based on a very fine grid that covers the domain of interest. In practise, the calculation of the variance is done on a finite number of measurements only, such that \( K \) is finite.

**Remark:** In this thesis, biasing effects on the parameters caused by different starting values and the use of early stopping algorithms, are assumed to be zero and will not be considered.

**B. The measurement variance case**

Consider the case where different BB models are mapped on \( K \) different sets of measurement data. For each mapping, a new set of noisy measurements is chosen.
**Assumption 7.1** Assume that there are no “bad” mappings, i.e. mappings that get stuck in local minima or that have a bad generalization.

Under Assumption 7.1, the BB modeling doesn’t suffer from overfitting behaviour, and variations on the different BB mappings are only due to the noise contributions $n_{u,k}$ and $n_{y,k}$ on the measurements $(u_k, y_k)$ in the different measurement sets $i = 1, 2, ..., K$.

**Definition 7.3** Under Assumption 7.1, the mean BB mapping is defined as

$$
\bar{f}_{BB}(x) = \lim_{K \to \infty} \left( \frac{1}{K} \sum_{i=1}^{K} f_{BB}(x, \hat{\theta}_i(u, y)) \right) \Bigg|_{x \in \psi} \tag{7-4}
$$

with $\hat{\theta}_i(u, y)$ the estimated BB parameters, based on the input-output measurements $(u, y)_i = \{(u_1, y_1), (u_2, y_2), ..., (u_N, y_N)_i\}$, $i = 1, 2, ..., K$, and $\psi$ the domain of interest, defined in section 2.4.

**Definition 7.4** Under Assumption 7.1, the variance of a BB model is defined as the variance of an infinite number of BB models that are mapped on the measurement data, and which is caused by the noise contributions $n_u$ and $n_y$ on the measurement data, or

$$
\sigma^2_{f_{BB}}(x) = E_{n_u,n_y} \{ \| f_{BB}(x, \hat{\theta}(u, y)) - E_{n_u,n_y} [f_{BB}(u, \hat{\theta}(u, y))] \|_2^2 \} \Bigg|_{x \in \psi} \tag{7-5}
$$

where $E_{n_u,n_y}\{\cdot\}$ denotes the expected value with respect to the noise contributions on the inputs and outputs, and $\hat{\theta}$ are the estimated parameters for each new set of measurements $(u, y)$.

In practise, the variance of the BB mapping can be calculated as

$$
\sigma^2_{f_{BB}}(x) = \lim_{K \to \infty} \left( \frac{1}{K-1} \sum_{i=1}^{K} \| f_{BB}(x, \theta_i(u, y)) - \bar{f}_{BB}(x) \|_2^2 \right) \Bigg|_{x \in \psi} \tag{7-6}
$$

where $x$ is chosen again by taking a very fine grid in the whole domain of interest. In practise only a limited number of measurements are available, such that $K$ is finite. The calculation of $\sigma^2_{f_{BB}}(x)$ is therefore more a theoretical than a practical issue.

The noise on the inputs not only leads to a variance on the BB model behaviour, but in general also causes biasing effects, in which case the mean model $y = \bar{f}_{BB}(u)$ doesn’t map the true function $y^* = f_{BB}(u^*, \theta^*)$. 

Definition 7.5 Under Assumption 7.1, the bias due to noisy data, is defined as

\[
\beta_{\text{f}_{\text{BB}}} (x) = E_{\epsilon_1, \epsilon_2} \{ f_{\text{BB}}(x, \hat{\theta}(u, y)) \} - f_{\text{BB}}(x, \theta^*) \bigg|_{x \in \Psi}.
\]

In the sequel of this chapter examples are given where the true function \( f_{\text{BB}}(u^*, \theta^*) \) is known, such that the bias can be calculated using

\[
\beta_{\text{f}_{\text{BB}}} (x) \to \text{Lim}_{K \to \infty} \left( \frac{1}{K} \sum_{i=1}^{K} [ f_{\text{BB}}(x, \hat{\theta}_i(u, y)) - f_{\text{BB}}(x, \theta^*) ] \right)_{x \in \Psi}.
\]

In the examples given further, \( K \) is finite but a large number of simulations is performed such that conclusions can be drawn in general.

C. The general case

It should be noted that the variability on a model can be much larger than the model variance caused by noise variance. This is especially true in the case of sparse data. In the case of noisy measurements, and if only one single BB mapping is realized on the measurement data, the variance and the variability of the BB model are inseparable. If a large number of mappings are performed on each set of measurements, the separation could be achieved. In practise the inhibitive large number of BB mappings needed for this separation, makes that it is not considered.

It can also be expected that the variability is badly influenced by noisy measurement data because of a larger potential risk to get stuck in local minima during the optimization. Therefore, in general Assumption 7.1 doesn’t apply. From a theoretical point of view, and given a set of measurements, a BB model can have a very high variance with zero variability, thus mapping all measurement data points exactly the same way during each optimization and for each new measurement set. Meanwhile, because of overfitting of the noisy measurement data a good model for the true plant is never reached and the model could be of little use, e.g. when used for simulation purposes.

Even if a BB model has a high variability, the mean result of a very large number of BB mappings, may still give the true functional, i.e. \( \bar{f}_{\text{BB}, \theta}(u^*) \equiv f_{\text{BB}}(u^*, \theta^*) \). A model can, therefore, have a high variability, while its mean outcome has zero variability. A few examples are given in FIGURE 7-1

Making the difference between variability and variance is rather important when discussing generalization. The goal of a good generalization should be to achieve a low variability, low
variance, together with low bias. Reaching a low variance with a low bias is therefore not sufficient.

7.1.3 Improving generalization using jitter

Within the NN domain a simple to use technique exists that is generally accepted to improve the generalization behaviour of a NN model. In this technique jitter (call it noise) is added to the existing measurements, thus increasing the existing data set with noisy data samples that are based on the original data set with additive noise. The NN model parameters are then optimized using this enlarged data set. The noise is typically added both on the input and the

FIGURE 7-1 Examples of [TOP LEFT] a mapping with a high variability, low variance and zero bias, [TOP RIGHT] a mapping with high variance but zero variability and bias, and [BOTTOM] a biased mapping with low variance and low variability. In each figure, the solid line is the true function $f_{BB}(u^*, \theta^*)$, the gray lines the mean $\bar{f}_{BB}(u)$, minimum $\inf(f_{BB})$ and maximum $\sup(f_{BB})$ values of 100 NN mappings. In the top left figure the mean mapping is the same as the true function. In the top right figure the mean, minimum and maximum values are the same, indicating that the noisy data is always mapped the same way.

7.1.3 Improving generalization using jitter

Within the NN domain a simple to use technique exists that is generally accepted to improve the generalization behaviour of a NN model. In this technique jitter (call it noise) is added to the existing measurements, thus increasing the existing data set with noisy data samples that are based on the original data set with additive noise. The NN model parameters are then optimized using this enlarged data set. The noise is typically added both on the input and the
output measurements. The reason for adding noise is based on the misconception that NN’s come from the AI domain and contain some hidden “intelligent” feature such that they are capable to find the best mapping for measurement data.

Bishop [26] states that incrementing training data by adding noisy data allows for the use of more complex models, which then reduce the bias. For his statement, Bishop uses the knowledge that one of the causes for bias in a BB model is a too low number of model parameters used. Not all details of the hyperplane are mapped, which resembles biasing behaviour. Adding noise, however, introduces bias at the level of the identified BB model, as is shown further in this chapter. This bias cannot be overcome by picking a more complex BB model, or by increasing the number of parameters.

The uncomfortable part when adding noise is that normally one tries to eliminate the effects of noise [71] [149] [197], or one tries to estimate the true input values out of the learning set based on the cost function [236]. It seems therefore questionable why someone wants to introduce noise in the measurements on purpose. In this chapter it will be shown that adding noise to the measurement samples is a bad idea in all cases.

7.1.4 The bias/variance trade-off

No source can be found where it is proven that adding output noise indeed improves generalization. On the other hand, different sources have studied the effects of adding input noise to NN models (An [13], Bishop [26], Geman et al. [75], Haykin [86], Mackay [145], Reed et al. [182] and Twomey et al. [218], Wright [251]). It is stated that input noise can be used, as long as it is balanced with the model variance.

Sometimes the use of noise with low, but nonzero variances is recommended (e.g. [63] and [145] for NN models), while other sources stress the importance of using large enough variances such that generalization is guaranteed. Within the NN domain, this is called the bias/variance trade-off. Because of the unpredictable results that can be caused by adding noise, it is generally accepted that the level of noise must be designed, e.g. by putting constraints in the BB architecture [86] which makes the simple method of adding noise suddenly more complex. Yet, this complexity cannot be avoided and the variance of the noise must be chosen carefully [26]. The key insight to the bias/variance trade-off problem comes from the decomposition of NN by Geman [75]. The decomposition is based on a second order analysis from a pure probabilistic point of view.

In the literature the discussion on the bias/variance trade-off is often restricted to the variance of the BB mapping with respect to the true plant. The variability from one BB realization to the other, using a large number of BB mappings on the same data, is usually neglected which was also pointed out by An [13].
Geman reports in his much referenced example [75] that adding noise results in a nonlinear mapping that comes very close to the true functional in at least one case. First, this implies that the true functional is known, which is usually not the case. Second, the model variability is completely left out of the picture in this discussion, as only one realization was taken as an example. It is, therefore, always necessary to analyse the mean BB mapping $f_{BB}$.

The difference between the model variance and variability is discussed in detail in the example of section 7.4, and shows how large noise levels on the measurement samples indeed result in lower variability but also that biasing effects come into play. To partially avoid the risk that the noise variance has a severe influence on the model variance, a very large number ($N > 1000$) of repeated noisy measurements [13], or very large data sets [75] can be chosen, which results in a slow optimization process and huge memory needs.

### 7.2 Analysis of the effects of adding noise on measurement data

Consider a SISO system that is measured $N$ times with measured input samples $u_k \in \mathbb{R}$ and output samples $y_k \in \mathbb{R}$ where $k = 1, 2, ..., N$. The input-output pairs $(u_k, y_k)$ are used to optimize the parameters of a BB model

$$f_{BB} : \mathbb{R} \rightarrow \mathbb{R} : y_k = f_{BB}(u_k, \theta) \quad (7-9)$$

**Assumption 7.2** The parameters $\theta$ of the nonlinear model are optimized by minimizing the LS cost function

$$C_{LS} = \frac{1}{N} \sum_{k=1}^{N} (y_k - f_{BB}(u_k, \theta))^2. \quad (7-10)$$

This assumption means that no specific care is taken to minimize the effects of noisy measurements, as can be done using a WLS or EIV cost function or regularization terms.

**Assumption 7.3** There is a unique parameter vector, denoted as $\hat{\theta} = \hat{\theta}(u_k, y_k)$, for which the cost function (7-10) has a unique global minimum.

This assumption is made to simplify the notations of what is brought in the sequel of this chapter. In practice it is very well possible that more parameter vectors exist that all have the same BB input-output relationship $f_{BB}(u_k, \hat{\theta})$ and that all give the same global minimum. The
calculations should then be repeated for each of these parameter vectors. The conclusions, however, remain the same in all cases.

**Assumption 7.4** In the case that enough noiseless measurements are available and that the measurements cover the whole domain of interest, it is possible to find the true BB parameters $\theta^*$, or

$$\hat{\theta}(u^*, y^*) = \theta^*. \quad (7-11)$$

The idea behind this assumption is that the BB model is expected to be a universal approximator.

Now consider the case where the experimenter decides to add noise as a means of generalization. In the most general case the addition of noise is done by taking the measurements $R$ times and add noise on the resulting measurement set of size $R \times N$ such that

$$u_k^{[r]} = u_k + n_k^{[r]} \quad \text{and} \quad y_k^{[r]} = y_k + n_k^{[r]} \quad (7-12)$$

with $n_k^{[r]} \in \mathbb{R}$ and $n_k^{[r]} \in \mathbb{R}$ random values, and $r = 1, 2, ..., R$.

**Assumption 7.5** $n_k^{[r]}$ and $n_k^{[r]}$ are independent, mutually uncorrelated, zero mean Gaussian distributed random variables with known standard deviations $\sigma_{u,k}$, $\sigma_{y,k}$ and such that $\sigma_{uy,k} = 0; \forall k$.

The former LS cost function now also sums the added noisy samples and becomes

$$C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \Theta) = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_k^{[r]}, \Theta) - y_k^{[r]} \right)^2. \quad (7-13)$$

**Assumption 7.6** It is assumed that a very large number of noisy samples is taken, or

$$R \to \infty. \quad (7-14)$$

**Lemma 7.1** Under assumptions 7.5 and 7.6 the LS cost function (7-13) converges strongly to its expected value uniformly in $\Theta$, or

$$\text{a.s.lim}_{R \to \infty} E_{n_k^{[r]}, y_k^{[r]}} \left[ C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \Theta) - C_{LS}(u_k, y_k, \Theta) \right] = 0 \quad (7-15)$$

**Proof:** see Lukacs [142].

---

186
Analysis of the effects of adding noise on measurement data

Hence to study the influence of the added noise samples for $R \to \infty$ it is sufficient to study $E\{ C_{LS}(R, u_k^r, y_k^r, \theta) \}$ which is more tractable than (7-13). Since the added noisy samples on inputs and outputs are mutually uncorrelated, the two cases are treated differently in the next paragraphs.

A. Adding output noise

**Theorem 7.1** Under assumptions 7.2 to 7.6 the addition of noisy output samples for training BB models, has no effect on finding the BB parameters $\theta^*$ that minimize the LS cost function.

**Proof:** see APPENDIX A.

B. Adding input noise

**Theorem 7.2** Under assumptions 7.2 to 7.6 the addition of noisy input samples leads to a suppression of higher order derivatives of the BB function. This suppression is proportional to the variance of the added noise, due to a regularizer term in the cost function, such that

$$E\{ C_{LS} \} = \frac{1}{N} \sum_{k=1}^{N} \left[ \left( f_{BB}(u_k^r, \theta) - y_k \right)^2 + \epsilon_k^2(\sigma_u^2) \right]$$

with $\epsilon_k^2(\sigma_u^2)$ a regularizer term that depends on the higher order derivatives of $f_{BB}$ and the noise variance and which equals

$$\epsilon_k^2 = \sum_{l=1}^{\infty} \sigma_{k, u}^{2l} \frac{(2l)!}{2^l l!^3} \left( \frac{\partial^{l} f_{BB}(u_k, \theta)}{\partial u_k^l} \right)^2$$

$$+ \sum_{l=2t=1}^{\infty} \sum_{l=1}^{\infty} \sigma_{k, u}^{2l} \frac{(2l)!}{2^{l} l!^3} \frac{\partial^{l} f_{BB}(u_k, \theta)}{\partial u_k^l} \frac{\partial^{(2l-t)} f_{BB}(u_k, \theta)}{\partial u_k^{2l-t}}.$$

**Proof:** see APPENDIX A.

Based on this theorem we can conclude that the noise puts a constraint on the higher order derivatives of the BB model. The regularizer term $\epsilon_k^2$ increases when $\sigma_u^2$ increases. In effect, adding noise makes the resulting model smoother and influences directly the variability of the model, but it must be well understood that no conclusion can be made concerning the variance of a single mapping.
The noise, however, also introduces bias. The $\varepsilon_k^2$ regularizer is not $\theta$ independent and it is highly unlikely that both the regularizer and the original function have the same minimum for $\theta$. As a result the regularizer pulls the BB parameters away from the ideal solution $\theta^*$. This effect also increases with the amplitude of the added noise.

The effects of the higher order derivatives should not be underestimated. Opposed to systems that are linear in the parameters and where the second and higher order derivatives are zero, the maximum values of the higher order derivatives in nonlinear systems tend to increase with the order.

To fix the ideas, assume that a SISO one hidden layer perceptron NN architecture is trained. The NN uses the $\tanh(\cdot)$ activation function in the hidden layer and a linear neuron in the output layer. The NN transfer function becomes

$$f_{NN}(u_k, \theta) = \sum_{i=1}^{P} (w_2^{[i]} \tanh(w_1^{[i]}u_k + b_1^{[i]}) + b_2)$$  \hspace{1cm} (7-18)$$

with $P$ the number of neurons and $B_1 = [b_1^{[1]}, b_1^{[2]}, ..., b_1^{[P]}]^T$, $B_2 = [b_2]$, $W_1 = [w_1^{[1]}, w_1^{[2]}, ..., w_1^{[P]}]^T$ and $W_2 = [w_2^{[1]}, w_2^{[2]}, ..., w_2^{[P]}]^T$. For a SISO system, the parameter vector can be written as $\theta = [W_1^T, B_1^T, W_2^T, B_2^T]^T$. It is possible to calculate the $j$-th derivative of this NN model, with respect to $u_k$, as

$$\frac{\partial f_{NN}(u[k], \theta)}{\partial u[k]^j} = \sum_{i=1}^{P} w_2^{[i]} \frac{\partial \tanh(a_k^{[i]})}{\partial a_k^{[i]^j}} (w_1^{[i]^j})$$  \hspace{1cm} (7-19)$$

FIGURE 7-2 Higher order derivatives of the $\tanh(.)$ function. Shown here are the first five derivatives.
with \( a_k^{[i]} = w^{[i]} u_k + b^{[i]} \). The higher order derivatives of the \( \tanh(.) \) part in this equation become increasingly large. This is shown in FIGURE 7-2 for the first five derivatives.

The given maximum values don’t apply in all cases. A given neuron can be in a state such that the maximum value for a given derivative is not reached or the neuron may not be effective. Yet, the probability that higher order derivatives have large effects on the mapping, increases with the order. The effects of this are only tempered by the \( \sigma_{k,u}^{2l}(2l)!/(2^l(l!)^3) \) term in the regularizer.

<table>
<thead>
<tr>
<th>ORDER</th>
<th>MAXIMUM VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.7698</td>
</tr>
<tr>
<td>3</td>
<td>2.000</td>
</tr>
<tr>
<td>4</td>
<td>4.086</td>
</tr>
<tr>
<td>5</td>
<td>16.01</td>
</tr>
<tr>
<td>6</td>
<td>52.27</td>
</tr>
<tr>
<td>7</td>
<td>272.0</td>
</tr>
<tr>
<td>8</td>
<td>1223.7</td>
</tr>
<tr>
<td>9</td>
<td>7935.8</td>
</tr>
<tr>
<td>10</td>
<td>45569</td>
</tr>
<tr>
<td>11</td>
<td>357800</td>
</tr>
</tbody>
</table>

**TABLE 7-1.** Maximum absolute values of higher order derivatives of the \( \tanh(.) \) function.

---

**C. Adding both input and output noise**

The case where noisy samples are added to both the input and the output samples, reduces to the previous case, as is expressed in the following theorem.

**Theorem 7.3** Under assumptions 7.2 to 7.6 the addition of noisy input samples and noisy output samples, has the same effect as adding noisy input samples only.

**Proof:** see APPENDIX A. □

---

**7.3 The relation with regularization**

Going back to the theory of regularization, consider the definition given by Haykin [86]. For Tikhonov regularization, an extra term is added to the cost function, such that
On generalization by adding noise

\[ C_R = C(u_k, y_k, \theta) + \delta \epsilon_c(\theta) \]  \hspace{1cm} (7-20)

with \( C_R \) the new cost function based on regularization, \( C \) any of the cost functions defined in Chapter 2, \( \delta \) the regularization parameter and \( \epsilon_c(\theta) \) some complexity penalty imposed on the model and its parameters. One choice of the complexity penalty term is the \( k \)-th order smoothing integral

\[ \epsilon_c(\theta, k) = \frac{1}{2} \int \left\| \frac{\partial^k}{\partial u_k} f_{BB}(u, \theta) \right\|^2 \mu(u) du \] \hspace{1cm} (7-21)

where \( \mu(u) \) is some weighting function that determines the region of the input space over which the mapping \( f_{BB}(u, \theta) \) is required to be smooth. The motivation is to make the \( k \)-th derivative of \( f_{BB}(u, \theta) \) with respect to \( u \) small [86].

The complexity penalty (7-21) can be compared with the effects of adding noise, described in (7-16) and (7-17). Choosing the weighting function as a constant value

\[ \mu(u) = \mu = \sigma_k^2 \frac{(2I)!}{2^{(l)}I^l} \] \hspace{1cm} (7-22)

it then follows that adding noise can be considered as a way of regularization where \( k \) is chosen even, which results in local smoothing instead of global smoothing [155]. The same result was recently found by Aires et al. [4].

### 7.4 An alternative for adding noise

We now know that only input noise is really effective to decrease the variance of the BB model (not considering variability and bias). To fix the ideas, and as an illustration, consider the arbitrary SISO nonlinear functional

\[ y_k = \sin(-1 + 7(\sin(\exp(u_k))))^5) \] \hspace{1cm} (7-23)

The functional is sampled at 11 equidistant points with \( u_{[k]} \in [0, 1] \) and a multilayer perceptron NN is trained with 5 neurons in the hidden layer. The NN transfer function is given by (7-18) such that

\[ f_{NN}(u_k, \theta) = W_2^T \tanh(W_1^T u_k + B_1) + B_2 \] \hspace{1cm} (7-24)

in which the \( \tanh(\cdot) \) function is taken element wise. A NN mapping using noiseless data points is shown in FIGURE 7-3 The simulations were repeated 100 times and the mean results of these 100 simulations are shown, with the standard deviations on the mappings. From the
figure and the inset, it can be seen that, despite of the variability on the mappings, all mappings fit well the simulation data. We are in the case of sparse, but exact, data.

Now consider the case where the experimenter decides to “introduce generalization” by adding noisy samples. To fill in the gaps in between the measurements, 100 noisy realizations ($R = 100$) are added to each input sample, with a standard deviation $\sigma = 0.03$.

![Figure 7-3](image.png)

**FIGURE 7-3** Simulation results of 100 NN mappings on sparse data.

The results of this mapping are given in **FIGURE 7-4** It can be seen that the variability isn’t much less, while there is clearly a bias effect: the mean result of the mapping doesn’t quite include all of the original measurement points (see inset). From this simple experiment it can already be concluded that adding noise might result in a smoother mapping, but the variability on the mapping remains, while bias is introduced.

![Figure 7-4](image.png)

**FIGURE 7-4** Using noisy input data to obtain better generalization.
It is our observation that the bias effects are more severe whenever the input samples start to overlap. This can be easily explained when looking at the rightmost example of FIGURE 7-5. Whenever the measurements start to overlap, the LS error of the overlapping sections becomes so large that the added penalty for not mapping the original data is negligible.

In the NN domain the \( \sigma \) is called the *ridge value* and both statistical and empirical methods exist to obtain this value. Choosing the ridge value too high causes overlapping input data with an increased danger of bias effects. A too low ridge value is of no help. In the case of a nonequidistant grid (as in FIGURE 7-5) the choice of the proper noise level is a tedious task and in practise it is said that the noise value should be optimized carefully [26].

One might wonder why it is needed to go through the pain of choosing this proper noise value that is needed to obtain a good generalization. Adding noisy measurements makes that the experimenter is less aware of what is happening with the NN mapping. It is a non-proven belief that adding noise on the loose makes things better.

The remaining question is how a good mapping can be achieved if the input samples are too few to guarantee a good generalization in between the measurements. In practise it is the experimenter who should give constraints for the model, e.g. the maximum frequency content that is expected in the measurement data. Remind that adding noise to the inputs also puts a constraint on the model, more precise the constraint that the mapping should be smooth, albeit in a not well controlled sense.

Looking at the middle example of FIGURE 7-5 the optimal amount of noise has much resemblance with a simple zero-order interpolation of the data. A much easier way of imposing a smoothing constraint would be the use of a first or second order interpolation technique [233] or the use of multilinear interpolation with a given upper frequency limit [230].

FIGURE 7-5 Choosing the proper ridge value for a MLP network with 10 neurons in the hidden layer. [LEFT] A too low ridge value leaves a risk for overfitting. [MIDDLE] The optimal amount of noise should preserve the measurements and avoid overfitting. [RIGHT] Too much noise leads to overlapping simulation data and introduces bias errors, i.e. the original data is not necessarily mapped.
To illustrate this, the previous example of FIGURE 7-3 was repeated using a multilinear interpolation [230] with an upper limit frequency of 8 Hz on the 11 measurement points, and a NN was used as a compression tool for the interpolated data.

The original 11 measurements were interpolated with a finer grid of 500 points and a NN was trained using the finer grid. Because of the larger number of interpolated points, more neurons could be used in the hidden layer without risk of overfitting. The results (mean value and variability) of 100 simulations with 10 neurons in the hidden layer are shown in FIGURE 7-6.

![FIGURE 7-6 modelization with interpolated data (interpolated points not shown).](image)

It is clear that the interpolation results in far less variability of the NN models, while biasing effects are practically absent. Moreover the use of multilinear interpolation guarantees a bandlimited interpolation and thus a smooth mapping.

### 7.5 The MIMO case

To simplify the idea that is brought in this chapter, the previous sections considered the SISO case of the problem only. In this section the proofs are elaborated and it is proven that the principles brought before, also apply for the MIMO case.

Therefore, consider a MIMO system that is measured $N$ times with measured input samples

$$ u_k = [u_{k,1}, u_{k,2}, ..., u_{k,m}]^T \in \mathbb{R}^m \quad k = 1, 2, ..., N $$

(7-25)

and output samples

$$ y_k = [y_{k,1}, y_{k,2}, ..., y_{k,n}]^T \in \mathbb{R}^n $$

(7-26)
On generalization by adding noise

and \( k = 1, 2, \ldots, N \). The input-output pairs \((u_k, y_k)\) are used to optimize the parameters of a BB model

\[
f_{BB}: \mathbb{R}^m \to \mathbb{R}^n; y = f_{BB}(u, \theta) \tag{7-27}
\]

where the elements of the output vector \( y = [y_1, y_2, \ldots, y_n]^T \) are mapped by the individual BB models \( y_i = f_{BB,i}(u_k, \theta) \).

**Assumption 7.7** The parameters \( \theta \) of the nonlinear model are optimized by minimizing the LS cost function

\[
C_{LS} = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{n} \sum_{i=1}^{n} e_{k,i}^2 \tag{7-28}
\]

with the error \( e_{k,i} \) defined as

\[
e_{k,i} = y_{k,i} - f_{BB,i}(u_k, \theta). \tag{7-29}
\]

Again, no special precautions are taken to minimize biasing effects.

In accordance with assumptions 7.3 and 7.4, the true MIMO parameters \( \theta^* \) that model the measured plant are found as \( \hat{\theta}(u^*, y^*) = \theta^* \) where \( \hat{\theta} = \arg \min_{\theta} C_{LS}(u, y, \theta) \) is the minimizer of (7-28).

Consider again the case where the experimenter decides to add noise on the input and output measurements. The measurements are taken \( R \) times and noise is added on the resulting measurement set of size \( R \times N \) such that

\[
u_k^{[r]} = u_k + n^{[r]}_{k,u}
\]

and

\[
y_k^{[r]} = y_k + n^{[r]}_{k,y}
\]

with \((n^{[r]}_{k,u} = [n^{[r]}_{k,1,u}, n^{[r]}_{k,2,u}, \ldots, n^{[r]}_{k,n,u}] \in \mathbb{R}^m \) and \( n^{[r]}_{k,y} = [n^{[r]}_{k,1,y}, n^{[r]}_{k,2,y}, \ldots, n^{[r]}_{k,n,y}] \in \mathbb{R}^n \) vectors filled with random values, and \( r = 1, 2, \ldots, R \). The LS cost function (7-28) now also sums the added noisy samples and becomes

\[
C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \theta) = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \sum_{i=1}^{n} \left( f_{BB,i}(u_k^{[r]}, \theta) - y_k^{[r]} \right)^2, \tag{7-32}
\]
**Assumption 7.8** $n_{k,u}^{[r]}$ and $n_{k,y}^{[r]}$ are independent, mutually uncorrelated, zero mean Gaussian distributed random variables with known variances $\sigma_{u,k}^2$, $\sigma_{y,k}^2$ and such that $\sigma_{u,y,k}^2 = 0$ and $\sigma_{y,u,k}^2 = 0$, $\forall k$. The covariance matrix only contains diagonal elements.

**Lemma 7.2** Under assumptions 7.5 and 7.6 the LS cost function (7-32) converges strongly to its expected value, or:

$$\text{a.s.lim }_{\mathcal{R} \to \infty} E_{n_u,n_y} \{ C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \Theta) \} - C_{LS}(u_k, y_k, \Theta) = 0 \quad (7-33)$$

**Proof:** see Lukacs [142].

Therefore, also in the MIMO case, it is sufficient to study $E\{ C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \Theta) \}$ instead of $C_{LS}(R, u_k^{[r]}, y_k^{[r]}, \Theta)$.

**A. Adding output noise**

**Theorem 7.4** Under assumptions 7.6, 7.7 and 7.8 the addition of noisy output samples for training MIMO BB models, has no effect on finding the BB parameters $\Theta^*$ that minimize the LS cost function 7-28.

**Proof:** see APPENDIX A.

**B. Adding input noise**

**Theorem 7.5** Under assumptions 7.6, 7.7 and 7.8 the addition of noisy input samples leads to a suppression of higher order derivatives of the MIMO BB function. This suppression is proportional to the variance of the added noise, due to a regulizer term in the cost function, such that

$$E_{n_u} \{ C_{LS} \} = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \left[ \left( f_{BB,i}(u_k, \Theta) - y_{k,i} \right)^2 + \varepsilon_{k,i}^2(\sigma_u) \right] \quad (7-34)$$

with $\varepsilon_{k,i}^2(\sigma_u)$ a regulizer term that is only dependent on the higher order derivatives of $f_{BB}$ multiplied with the noise variance.

**Proof:** see APPENDIX A.
Based on this theorem the conclusions that were drawn for the SISO case, also apply in the MIMO case.

C. Adding both input and output noise

For MIMO BB modeling the case where noisy samples are added to both input and output samples reduces to the previous case, as is expressed in the following theorem.

**Theorem 7.6** When using a MIMO BB model, and under assumptions 7.6, 7.7 and 7.8 the addition of noisy input samples and noisy output samples, has the same effect as adding only noisy input samples.

**Proof:** see APPENDIX A. □

### 7.6 Conclusions

This chapter showed that deliberately adding noisy measurements to an existing measurement set, is a bad idea. The technique of adding noise is, however, a popular method in the NN modeling field to improve generalization.

To analyse the method, this book defined the variability concept, next to the variance of a model. It was shown that adding noise is a means of decreasing the variance. Indeed, adding noise makes that the nonlinear model is less prone to overfitting behaviour, as higher order derivatives of the model are more and more suppressed. Yet, no conclusions can be drawn about the variability of the NN model, while there is no doubt that biasing errors are introduced.

In the past years of research, not a single occurrence could be detected where the sum of the variance, bias and variability were in a sort of minimum with respect to the noise variance. Moreover, despite the many papers dedicated to the subject, no description of such a minimum has been found in the literature. There do exist examples where the variance was decreased with an acceptable increase in bias. It is questionable whether this can be defined as a better generalization.

This thesis proposes interpolation techniques as an alternative. It allows to create a larger data set where some physical meaning remains, *in casu* an upper frequency filter on the data. BB models are then used as a compression tool on the data.
8.1 Steel plant analysis and optimization

8.1.1 Introduction

Even if parts of a chemical process are well known and can be described precisely, describing a complete chemical plant is a difficult task. In this section a steel plant is modelled that can be seen as a nonlinear MIMO system. The control of the plant is currently based on human experience and a model of the plant is needed for optimizing the steel quality and plant throughput. The problems that have to be overcome are a limited number of measurement samples, low accuracy and a high uncertainty. Different Neural Network models are described for making an optimal use of the given measurements.

Two novel techniques are described in this section. First, it is assumed that the estimated plant model equals the true plant model based on the results of a large number of NN models and early stopping.

Second, a proposal for optimization of the plant is given by introducing a novel method that uses Variance Backpropagation of the controlled output to obtain the input variances that can influence the product quality. A number of input candidates that should be better controlled or measured, are pinpointed. Suggestions are given where to improve the measurements in order to improve the NN performance.

8.1.2 Steel plant description

The steel plant that is modelled is a chemical converter that burns down the carbon level in a mix of melted iron and chemical additives. This is done by blowing pure oxygen in the melt. The goal of the modeling effort is to predict the amount of pure oxygen that is needed in the process to reach a specified Carbon concentration and end temperature of the steel. The modeling is restricted to the prediction of the end temperature for an externally specified amount of oxygen. The underlying idea is that it is possible to control the plant, once it can be accurately modelled (Certainty Equivalence Principle). We assume that the plant is inversible because to date this is done manually and intuitively.

The steel is processed in two phases. In a first stage raw iron and waste steel are melted with added chemicals. The melted iron is then poured in a convertor and during the second stage pure oxygen is blown into the melt to obtain a specified Carbon content and specified goal temperature. This two stage process will further be called a batch. The total number of input and output parameters for the system equals 59. Not all of these parameters are useful for the NN modeling and a selection was made with a maximum of 43 input parameters, based on expert knowledge.
Table 8-1 gives an overview of the most important parameters, with their tolerances and minimum/maximum boundaries. An (I) indicates that the value is known prior to the batch and can be used as an input to the NN. An (O) indicates that the value is measured at the end of the batch and can only be used for validation. The sensitivity values $s_i^l$ and $s_i^n$ are discussed in section 8.1.5.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>[Min / Max] value</th>
<th>$\sigma_u$</th>
<th>Unit</th>
<th>(I) input / (O) output</th>
<th>$s_i^l$</th>
<th>$s_i^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converter age</td>
<td>[1 / 3000]</td>
<td>0</td>
<td>days</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooling time</td>
<td>[5 / 25]</td>
<td>1</td>
<td>min</td>
<td>(I)</td>
<td>0.32</td>
<td>0.30</td>
</tr>
<tr>
<td>Measure time</td>
<td>[1 / 4]</td>
<td>1</td>
<td>min</td>
<td>(I)</td>
<td>-1.75</td>
<td>-2.33</td>
</tr>
<tr>
<td>Batch type</td>
<td>[30 / 40]</td>
<td>0</td>
<td></td>
<td>(I)</td>
<td>0.22</td>
<td>0.2</td>
</tr>
<tr>
<td>Raw Iron</td>
<td>[95 / 120]</td>
<td>0.36</td>
<td>ton</td>
<td>(I)</td>
<td>-1.21</td>
<td>-1.2</td>
</tr>
<tr>
<td>Added Steel 1</td>
<td>[15 / 45]</td>
<td>1.22</td>
<td>ton</td>
<td>(I)</td>
<td>-2.89</td>
<td>-3</td>
</tr>
<tr>
<td>Added Steel 2</td>
<td>[0 / 19]</td>
<td>0.99</td>
<td>ton</td>
<td>(I)</td>
<td>-3.39</td>
<td>-3.68</td>
</tr>
<tr>
<td>Added Steel 3</td>
<td>[0 / 11]</td>
<td>0.57</td>
<td>ton</td>
<td>(I)</td>
<td>-4.73</td>
<td>-4.73</td>
</tr>
<tr>
<td>Start Value C</td>
<td>[4 / 4.7]</td>
<td>2%</td>
<td>%</td>
<td>(I)</td>
<td>-8.55</td>
<td>-8.57</td>
</tr>
<tr>
<td>Goal Value C</td>
<td>[0.03 / 0.06]</td>
<td>10%</td>
<td>%</td>
<td>(O)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal temperature</td>
<td>[1650 / 1690]</td>
<td>0</td>
<td>°K</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxygen</td>
<td>[7000 / 8500]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
<td>33.0 $10^{-3}$</td>
<td>33.3 $10^{-3}$</td>
</tr>
<tr>
<td>Feed Type 1</td>
<td>[800 / 1800]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
<td>15.6 $10^{-3}$</td>
<td>16.0 $10^{-3}$</td>
</tr>
<tr>
<td>Feed Type 2</td>
<td>[500 / 2000]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
<td>3.95 $10^{-3}$</td>
<td>4 $10^{-3}$</td>
</tr>
<tr>
<td>Feed Type 3</td>
<td>[-150 / 150]</td>
<td>10</td>
<td>mm</td>
<td>(I)</td>
<td>-13.2 $10^{-3}$</td>
<td>16.6 $10^{-3}$</td>
</tr>
<tr>
<td>$Mn$</td>
<td>[0.4 / 1.4]</td>
<td>1%</td>
<td>%</td>
<td>(I)</td>
<td>8.23</td>
<td>9</td>
</tr>
<tr>
<td>$Si$</td>
<td>[0.3 / 1.7]</td>
<td>2%</td>
<td>%</td>
<td>(I)</td>
<td>38.5</td>
<td>35.7</td>
</tr>
<tr>
<td>$N_2$</td>
<td>[0 / 40]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
<td>0.158</td>
<td>0.15</td>
</tr>
<tr>
<td>$Ar$</td>
<td>[0 / 60]</td>
<td>0.5%</td>
<td>ton</td>
<td>(I)</td>
<td>-43.1 $10^{-3}$</td>
<td>-66.7 $10^{-3}$</td>
</tr>
<tr>
<td>$CaO_2$</td>
<td>[5 / 12]</td>
<td>0.05</td>
<td>ton</td>
<td>(I)</td>
<td>-5.62 $10^{-3}$</td>
<td>-5.7 $10^{-3}$</td>
</tr>
<tr>
<td>Additive 1</td>
<td>[0 / 500]</td>
<td>10</td>
<td>kg</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additive 2</td>
<td>[12 / 24]</td>
<td>5%</td>
<td>kg</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additive 3</td>
<td>[3.5 / 9]</td>
<td>?</td>
<td>kg</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additive 4</td>
<td>[0.1 / 0.45]</td>
<td>3%</td>
<td>%</td>
<td>(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 1 temperature</td>
<td>[1503 / 1683]</td>
<td>4</td>
<td>°K</td>
<td>(I)</td>
<td>0.221</td>
<td>0.222</td>
</tr>
<tr>
<td>Steel end temperature</td>
<td>[1913 / 1983]</td>
<td>4</td>
<td>°K</td>
<td>(O)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 8-1.** Main converter parameters used. $s_i^l$ and $s_i^n$ are defined in section 8.1.5.
detail in section 8.1.5. In the sequel references to the parameters in Table 8-1 will be written in *italic*.

The goal of the modeling is the prediction of *Steel end Temperature* within the boundaries \([-10, +15]\) °K from a given goal temperature. Denote \(h\) as the hit rate that the end temperature lies within this temperature window, or:

\[
h = 100 \cdot P(\text{Phase2temp} \in \text{GoalTemp} + [-10, 15]).
\]  

(8-1)

To date, the hit rate for the manually controlled plant equals \(h = 60.7\%\).

### 8.1.3 Analysis of the measurement data

The data was already filtered for measurement errors by taking out the outliers and incomplete measurements. After this manual filtering of the data set, 2091 measurement samples remained that were considered to be usable for learning. Histograms show that not all parameters have Gaussian shaped noise distributions. From the data it is impossible to determine whether this is due to uncertainties, random noise or *Batch Type* specifications. For that reason all noise contributions to the measurement data are considered to be Gaussian shaped with zero mean. For some parameters the tolerance is estimated, as shown in Table 8-1. Further a more detailed analysis is made in order to know if such an estimation is justified.

The samples in the data set do not necessarily come from subsequent batches. Large gaps exist in the *Converter age*, during which special batches were performed that are not included in the measurements. Analysis showed that the hit rate during subsequent batches is significantly higher. The reason is that the batch responsibles based the amount of needed *Oxygen* on the results of previous batches. It is therefore justified to consider NARX models [140] [205] besides NFIR models. Limited data sets are created by sliding a window over the current data set and rejecting data points if not all of the data points within the window are subsequent. These limited data sets will further be called *windowed* data sets. FIGURE 8-1 shows how windowing reduces the number of useable data samples which puts severe constraints on the number of parameters that can be optimized.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td># Samples</td>
<td>2091</td>
<td>945</td>
<td>579</td>
<td>380</td>
<td>243</td>
<td>152</td>
<td>100</td>
</tr>
</tbody>
</table>

**FIGURE 8-1** Sample set size versus window size

Windowing is also used to calculate variances on the measurement samples for use with Bayesian Learning [26] [235]. Calculating the variance of the data samples in a window must be done with caution. This is certainly the case with smaller window sizes, in which the
smaller number of measurement samples can lead to unpredictable variances. For that reason the dynamic range of the variances is limited to 10dB.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mean value</th>
<th>95% boundary (guessed)</th>
<th>95% boundary (measurements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added Steel 1</td>
<td>28.8 ton</td>
<td>1.22 ton</td>
<td>9.10 ton</td>
</tr>
<tr>
<td>Added Steel 2</td>
<td>7.18 ton</td>
<td>0.99 ton</td>
<td>7.17 ton</td>
</tr>
<tr>
<td>Added Steel 3</td>
<td>4.72 ton</td>
<td>0.57 ton</td>
<td>3.19 ton</td>
</tr>
<tr>
<td>Raw Iron</td>
<td>112.5 ton</td>
<td>0.36 ton</td>
<td>6.52 ton</td>
</tr>
</tbody>
</table>

**TABLE 8-2.** 95% boundaries of guessed values.

Special care is needed when using the parameters *Raw Iron, Added Steel 1, Added Steel 2* and *Added Steel 3*. The amount of pure iron in these four parameters are guessed values and are fully based upon the skills of the batch responsibles. E.g. the percentage of pure iron in a pile of raw ore is such an estimated value. In Table 8-2 the 95% boundaries of these guesses are shown, based on the mean of all guesses of the seven batch responsibles. I.e. the mean of all guesses of each responsible is taken and the variance of the seven means is calculated. From the table can be concluded that these means have remarkable small 95% boundaries and that care must be taken when using these boundaries. For that reason 95% boundaries are taken, based on all measurements. The used boundaries, therefore, also include the variance on the used *Added Steel*, and not only measurement errors.

### 8.1.4 Modeling

**A. The NN structure**

The Neural Network (NN) topology used, was a one hidden layer Multilayer Perceptron (MLP) with a tangent hyperbolic function in the hidden layer and a linear function for the output layer neuron. The number of hidden neurons was typically 5, while the number of neurons in the input and output layers differ with the number of chosen input and output parameters. Denote the number of input parameters as $n$, the number of training samples as $N$ and the number of outputs as $m$. The inputs are denoted as $u_k^i$ with $k = 1, 2, ..., N$ and $i = 1, 2, ..., n$, while the outputs are denoted as $y_j^k$ with $j = 1, 2, ..., m$. In this section we consider one output parameter (*Steel end temperature*) only such that $m = 1$. The weights for the first layer are defined by an weight matrix $W_1$ with size $5 \times (n + 1)$ (5 neurons and n
inputs + bias term). For the output layer this becomes an $m \times 6$ matrix $W_2$. The vector of NN parameters is denoted as $\theta = [W_1^T \; W_2^T]^T$.

B. Choice of inputs and outputs

For the modeling, three different mappings were considered. The first was an NFIR structure

$$y_k = f_{NN}(u_k, u_{k-1}, \ldots, u_{k-WS+1}, \theta) \quad (8-2)$$

with $u_k = [u_k[1], u_k[2], \ldots, u_k[n]]^T$, $y_k = [y_k[1], y_k[2], \ldots, y_k[m]]^T$ and in which $WS$ denotes the window size. The first mapping used a Least Squares cost function, while the second mapping implemented the NFIR with a Bayesian (WLS) cost function

$$C_{WLS} = \sum_{k=1}^{N} \frac{(y_k - f_{NN}(u_k, u_{k-1}, \ldots, u_{k-WS+1}, \theta))^2}{\sigma_{y,k}^2}. \quad (8-3)$$

A third NN model mapped the dynamic behaviour with an NARX model (nonlinear ARX [140] [205])

$$y_k = f_{NN}(u_k, u_{k-1}, \ldots, u_{k-WS+1}, y_{k-1}, y_{k-2}, \ldots, y_{k-WS+1}, \theta) \quad (8-4)$$

with an LS cost function. The NN function is defined as $f_{NN} = W_2 \tanh(W_1(x_k))$ in which the $x_k$ contains the $U = [u_1, u_2, \ldots, u_k, \ldots]$ and $Y = [y_1, y_2, \ldots, y_k, \ldots]$ parameters, enlarged with the bias term. The learning step was calculated using a Levenberg-Marquardt optimization step.

C. Identification results

Learning of the NN was done with the use of early stopping [192] in order to prevent overtraining. The data was split in three sets: one seventh of the data was used as a test set in order to test the performance of the obtained model. One fifth of the data was used as a validation set for early stopping. The remainder of the measurement data was used for learning. The sizes for test set and validation set were chosen arbitrarily, keeping in mind that only a limited amount of measurements was available such that choosing three sets of equal size was undesired. The low number of available measurements was a major problem. To fix the ideas consider the case where the NN was trained with 5 neurons, 21 inputs and one output. The number of NN parameters for this case is 116. This becomes as high as 226 parameters if the full NARX model is used with all 43 inputs. The rule of thumb is that modeling should be done with at least 10 to 20 times more measurements than parameters. With 2,091 measurements this rule of thumb is met for both the simple NFIR and the NARX modeling.
The windowed measurements, however, contain far less measurements (FIGURE 8-1) and a trade-off must be made between the improved quality of the windowed samples and the complexity of the model used. Even if it is highly likely that the windowed measurements contain more information with respect to subsequent batches, the decrease in number of samples limited the use of the NARX model to a window size 2.

A total of 26 models was designed, with a different number of inputs and window sizes. Table 8-3 shows the mean performances of the models for the prediction of the end temperature of the steel, compared to the window sizes. The decrease in performance of prediction with larger window sizes is possibly due to the decrease in measurement data. From Table 8-3 it can be concluded that a prediction performance of 72% can be achieved with the NN modeling, compared with a hit rate of 60.7% that is currently reached by the batch leaders.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>NFIR (LS cost function)</th>
<th>NFIR (Bayesian cost function)</th>
<th>NARX</th>
</tr>
</thead>
<tbody>
<tr>
<td>no window</td>
<td>72%</td>
<td>73%</td>
<td>72%</td>
</tr>
<tr>
<td>2</td>
<td>72%</td>
<td>73%</td>
<td>72%</td>
</tr>
<tr>
<td>3</td>
<td>71%</td>
<td>73%</td>
<td>71%</td>
</tr>
<tr>
<td>4</td>
<td>69%</td>
<td>69%</td>
<td>69%</td>
</tr>
</tbody>
</table>

TABLE 8-3. Performance of NN mappings versus window size

In the table can be seen that all NN mappings are seemingly limited to a hit rate of approximately 73%. This indicates that the variance on the output is due to the variance of one of the inputs, and is discussed more in detail in the next section.

8.1.5 Plant analysis based on NN identification

A. Sensitivity Analysis

A sensitivity analysis is used to determine the importance of the different input parameters in the model. If the contribution of each parameter is known, conclusions can be drawn whether this parameter can be left out of the model. It might seem logical to include all possible inputs, no matter how big its contribution to the model is. Adding a single input to the model, however, increases the number of NN parameters with 5 (in the case of 5 hidden neurons). Since the number of data samples is limited, omitting unneeded inputs can lead to a better performing model.

Two numerical methods were considered for the sensitivity analysis. Both methods start from a mean input value, i.e. the system is expected to be excited by a very common input, that is based on mean values of each input. A first linear sensitivity analysis $s_i$ calculates the first
order partial derivative of the output with respect to each input, and is based on the local linearization of the NN model

\[
s_i^l = \frac{\partial}{\partial x_i} f_{NN}(\hat{u}, \theta) \quad \text{with} \quad i = 1, 2, ..., n
\]  

(8-5)

and is calculated in the mean value of the inputs

\[
\hat{u} = \frac{1}{N} \sum_{k=1}^{N} u_k.
\]  

(8-6)

A second nonlinear sensitivity analysis \( s_i^n \) also starts from the mean input values, but sweeps each input from its minimal value to its maximal value, while monitoring the NN mapping. All other inputs are kept at their mean values during this sweep.

\[
s_i^n = \frac{(sup(f_{NN}(\hat{u}, \theta)) - inf(f_{NN}(\hat{u}, \theta)))}{max(u[i]) - min(u[i])}. \quad (8-7)
\]

Care must be taken when interpreting the results from this sort of analysis and the results should not be extrapolated to any given measurement. Yet differences in both sensitivity values show to give an indication whether the contribution of a given input to the output temperature is linear or nonlinear. The \( s_i^l \) and \( s_i^n \) values are given in Table 8-1. The table shows that all input parameters have a significant contribution and none should be left out.

![FIGURE 8-2 Definition of both sensitivity values](image)

**B. Variance analysis**

From the overview of the results in TABLE 8-3, it is apparent that none of the models succeeds to perform better than a hit rate of approximately 73%. For that reason we state the following assumption.

**Assumption 8.1** Due to the early stopping algorithm used, it is accepted that the NN mapping of the steel plant is close enough to or equals the true plant model.
If Assumption 8.1 is accepted, then the only reason for the variance on the output, is that the variance of one of the inputs is too large. Since the sensitivity of the output is known for each input, it is possible to calculate the output variance back to the inputs and compare them with the known or guessed input variances. This method has much correlation with the use of backpropagation for learning of the NN, where the error at the output is propagated back through the NN to become the error on the NN parameters.

$$A \text{ hit rate of 73\% implies a standard deviation on the output of 11.05 } ^\circ \text{K, or a 95\% interval } \Delta T_{\text{steel}} \text{ that equals } \Delta T_{\text{steel}} = 43.3 \text{ } ^\circ \text{K (Phase2temp} \in [2194.3, 2237.7]). \text{ For backpropagation the interval for each parameter is calculated as}$$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\Delta u_i$</th>
<th>$\Delta u_i$ (guessed)</th>
<th>$\Delta v_i$</th>
<th>$\Delta T_{\text{steel}}$ (guessed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooling time</td>
<td>1</td>
<td>0.30</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>Measure time</td>
<td>1</td>
<td>-2.33</td>
<td>2.33</td>
<td></td>
</tr>
<tr>
<td>Batch type</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Raw Iron</td>
<td>6.52</td>
<td>0.36</td>
<td>-1.2</td>
<td>7.82</td>
</tr>
<tr>
<td>Added Steel 1</td>
<td>9.10</td>
<td>1.22</td>
<td>-3</td>
<td>27.3</td>
</tr>
<tr>
<td>Added Steel 2</td>
<td>7.17</td>
<td>0.99</td>
<td>-3.68</td>
<td>26.4</td>
</tr>
<tr>
<td>Added Steel 3</td>
<td>3.19</td>
<td>6.52</td>
<td>-4.73</td>
<td>15.1</td>
</tr>
<tr>
<td>Start Value C</td>
<td>89 $10^{-3}$</td>
<td>-8.57</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>Oxygen</td>
<td>38.51</td>
<td>33.3 $10^{-3}$</td>
<td>1.28</td>
<td></td>
</tr>
<tr>
<td>Feed Type 1</td>
<td>10</td>
<td>16.0 $10^{-3}$</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>Feed Type 2</td>
<td>10</td>
<td>4 $10^{-3}$</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>Feed Type 3</td>
<td>10</td>
<td>16.6 $10^{-3}$</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>$Mn$</td>
<td>7.48 $10^{-3}$</td>
<td>9</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>$Si$</td>
<td>16 $10^{-3}$</td>
<td>35.7</td>
<td>0.57</td>
<td></td>
</tr>
<tr>
<td>$N_2$</td>
<td>78 $10^{-3}$</td>
<td>0.15</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>$Ar$</td>
<td>44 $10^{-3}$</td>
<td>-66.7 $10^{-3}$</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$CaO_2$</td>
<td>50 $10^{-3}$</td>
<td>-5.7 $10^{-3}$</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Phase 1 temperature</td>
<td>4</td>
<td>0.222</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>Steel end temperature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 8-4. Standard deviation analysis**
and should be much larger than the actual interval $\Delta u^i$ of the parameter, if the variance of the parameters has no effect on the variance of the NN output. In order to clarify the results, we will not use backpropagation but instead use a forward calculation of the 95% interval $\Delta T_{steel}^i$ of each input $i$ as

$$\Delta x_{backprop}^i = \Delta T_{steel}^i / s_i$$  \hspace{1cm} (8-8)

This means that, if a given input $i$ has no influence on the variance of the NN, its $\Delta T_{steel}^i$ value should be significantly smaller than $\Delta T_{steel}$. The analysis of TABLE 8-4. shows that the four previously mentioned parameters Raw Iron, Added Steel 1, Added Steel 2 and Added Steel 3 have 95% intervals that lay within an order of magnitude of $\Delta T_{steel}$ and that these inputs should be measured with a higher accuracy. In order to check the validity of the different $\Delta T_{steel}^i$ values, it is possible to calculate a theoretical 95% boundary for $\Delta T_{steel}$, based on the measurements and knowing that

$$\Delta T_{steel, measurements} = \sqrt{\sum_i (\Delta T_{steel}^i)^2} = 41.7$$  \hspace{1cm} (8-10)

which comes close to the measured 95% interval $\Delta T_{steel}$. From TABLE 8-4, we can thus conclude that, in order to improve prediction on the steel end temperature, better measurements for Added Steel 1, Added steel 2 and Added steel 3 are needed.
Abstract: this thesis has tried to be part in the demystification of NN and FL systems. Both modeling techniques are treated as mere input-output relationships in a more general Black Box modeling scheme.

At the end of this thesis, there is still a lot of work to be done. This chapter gives a brief overview of ideas and applications that haven’t been treated yet, and can be considered as future work.
A. Early stopping for linear systems

The technique of early stopping was developed in the field of NN modeling. Based on the scheme of chapter 2, it should be possible to use it for different other modeling techniques, including linear system identification. Although a few references exist where this has been done, further research is still needed. There is an indication that at least for linear systems, early stopping resembles optimization with the EIV cost function.

B. The EIV cost function for Fuzzy Logic systems

In chapter 6, the EIV cost function was applied on NN systems. The expansion of the theory to FL systems and the application to a large number of examples, seems a logical next step. It can be expected that the same conclusions apply as for the NN case, but more detailed research is necessary to confirm this.

C. Model order selection for local linear Fuzzy Systems, based on filtering theory

In Van Gorp and Rolain [230] an interpolation technique was developed, using local linear modeling. The technique is straightforward with little or no further optimization of the parameters necessary. Moreover, the number of local-linear sections shows a direct relationship with the cut-off frequency of a low-pass filter on the measurement data.

This indicates that it should be possible to come up with a necessary and sufficient condition for choosing the number of Fuzzy Sets for FL system identification, and the number of kernels in RBF modeling. The experimenter only needs to provide an upper frequency for the measurement data, which is commonly much easier than the choice of the number of fuzzy sets or RBF kernels.

For the case of FL modeling, the initialization of the fuzzy sets can still be based on the interpolation techniques present. If further optimization is necessary, it can be expected that the FL parameters are already very close to the optimal solution, such that there is only a small risk for the optimization scheme to get stuck in a local minimum of the cost function.

D. The description of Takagi-Sugeno FL systems within $\text{NL}_q$ and $\text{NL}_q^p$ theories

In chapter four a modeling scheme is given that suggests the use of Takagi-Sugeno FL systems. Yet, in this thesis only Mamdani systems are described within $\text{NL}_q$ and $\text{NL}_q^p$. These Mamdani systems only make use of output FS with a fixed area. A class of Takagi-Sugeno systems can be described within the $\text{NL}_q^p$ framework.
There is an indication that this can also be done for more complex TS systems even if Gaussian-shaped Fuzzy Sets of different size are used in the output layer. With the model order selection method described in the previous section, this opens a perspective for a "full" Fuzzy Logic Controller model that combines a standard modeling scheme, model order selection and stability constraints.

E. Less conservative version of $NL_q^p$ theory

Although the theorems for the $NL_q$ and $NL_q^p$ theories can provide stability conditions for nonlinear systems, these theorems are still far too conservative for practical use with complex systems.

For example, consider the case where a SISO FL controller is used to control a linear system. Different simulations are done where the controller is optimized, starting with a simple first order linear system and increasing the order of the linear system. It was observed that the controller had to be made more and more conservative (overdamped) in order to guarantee stability for the closed loop system. A practical upper limit was the case with a fifth order linear system, where the output gain of the controller was set to nearly zero before the stability was theoretically guaranteed.

The effects of the model complexity on the stability condition is still an open question. From recent research in the $NL_q$ field, it can be expected that the stability conditions can be less conservative, when local stability only is considered.

F. Application of the Errors-In-Variables cost function in real-life applications

In the past years, different practical problems were presented to the author, where an EIV based Neural Network seemed a promising model to tackle the problem. One of these problems is the identification of shock absorbers, for which the measurement data were already present, but no time was left for the actual modeling. Another practical problem is the recognition of heart beats in an ECG signal.

G. Mixed model NN and FL identification.

From the typical transfer functions of FIGURE 2-3 on page 21, it can be concluded that e.g. FL systems are better fit for local linear systems and step functions, while Neural Network models provide a soft nonlinear transfer function. Both models are particular successful, such that a mixed model seems worth studying.
Knowing that the \texttt{sat(.)} function can be used in FL models, and the \texttt{tanh(.)} in NN models, a mixed model could have e.g. a transfer function

\begin{equation}
  f_{BB} = \alpha \text{sat}(u) + (1 - \alpha) \text{tanh}(u)
\end{equation}

where the added parameter \( \alpha \) balances between the two models.

\section*{H. Complex model BB identification}

The best model to use when fitting a plant, is the plant itself. Of course, this is not a practical way of working. However, possibly parts of a plant can be used as a kernel function in a more general black box model.

An example of this is the use of basic heart beat patterns as kernel functions in a heart beat detector. A number of functionals are available that are accepted to serve as (part of) a typical heart beat. Using these transfer functions as basic functions for a Neural Network seems a good starting point for a heart beat detector. This concept is related to the use of known phonemes in speech recognition and the use of basic figures in pattern recognition.

\section*{I. Design of a domain of interest \( \psi \)}

One of the issues that wasn’t treated in detail in this thesis, is the way how measurements should be done. The test and measurement domains are complete fields of research on their own, such that it is impossible to tackle the measuring problem in the whole. Yet, there is one problem worth mentioning. For the optimization of the BB parameters, a large number of measurements are needed that are equally spread over the domain of interest.

Moreover, for EIV a number of repeated measurements is needed, while the identification of dynamic systems and recurrent models calls for the use of measurements that are designed for both axes of the phase plane. The design of actuator signals that equally cover the phase plane, has been done at the ELEC department of the Vrije Universiteit Brussel. With these type of actuator signals, the amount of data needed for BB modeling can be less, while a good covering of the domain of interest is guaranteed.
REFERENCES


[74] Ge S. S., "Robust Adaptive Control of Robots Based on Static Neural Networks", 13th Triennial World Congress of the IFAC, San Francisco, USA, 1996, pp. 139 - 144


[147] MacKay D. J. C., "Probable Networks and Plausible Predictions - a Review of Practical Bayesian Methods for Supervised Neural Networks",


[197] Soucek B., the IRIS group (Eds.), "Fast Learning and Invariant Object Recognition, the Sixth-generation Breakthrough", John Wiley & Sons, Inc., 1992.


A.1 Kolmogorov’s theorem: universal approximation of continuous functions

Kolmogorov states that any real-valued continuous function $f(u_1, u_2, ..., u_n) \in \mathbb{R}^n$ defined on an n-dimensional hypercube $I^n$ with $n \geq 2$ and where $I$ denotes the closed unit interval $I = [0, 1]$ and $I^n$ denotes the Cartesian product of $I$, can be represented in the form

$$f(u) = \sum_{j=1}^{2n+1} \chi_j \left( \sum_{i=1}^{n} \psi_{ij}(u_i) \right) \tag{A-1}$$

where $\chi_j$ and $\psi_{ij}$ are continuous functions of one variable and the $\psi_{ij}$ are monotone functions which are not dependent on $f$.

**Proof:** see Kolmogorov [118] [122]. \qed

A.2 Proof of Corollary 4.1 on page 106: reduction of redundant Fuzzy rules

For Fuzzy inference of a single output $y$, the output fuzzy set is calculated as

$$B = \{ (y, \mu_B(y)); y \in \psi \} \tag{A-2}$$

with (equation (4-13) on page 86)

$$\mu_B(y) = I(\mu_{A_1^1 \otimes A_2^1 \otimes \ldots \otimes A_n^1} \otimes B^1(u_1^1, u_2^1, ..., u_p^1, y^1),$$

$$\mu_{A_1^2 \otimes A_2^2 \otimes \ldots \otimes A_n^2} \otimes B^2(u_1^2, u_2^2, ..., u_p^2, y^2), ..., \mu_{A_1^r \otimes A_2^r \otimes \ldots \otimes A_n^r} \otimes B^r(u_1^r, u_2^r, ..., u_p^r, y^r))) \tag{A-3}.$$

Given Assumption 4.6 on page 105, the t-norm ($\otimes$) is chosen as the sum-operator, while the s-norm ($\oplus$) is the product operator, which gives

$$\mu_B(y) = \sum_{i=1}^{R} \mu_{A_1^i}(u_1^i) \mu_{A_2^i}(u_2^i) ... \mu_{A_n^i}(u_m^i) \mu_B(y) \tag{A-4}$$

with $R$ the total number of rules, $u_i^i; i = 1, 2, ..., m$ the $m$ inputs, $\mu_B(y^i)$ the membership degree of the $i$-th rule, and $y$ the output. The consequence $\mu_B(y^i)$ is typically calculated before the evaluation of the rulebase for the inputs $u_1, u_2, ..., u_m$. It is either a constant value, or a lookup table. In the latter case the lookup-table is generated at arbitrarily chosen discrete values that provide a sufficient calculation precision.
A full rulebase has a maximum of \( p_1 \times p_2 \times \ldots \times p_m \) nonredundant rules with \( p_i \) the number of FS for the input \( i \). Now assume that two rules \( i \) and \( j \) are redundant, which means that these rules have the same premises, but conflicting consequences. Moreover, typically a weight factor \( w_i \) or \( w_j \) is used to express the importance of each rule. The combination of these two rules becomes

\[
\mu_{A_i}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) w_i \mu_B(y) + \mu_{A_i}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) w_j \mu_B(y)
\]

(A-5)

where \( \mu_{A_i}(u_1) = \mu_{A_i}(u_1) \), such that the combination can be written as

\[
\mu_{A_i}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) [w_i \mu_B(y) + w_j \mu_B(y)]
\]

(A-6)

or

\[
\mu_{A_i}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) \mu_B'(y).
\]

(A-7)

The term \( \mu_B'(y) = [w_i \mu_B(y) + w_j \mu_B(y)] \) is either a new constant that can be calculated prior to the evaluation of the Rule Base, or a new lookup table that is a linear combination of the two lookup tables. The Rule Base can be reduced with one rule. This is done recursively until only the \( p_1 \times p_2 \times \ldots \times p_m \) nonredundant rules remain. At that point the Rule Base is minimal.

\[\square\]

**A.3 Proof of Theorem 4.1 on page 106: Mamdani FLS are special cases of Takagi-Sugeno FLS**

After inference, the resulting FS equals equation (A-4). Using the CoS defuzzification method from TABLE 4-3. on page 87, the crisp output is calculated as

\[
y = \frac{\int_{x \in \mathcal{V}} \sum_{i=1}^{R} \mu_{A_i}(u_1) \mu_{A_i}(u_2) \ldots \mu_{A_i}(u_m) \mu_B(x) dx}{\int_{x \in \mathcal{V}} \sum_{i=1}^{R} \mu_{A_i}(u_1) \mu_{A_i}(u_2) \ldots \mu_{A_i}(u_m) \mu_B'(x) dx}.
\]

(A-8)

The integral is independent of the summation, so that it is possible to exchange both terms, thus

\[\square\]
The area $b_i$ and the CoG centre $c_j^B$ of each individual output Fuzzy Set are defined as

$$
b_i = \int_{x \in \psi} x \mu_B(x) dx \quad \text{and} \quad c_j^B = \frac{\int_{x \in \psi} x \mu_B(x) dx}{\int_{x \in \psi} \mu_B(x) dx} \quad (A-10)$$

such that (A-9) becomes

$$
y = \frac{\sum_{i=1}^{R} \mu_{A_1}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) \left( \int_{x \in \psi} x \mu_B(x) dx \right)^2}{\sum_{i=1}^{R} \mu_{A_1}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) \int_{x \in \psi} \mu_B(x) dx} \quad (A-11)
$$

$$
y = \frac{\sum_{i=1}^{R} \mu_{A_1}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) b_i c_j^B}{\sum_{i=1}^{R} \mu_{A_1}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) b_i} \quad (A-12)
$$

The term in the denominator can be replaced by a function $f(u_1, u_2, \ldots, u_m)$ in the $u_i$ terms. Since it is assumed that the output FS are constant, these functions have constant parameters for each individual rule. In the case that triangular and trapezoid functions were used for the input FS, the denominator becomes a polynomial function in the $u_i$. To fix the ideas, consider the two-input case with triangular shaped FS. The denominator then equals the polynomial $p(u_1, u_2)$,

$$
\sum_{i=1}^{R} \mu_{A_1}(u_1) \mu_{A_2}(u_2) \ldots \mu_{A_m}(u_m) b_i = \sum_{i=1}^{R} a_i^1 u_1 a_i^2 u_2 b_i = p(u_1, u_2) \quad (A-13)
$$

with $a_i^1$ and $a_i^2$ two constants that denote the slopes of the triangular FS used in the $i$-th rule, and $b_i$ a constant that denotes the area of the output set in the $i$-th rule. The general case is written as
where \( f_i(u_1, u_2, ..., u_m) \) has constant parameters for each rule, and which is a special case of a Takagi-Sugeno FLS.

### A.4 Proof of Lemma 5.2 on page 120: SISO Fuzzy Logic System as NLq

Under Assumption 5.1 on page 115 and using (5-31) on page 119, equation (5-25) becomes

\[
\begin{aligned}
\begin{cases}
  s_{k+1} = As_k + Bx_k \\
  y_k = (c^b)^T \Phi (Cs_k + Dx_k)
\end{cases}
\end{aligned}
\tag{A-15}
\]

and thus

\[
\begin{aligned}
\begin{cases}
  s_{k+1} = \text{lin}(V_1 \text{lin}(V_2 p_k + B_2 w_k) + B_1 w_k) \\
  y_k = \text{lin}(W_1 \text{sat}(W_2 p_k + D_2 w_k) + D_1 w_k)
\end{cases}
\end{aligned}
\tag{A-16}
\]

with \( p_k = s_k, w_k = [x_k 1]^T \),

\[
V_1 = \begin{bmatrix} A & B \end{bmatrix}, \quad V_2 = \begin{bmatrix} I \\ 0 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 \\ I & 0 \end{bmatrix},
\tag{A-17}
\]

\[
W_1 = (c^b)^T \frac{1}{2} [I_r \ I_r], \quad W_2 = \begin{bmatrix} 2S^L C \\ -2S^R C \end{bmatrix}, \quad D_1 = \begin{bmatrix} 0 & 0 \end{bmatrix},
\tag{A-18}
\]

and

\[
D_2 = \begin{bmatrix} 2S^L C & 1 + 2(K^L - c^A) \Theta S^L \\ -2S^R C & 1 + 2(K^R + c^A) \Theta S^R \end{bmatrix}.
\tag{A-19}
\]

The sector bounded functions are \( \sigma_1 = \text{lin}(\bullet) \) and \( \sigma_2 = \text{sat}(\bullet) \). \( \Lambda_1 \) and \( \Gamma_1 \) are two identical matrices, while \( \Lambda_2 \) and \( \Gamma_2 \) are diagonal matrices with diagonal elements \( \text{sat}(\omega_i)/\omega_i \) and \( \text{lin}(\omega_i)/\omega_i \) that satisfy the sector boundedness property (5-3).
A.5 Proof of Lemma 5.4 on page 131, 2-norm of a blockdiagonal matrix

The square matrix $A \in \mathbb{R}^{n \times n}$ can be decomposed as $A = U \Sigma V^T$ where $\Sigma = \{ \sigma_1(A), \sigma_2(A), \ldots, \sigma_n(A) \}$ contains the ordered singular values of $A$. The 2-norm of $A$ is defined as

$$\|A\|_2 = \max_{1 \leq k \leq n} \sigma_k(A). \quad (A-20)$$

In the case that $A$ is defined as a blockdiagonal matrix

$$A = \text{blockdiag}(A_1, A_2, \ldots, A_m) \quad (A-21)$$

the $\Sigma$ matrix contains the ordered singular values of all individual $A_j$ matrices, i.e. $\Sigma = \{ \ldots, \sigma_i(A_j), \ldots \}$ with $j = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, n_j$. This also applies if two rows or columns, containing two matrices $A_{j_1}$ and $A_{j_2}$, are randomly swapped. The 2-norm of the $A$ matrix then equals

$$\|A\|_2 = \max_{1 \leq j \leq m} \left( \max_{1 \leq i \leq n_j} \sigma_i(A_j) \right). \quad (A-22)$$

Thus

$$\|A\|_2 = \max(\|A_1\|_2, \|A_2\|_2, \ldots, \|A_m\|_2), \quad (A-23)$$

or

$$\|A\|_2^{\geq} \geq \|A_1\|_2 \cdot \|A_2\|_2 \cdot \ldots \cdot \|A_m\|_2, \quad (A-24)$$

which proves the lemma. \qed

A.6 Proof of Theorem 5.4 on page 132, Global asymptotic stability of autonomous NLqp systems [Diagonal Scaling]

Define the Lyapunov function

$$L_k = \|\Delta p_k\|_2. \quad (A-25)$$

For the autonomous system (5-67) on page 130, $L_{k+1}$ is calculated as

$$L_{k+1} = \|\Delta p_{k+1}\|_2 = \|\Delta V_{j=1}^p \{ \Gamma_j(V_j^{[j]} \Gamma_j^{[j]}(V_j^{[j]} \ldots \Gamma_j^{[j]}(V_j^{[j]} p_j)))\} \|_2. \quad (A-26)$$
Insert \((\Delta[j])^{-1}\Delta[j]\) before each \(V^{[j]}\) for \(i = 2...q - 1\) and insert \(\Delta^{-1}\Delta\) after each \(V_q^{[j]}\). All \(\Gamma_i^{[j]}\) and \(\Delta_i^{[j]}\) are real diagonal matrices, such that

\[
\Gamma_i^{[j]}(\Delta_i^{[j]})^{-1} = (\Delta_i^{[j]})^{-1}\Gamma_i^{[j]}.
\]  
(A-27)

Equation (A-26) can be written as

\[
L_{k+1} \leq \|\Delta V\|_2 \prod_{j=1}^{p} \left\{ \|\Gamma_j^{[j]}\|_2 \|V^{[j]}(\Delta_j^{[j]})^{-1}\|_2 \|\Gamma_j^{[j]}\|_2 \right\}.
\]  
(A-28)

With the property that \(\|\Gamma^{[j]}\|_2 \leq 1\) this becomes

\[
L_{k+1} \leq \zeta L_k^p
\]  
(A-29)

with

\[
\zeta = \|\Delta V\|_2 \prod_{j=1}^{p} \left\{ \|V^{[j]}(\Delta_j^{[j]})^{-1}\|_2 \|\Delta_j^{[j]}V^{[j]}(\Delta_j^{[j]})^{-1}\|_2 \|\Delta_j^{[j]}V^{[j]}(\Delta_j^{[j]})^{-1}\|_2 \right\}.
\]  
(A-30)

A sufficient condition for the system (A-29) to be globally asymptotically stable is that

\[
\zeta < 1 \text{ and } L_0 \leq 1.
\]  
(A-31)

The condition \(L_0 \leq 1\) is met if the initial \(\Delta\) is chosen such that Assumption 5.6 on page 131 is true. With the definitions (5-77), (5-78) and Lemma 5.4 on page 131, it is possible to write the relaxation factor \(\zeta\) in equation (A-30) as

\[
\zeta \leq \|\Delta_{tot} V_{tot} \Delta_{tot}^{-1}\|_2^{p+1}
\]  
(A-32)

such that a sufficient condition for global asymptotic stability of the autonomous NL_q^p system (5-67) is that nonzero diagonal matrices \(\Delta\) and \(\Delta_i^{[j]}\) are found, such that

\[
\|\Delta_{tot} V_{tot} \Delta_{tot}^{-1}\|_2^{q+1} < 1
\]  
(A-33)

which can be reduced to the condition

\[
\|\Delta_{tot} V_{tot} \Delta_{tot}^{-1}\|_2 < 1.
\]  
(A-34)

This completes the proof.
A.7 Proof of Theorem 5.5 on page 133, Global asymptotic stability of autonomous NLqp systems [Condition Number Factor]

Define the Lyapunov function

\[ L_k = \|Pp_k\|_2. \]  \hspace{1cm} (A-35)

The proof follows the same lines as the proof of Theorem 5.4 such that

\[ L_{k+1} = \left\| PP_{k+1} \right\|_2 = \left\| PV \prod_{j=1}^{p} \left\{ \Gamma^{[j]} \Gamma^{[j]} \Gamma^{[j]} V^{[j]} \ldots \Gamma^{[j]} V^{[j]} P_{p}^{-1} \right\} \right\|_2. \]  \hspace{1cm} (A-36)

Insert \((P^{[j]}_i)^{-1} P^{[j]}_i)\) before each \(V^{[j]}_i\) and each \(\Gamma^{[j]}_i\) for \(i = 2 \ldots q - 1\) and insert \(P^{-1} P\) after each \(V^{[j]}_q\). (A-36) becomes

\[ L_{k+1} = \left\| PV \prod_{j=1}^{p} \left\{ \Gamma^{[j]}_i V^{[j]}_i (P^{[j]}_2)^{-1} P^{[j]}_2 \Gamma^{[j]}_2 (P^{[j]}_2)^{-1} P^{[j]}_2 V^{[j]}_2 \ldots \Gamma^{[j]}_q (P^{[j]}_q)^{-1} P^{[j]}_q V^{[j]}_q P^{-1} P_{p} \right\} \right\|_2 \]

which can be written as

\[ L_{k+1} \leq \|PV\|_2 \cdot \prod_{j=1}^{p} \left\{ \prod_{i=2}^{q} \left[ \|P^{[j]}_i\|_2 \|\Gamma^{[j]}_i \|_2 \|V^{[j]}_i\|_2 \right] \right\} \prod_{i=1}^{q} \left[ \|P^{[j]}_i\|_2 \|\Gamma^{[j]}_i \|_2 \|V^{[j]}_i\|_2 \right] L_k \]  \hspace{1cm} (A-38)

or

\[ L_{k+1} \leq \|PV\|_2 \cdot \prod_{j=1}^{p} \left\{ \prod_{i=2}^{q} \left[ \kappa(P^{[j]}_i) \right] \prod_{i=1}^{q} \left[ \|P^{[j]}_i\|_2 \|\Gamma^{[j]}_i \|_2 \|V^{[j]}_i\|_2 \right] \right\} L_k \]  \hspace{1cm} (A-39)

because \(\|\Gamma^{[j]}_i\|_2 \leq 1\). Using Assumption 5.7 a sufficient condition for (A-39) to be stable, is that \(\zeta < 1\) with

\[ \zeta = \|PV\|_2 \cdot \prod_{j=1}^{p} \left\{ \prod_{i=1}^{q} \left[ \|P^{[j]}_i\|_2 \|\Gamma^{[j]}_i \|_2 \|V^{[j]}_i\|_2 \right] \right\} \cdot \prod_{j=1}^{p} \left\{ \prod_{i=2}^{q} \left[ \kappa(P^{[j]}_i) \right] \right\} < 1. \]  \hspace{1cm} (A-40)

With the definitions (5-78), (5-82) and Lemma 5.4, this condition can be written as
\[ \| P_{\text{tot}} V_{\text{tot}} P_{\text{tot}}^{-1} \|_2^{pq+1} \prod_{j=1}^{p} \prod_{i=2}^{q} [\kappa(P^{[j]}_{i})] < 1 \] (A-41)

which completes the proof.

A.8 Proof of Theorem 5.6 on page 134, I/O stability of NL\text{qp} systems
[Diagonal Scaling]

Define the Lyapunov function
\[ L_k = \left\| \Delta \begin{bmatrix} p_k \\ w_k \end{bmatrix} \right\|_2. \] (A-42)

Using Lemma 5.3, \( L_{k+1} \) is calculated as
\[ L_{k+1} = \left\| \Delta \left[ \begin{bmatrix} p_{k+1} \\ w_{k+1} \end{bmatrix} \right] \right\|_2 = \left\| \Delta T \prod_{j=1}^{p} \left\{ \Omega^{[j]} \left( T^{[j]} \Omega^{[j]}_2 \left( T^{[j]}_2 \cdots \Omega^{[j]}_q \left( T^{[j]}_q \begin{bmatrix} p_k \\ w_k \end{bmatrix} \right) \right) \right\} \right\|_2. \] (A-43)

The proof follows using the same lines on the \( \Delta \) matrices as in the proof of Theorem 5.4.

A.9 Proof of Theorem 5.7 on page 135, I/O stability of NL\text{qp} systems
[Condition Number Factor]

Define the Lyapunov function
\[ L_k = \left\| P \begin{bmatrix} p_k \\ w_k \end{bmatrix} \right\|_2. \] (A-44)

Using Lemma 5.3, \( L_{k+1} \) is calculated as
\[ L_{k+1} = \left\| P \begin{bmatrix} p_{k+1} \\ w_{k+1} \end{bmatrix} \right\|_2 = \left\| PT \prod_{j=1}^{p} \left\{ \Omega^{[j]} \left( T^{[j]} \Omega^{[j]}_2 \left( T^{[j]}_2 \cdots \Omega^{[j]}_q \left( T^{[j]}_q \begin{bmatrix} p_k \\ w_k \end{bmatrix} \right) \right) \right\} \right\|_2. \] (A-45)

The proof follows using the same lines on the \( P \) matrices as in the proof of Theorem 5.5.
A.10 Proof of Theorem 5.8 on page 137

The membership functions $\mu^A_{(x_j = A_{j,i})}$ can be written as

$$
\mu^A_{(x_j = A_{j,i})} = \frac{1}{2} \left\{ sat \left( \frac{x_j - c^A_{j,i} + k^L_{j,i}}{\sigma^L_{j,i}} + 1 \right) + sat \left( -\frac{x_j - c^A_{j,i} - k^R_{j,i}}{\sigma^R_{j,i}} + 1 \right) \right\} \tag{A-46}
$$

with $c^A_{j,i}, k^L_{j,i}, k^R_{j,i}, \sigma^L_{j,i}$ and $\sigma^R_{j,i}$ the centres of the input Fuzzy Sets, the positive definite shift vectors and the slope vectors respectively. These vectors are defined, based on the definitions (5-27) to (5-29) on page 119 for each input set $j$. Using straightforward calculation it is possible to write the fuzzy system (5-99) as

$$
FLS(x) = (e^B)^T \bigotimes_{j=1}^p \frac{1}{2} \begin{bmatrix} I_r & I_j \end{bmatrix} sat \left( 2 \begin{bmatrix} S^L_j \\ -S^R_j \end{bmatrix} x_j + 1 + 2 \left( K^L_j - c^A_j \right) \otimes S^L_j \right) \tag{A-47}
$$

which corresponds to the NL$_2^p$ system

$$
FLS(x) = V \bigotimes_{j=1}^p \{ \Gamma^{[j]}(V^{[j]} \Gamma^{[j]}(V^{[j]} x_j + B^{[j]})) \} \tag{A-48}
$$

or

$$
FLS(x) = V \bigotimes_{j=1}^p \{ lin(V^{[j]} sat(V^{[j]} x_j + B^{[j]})) \} \tag{A-49}
$$

with

$\quad V = (e^B)^T, \quad V^{[j]} = \frac{1}{2} \begin{bmatrix} I_r & I_j \end{bmatrix}, \quad V^{[j]} = \begin{bmatrix} 2S^L_j \\ -2S^R_j \end{bmatrix}, \quad B^{[j]} = 0 \tag{A-50}$

and

$$
B^{[j]} = \begin{bmatrix} 1 + 2((K^L_j - c^A_j) \otimes S^L_j) \\ 1 + 2((K^R_j + c^A_j) \otimes S^R_j) \end{bmatrix} \tag{A-51}
$$

which completes the proof. \qed
A.11 Proof of Theorem 6.1 on page 148: training of a NN with noisy inputs and a linearised cost function

To study the stochastic behaviour of the linearised cost function, we study the expected value

$$E\{ \overline{C}_{LS}(\hat{u}_k, y^*_k) \} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \left( y^*_k - f_{NN}(u^*_k, \Theta) - n_{u,k} \frac{\partial f_{NN}(u^*_k, \Theta)}{\partial u^*_k} \right)^2 \right\}.$$  \hspace{1cm} (A-52)

Since

$$E\left\{ n_{u,k} \frac{\partial f_{NN}(u^*_k, \Theta)}{\partial u^*_k} (y^*_k - f_{NN}(u^*_k, \Theta)) \right\} = 0 \hspace{1cm} (A-53)$$

and

$$E\left\{ \left( n_{u,k} \frac{\partial f_{NN}(u^*_k, \Theta)}{\partial u^*_k} \right)^2 \right\} = \sigma_u^2 \left( \frac{\partial f_{NN}(u^*_k, \Theta)}{\partial u^*_k} \right)^2 \hspace{1cm} (A-54)$$

this cost function becomes after some calculations

$$E\{ \overline{C}_{LS}(\hat{u}_k, y^*_k) \} = \frac{1}{N} \sum_{k=1}^{N} \left[ (y^*_k - f_{NN}(u^*_k, \Theta))^2 + \sigma_u^2 \left( \frac{\partial f_{NN}(u^*_k, \Theta)}{\partial u^*_k} \right)^2 \right]. \hspace{1cm} (A-55)$$

The term (A-54) is not $\Theta$-independent. Therefore, the minimizer

$$\arg\min_{\Theta} \left( E\{ \overline{C}_{LS}(\hat{u}_k, y^*_k) \} \right) \hspace{1cm} (A-56)$$

is not guaranteed to equal $\Theta^*$. This means that the LS estimates are inconsistent in the presence of input noise. In the case where the $n_{u,k}$ become larger, higher order terms of the Taylor expansion (6-15) on page 148 must be taken into account. Those terms are also $\Theta$-dependent and the errors only deteriorate.
A.12 Proof of Theorem 6.2 on page 149: training of a NN with noisy inputs and noisy outputs and a linearised cost function

Using equation (6-6) on page 146, the expected value of the linearized cost function for noisy inputs and outputs equals

\[
E\{C_{LS}(\hat{u}_k, \hat{y}_k)\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \left( \frac{1}{\sigma^2_y} \left( y_k^* + n_{y,k} - f_{NN}(u_k^*, \Theta) - n_{u,k} \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*} \right)^2 \right) \right\}.
\] (A-57)

Since

\[
E\left\{ n_{y,k} + n_{u,k} \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*} \right\} = 0,
\] (A-58)

\[
E\{ (n_{y,k})^2 \} = \sigma^2_y,
\] (A-59)

\[
E\left\{ n_{u,k} n_{y,k} \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*} \right\} = \sigma^2_{u,y} \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*}
\] (A-60)

and using equation (A-54) this cost function becomes after some calculations

\[
E\{C_{LS}(\hat{u}_k, \hat{y}_k)\} = \frac{1}{N} \sum_{k=1}^{N} \left( y_k^* - f_{NN}(u_k^*, \Theta) \right)^2 + \sigma^2_u \left( \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*} \right)^2 + \sigma^2_y
\] (A-61)

\[
-2\sigma^2_{u,y} \frac{\partial(f_{NN}(u_k^*, \Theta))}{\partial u_k^*}
\]

The terms (A-54) and (A-60) that are present in equation (A-61) are not \( \Theta \)-independent. Therefore, the minimizer

\[
\arg\min_\Theta \ E\{C_{LS}(\hat{u}_k, \hat{y}_k)\}
\] (A-62)

is not guaranteed to equal \( \Theta^* \). This means that the LS estimates are inconsistent in the presence of input and output noise.

\[\Box\]
A.13 Proof of Lemma 6.1 on page 151

The estimated input values $u_k$ are found by minimizing the linearised EIV cost function (6-25) on page 150

$$
\bar{C}_{EIV}(\hat{u}_k, \hat{y}_k) = \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{(\hat{y}_k - f_{NN}(u^*_k, \theta) - \epsilon_{u,k} \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k})^2}{\sigma^2_{y,k}} + \frac{(\hat{u}_k - u_k)^2}{\sigma^2_{u,k}} \right] \tag{A-63}
$$

w.r.t. $u_k$. Thus,

$$
\frac{\partial \bar{C}_{EIV}(\hat{u}_k, \hat{y}_k)}{\partial u_k} = 0. \tag{A-64}
$$

Knowing that $\epsilon_{u,k} = u_k - u^*_k$ this gives

$$
\frac{2}{\sigma^2_{y,k}} (\hat{y}_k - f_{NN}(u^*_k, \theta) - (u_k - u^*_k) \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k}) \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k} + \frac{2}{\sigma^2_{u,k}} (\hat{u}_k - u_k) = 0 \tag{A-65}
$$

or

$$
u_k = \frac{\sigma^2_{u,k} \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k} (\hat{y}_k + u^*_k \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k} - f_{NN}(u^*_k, \theta)) + \sigma^2_{y,k} \hat{u}_k}{\sigma^2_{y,k} + \sigma^2_{u,k} \left(\frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k}\right)^2}. \tag{A-66}
$$

With this expression for $u_k$ the cost function (A-63) can be written as

$$
\bar{C}_{EIV}(\hat{u}_k, \hat{y}_k) = \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{(\hat{y}_k - f_{NN}(u^*_k, \theta) + (u_k - u^*_k) \frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k})^2}{\sigma^2_{y,k} + \sigma^2_{u,k} \left(\frac{\partial f_{NN}(u^*_k, \theta)}{\partial u^*_k}\right)^2} \right]. \tag{A-67}
$$

In order to assess the behaviour of this cost function for large $N$ the expected value $E\{\bar{C}_{EIV}(\hat{u}_k, \hat{y}_k)\}$ is calculated. For this the $\hat{u}_k$ and $\hat{y}_k$ are replaced by the noise representations given in (6-6):

$$
\hat{u}_k = u^*_k + n_{u,k} \text{ and } \hat{y}_k = y^*_k + n_{y,k}. \tag{A-68}
$$
Since
\[ E \left\{ n_{y,k} \left( y_k^* - f_{NN}(u_k^*, \theta) + (u_k^* - \hat{u}_k) \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} \right) \right\} = 0, \quad (A-69) \]
\[ E \left\{ n_{u,k} \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} (y_k^* - f_{NN}(u_k^*, \theta)) \right\} = 0 \quad (A-70) \]
and
\[ E \left\{ n_{y,k}^2 + n_{u,k}^2 \left( \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} \right)^2 \right\} = \sigma_{y,k}^2 + \sigma_{u,k}^2 \left( \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} \right)^2 \quad (A-71) \]
the expected value of equation (A-67) reduces to
\[ E \{ \tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k) \} = M + \frac{M}{N} \sum_{k=1}^{N} \left[ \frac{(y_k^* - f_{NN}(u_k^*, \theta))^2}{\sigma_{y,k}^2 + \sigma_{u,k}^2 \left( \frac{\partial f_{NN}(u_k^*, \theta)}{\partial u_k^*} \right)^2} \right]. \quad (A-72) \]
It is obvious that the \( M \) term is not \( \theta \) dependent. The term in the right hand side is minimal in the true \( \theta^* \) parameters, such that
\[ \arg \min_{\theta} \left( E \{ \tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k) \} \right) = \theta^* \quad (A-73) \]
which proves the lemma for the linearised NN. \( \square \)

**A.14 Proof of Theorem 6.3 on page 151: training of a NN with noisy inputs and outputs, and a linearised cost function**

In [189] it is proven that under Assumption 6.1 on page 146
\[ \text{a.s.} \lim_{N \to \infty} [ \tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k) - E \{ \tilde{C}_{EIV}(\hat{u}_k, \hat{y}_k) \} ] = 0 \quad (A-74) \]
uniformly in \( \theta \). Thus
With Assumption 6.4 and Lemma 6.1 on page 151 this implies that
\[
\text{a.s.lim}_{N \to \infty} \left[ \arg\min_{\theta} \left( \overline{C}_{EIV}(\hat{u}_k, \hat{y}_k) \right) - \arg\min_{\theta} \left( E\{\overline{C}_{EIV}(\hat{u}_k, \hat{y}_k)\} \right) \right] = 0 \quad [195]. \tag{A-75}
\]
which proves strong consistency of the NN parameters [195].

\[\square\]

### A.15 Proof of Theorem 7.1 on page 187: effects of output noise on BB modeling, SISO case

Assume only noise on the output samples. Using Lemma 7.1 on page 186 the expected value of the LS cost function (7-13) on page 186 becomes
\[
E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_k, \theta) - y_k \right)^2 \right\}. \tag{A-77}
\]

Put (7-12) in (A-77). Since
\[
E\left\{ \frac{1}{R} \sum_{r=1}^{R} (n_{k,r}^2) \right\} = \sigma^2_{k,y} \tag{A-78}
\]
and
\[
E\left\{ \frac{-2}{R} \sum_{r=1}^{R} n_{k,r}^2 \left( f_{BB}(u_k, \theta) - y_k \right) \right\} = 0 \tag{A-79}
\]
the expected value (A-77) becomes
\[
E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \left[ \left( f_{BB}(u_k, \theta) - y_k \right)^2 + \sigma^2_{k,y} \right] \right\}. \tag{A-80}
\]
The $\sigma^2_{k,y}$ term in this equation is independent of the BB parameters $\theta$. The minimizer of (A-80) for $\theta$ is therefore exactly the same as the minimizer of (7-10) such that adding noise to the output samples doesn’t change the BB parameters, i.e. has no regularization effect.

A.16 Proof of Theorem 7.2 on page 187: effects of input noise on BB modeling, SISO case

Assume only noise on the input samples. Using Lemma 7.1 on page 186 the expected value of the LS cost function (7-13) on page 186 becomes

$$
E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_k^r, \theta) - y_k \right)^2 \right\}.
$$

(A-81)

Put (7-12) in (A-81). The BB model is assumed continuous (Kolmogorov’s theorem, see section A.1) with continuous higher order derivatives such that we can replace the BB model by its Taylor expansion

$$
f_{BB}(u_k^r, \theta) = f_{BB}(u_k, \theta) + \sum_{l=1}^{\infty} \frac{(n_{k,u}^r)^l}{(l)!} \frac{\partial f_{BB}(u_k, \theta)}{\partial u_k^l}
$$

(A-82)

and put this expansion into the cost function. The calculation of the expectation value becomes

$$
E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left[ f_{BB}(u_k, \theta) + \sum_{l=1}^{\infty} \frac{(n_{k,u}^r)^l}{(l)!} \frac{\partial f_{BB}(u_k, \theta)}{\partial u_k^l} - y_k \right]^2 \right\}.
$$

(A-83)

After computing the summation terms and knowing that

$$
E\{(n_{k,u}^r)^{2l+1}\} = 0
$$

(A-84)

and (Stuart et al. [200])

$$
E\{(n_{k,u}^r)^{2l}\} = \frac{1}{\sigma_{k,u}^2/2\pi} \int_{-\infty}^{+\infty} (n_{k,u}^r)^{2l} \exp\left(-\frac{1}{2\sigma_{k,u}^2} \frac{(n_{k,u}^r)^2}{2\sigma_{k,u}^2} \right) d(n_{k,u}^r) = \frac{\sigma_{k,u}^{2l}(2l)!}{2^l l!}
$$

(A-85)
it is possible to write (A-83) as

\[
E\{C_{LS}\} = \frac{1}{N} \sum_{k=1}^{N} \left( f_{BB}(u_k, \theta) - y_k \right)^2 + \frac{1}{N} \sum_{k=1}^{N} \left( 2\mu_k \left( f_{BB}(u_k, \theta) - y_k \right) + \epsilon_k^2 \right) \tag{A-86}
\]

with

\[
\mu_k = \sum_{l=1}^{\infty} \sigma_{k,u}^{2l} \frac{\partial^{2l} f_{BB}(u_k, \theta)}{(\partial u_k)^{2l}} \tag{A-87}
\]

and \(\epsilon_k^2\) is the expectation of the square term

\[
\epsilon_k^2 = E\left\{ \left( \sum_{l=1}^{\infty} \frac{(n_{k,u})^l}{(l)!} \frac{\partial f_{BB}(u_k, \theta)}{(\partial u_k)^l} \right)^2 \right\} \tag{A-88}
\]

which can be decomposed as

\[
\epsilon_k^2 = \sum_{l=1}^{\infty} \sigma_{k,u}^{2l} \frac{(2l)!}{2((l!)^3} \left( \frac{\partial f_{BB}(u_k, \theta)}{(\partial u_k)^l} \right)^2 + \sum_{l=2l=1}^{\infty} \sigma_{k,u}^{2l} \frac{(2l)!}{2((l!)^3} \frac{\partial f_{BB}(u_k, \theta) \partial^{(2l-1)} f_{BB}(u_k, \theta)}{(\partial u_k)^{2l-1}} \tag{A-89}
\]

The term \(\sum_{k=1}^{N} \mu_k (f_{BB}(u_k, \theta) - y_k)\) in equation (A-86) contains values that are spread around zero due to the noise on \(y_k\), and is of the order \(O(\sqrt{N})\), while the \(\sum_{k=1}^{N} \epsilon_k^2\) term is of order \(O(N)\). For large \(N\) it is possible to skip the terms in \(\mu_k\). This was also concluded for SISO Neural Networks by An [13] and Bishop [26] for the case \(N \rightarrow \infty\). In the analysis given in this thesis \(N\) is assumed large enough, so that the analysis is restricted to

\[
E\{C_{LS}\} \approx \frac{1}{N} \sum_{k=1}^{N} \left[ (f_{BB}(u_k, \theta) - y_k)^2 + \epsilon_k^2 \right] \tag{A-90}
\]

A closer view on (A-89) shows that \(\epsilon_k^2\) is a summation of variance-derivative products, such that it acts as a regularizer term that adds higher order derivatives of the BB model into the cost function. \(\epsilon_k^2\) has become a regularizer term that is dependent on the noise variance in a very complex way.
A.17 Proof of Theorem 7.3 on page 189: effects of input and output noise on BB modeling, SISO case

The LS cost function (7-13) on page 186 becomes

$$E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_{k}^{[r]}, \theta) - y_{k}^{[r]} \right)^{2} \right\}. \quad (A-91)$$

Put (7-12) in (A-91). Following the same lines as the proof of Theorem 7.1 the expected value becomes

$$E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \left[ \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_{k}^{[r]}, \theta) - y_{k}^{[r]} \right)^{2} \right] \right\} + \frac{1}{N} \sum_{k=1}^{N} \sigma_{k,y}^{2}. \quad (A-92)$$

The $\sigma_{k,y}^{2}$ terms are $\theta$-independent, such that the minimizer of the cost function (A-92) is the same as the minimizer of the cost function

$$E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB}(u_{k}^{[r]}, \theta) - y_{k}^{[r]} \right)^{2} \right\}. \quad (A-93)$$

This is the same cost function as the cost (A-81) of Theorem 7.2 and we find ourselves in the situation as if there was only added input noise.

\[\square\]

A.18 Proof of Theorem 7.4 on page 195: effects of output noise on BB modeling, MIMO case

Assume only noise on the output samples. Using Lemma 7.2 on page 195 the expected value of the LS cost function (7-32) on page 194 becomes

$$E\{C_{LS}\} = E\left\{ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{R} \sum_{i=1}^{n} \left( f_{BB,i}(u_{k}, \theta) - y_{k}^{[r]} \right)^{2} \right\}. \quad (A-94)$$

Put (7-31) on page 194 in (A-94). Since
\[ E \left[ \frac{1}{R} \sum_{r=1}^{R} (n_{k,i,y}^{[r]})^2 \right] = \sigma_{k,i,y}^2 \]  
(A-95)

and

\[ E \left[ \frac{-2}{R} \sum_{r=1}^{R} n_{k,i,y}^{[r]} \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right) \right] = 0 \]  
(A-96)

the expected value (A-94) becomes

\[ E\{C_{LS}\} = E \left[ \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right)^2 \right] + \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{n} \sigma_{k,i,y}^2. \]  
(A-97)

The \( \sigma_{k,i,y}^2 \) term in this equation is independent of the BB parameters \( \theta \). The minimizer of (A-97) for \( \theta \) is therefore exactly the same as the minimizer of (7-28) such that adding noise to the output samples doesn’t change the BB parameters. \( \square \)

### A.19 Proof of Theorem 7.5 on page 195: effects of input noise on BB modeling, MIMO case

Assume only noise on the input samples. Using Lemma 7.2 on page 195 the expected value of the LS cost function (7-32) on page 194 becomes

\[ E\{C_{LS}\} = E \left[ \frac{1}{K} \sum_{k=1}^{K} \frac{1}{R} \sum_{r=1}^{R} \sum_{i=1}^{n} \left( f_{BB,i}(u_k^{[r]}, \theta) - y_{k,i} \right)^2 \right]. \]  
(A-98)

Put (7-30) in (A-98). The BB model and its higher order derivatives are assumed to be continuous such that we can replace the BB model by its Taylor expansion and put this expansion into the cost function. Knowing that

\[ E\{(n_{k,i,y}^{[r]})^{2l+1}\} = 0 \]  
(A-99)

and (Stuart et al. [200])
it is possible (after lengthy calculations) to write (A-98) as

$$E\{ C_{LS} \} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right)^2 + \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left( 2 \mu_k, i \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right) + \varepsilon_{k,i}^2 \right)$$

(A-101)

where \( \varepsilon_{k,i}^2 \) is again the expectation of the square term of the Taylor expansion, which is calculated analogous to the calculation of the square term (A-89) on page 249. The term \( \sum_{k=1}^{N} \mu_k, i \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right) \) in equation (A-101) contains values that are spread around zero and is of the order \( O(\sqrt{N}) \), while the \( \sum_{k=1}^{N} \varepsilon_{k,i}^2 \) term is of order \( O(N) \). For large \( N \) it is possible to skip the terms in \( \mu_k, i \), just as it was done in the SISO case. The analysis is restricted to

$$E\{ C_{LS} \} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left[ \left( f_{BB,i}(u_k, \theta) - y_{k,i} \right)^2 + \varepsilon_{k,i}^2 \right].$$

(A-102)

The regularizer term \( \varepsilon_{k,i}^2 \) is a summation of variance-derivative products, such that it acts as a regularizer term that adds higher order derivatives of the BB model into the cost function.

A.20 Proof of Theorem 7.6 on page 196: effects of input and output noise on BB modeling, MIMO case

The LS cost function (7-32) on page 194 becomes

$$E\{ C_{LS} \} = E \left[ \frac{1}{N} \sum_{k=1}^{N} \frac{1}{R} \sum_{r=1}^{n} \sum_{i=1}^{n} \left( f_{BB,i}(u_k^{[r]}, \theta) - y_k^{[r]} \right)^2 \right].$$

(A-103)

Put (7-31) in (A-103). Following the same lines as the proof of Theorem 7.4 the expected value becomes
The terms are independent, such that the minimizer of the cost function \((A-104)\) is the same as the minimizer of the cost function \((A-105)\). This is the same cost function as the cost \((A-98)\) of Theorem 7.5 and we find ourselves in the situation as if there was only added input noise.

\[
E\{C_{LS}\} = E\left[\frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \left( \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB, i}^{[r]}(u_k^{[r]}, \theta) - y_{k,i} \right)^2 \right) \right] + \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \sigma_{k,i,y}^2. \tag{A-104}
\]

The \(\sigma_{k,i,y}^2\) terms are \(\theta\)-independent, such that the minimizer of the cost function \((A-104)\) is the same as the minimizer of the cost function

\[
E\{C_{LS}\} = E\left[\frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \frac{1}{R} \sum_{r=1}^{R} \left( f_{BB, i}^{[r]}(u_k^{[r]}, \theta) - y_{k,i} \right)^2 \right]. \tag{A-105}
\]

This is the same cost function as the cost \((A-98)\) of Theorem 7.5 and we find ourselves in the situation as if there was only added input noise. \(\square\)
<table>
<thead>
<tr>
<th><strong>A</strong></th>
<th><strong>B</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>adaline</td>
<td>backpropagation</td>
<td>Cauchy machine</td>
</tr>
<tr>
<td>adaptation</td>
<td>backpropagation through time</td>
<td>centre of area</td>
</tr>
<tr>
<td>Adaptive Resonance Theory</td>
<td>BAM</td>
<td>centre of gravity</td>
</tr>
<tr>
<td>AIC</td>
<td>batch</td>
<td>centre of sums</td>
</tr>
<tr>
<td>Akaike</td>
<td>batch optimization</td>
<td>character recognition</td>
</tr>
<tr>
<td>approximators</td>
<td>Bayes</td>
<td>clustering</td>
</tr>
<tr>
<td>ART map</td>
<td>Bayesian</td>
<td>C-means</td>
</tr>
<tr>
<td>Artificial Intelligence</td>
<td>bayesian learning</td>
<td>complexity penalty</td>
</tr>
<tr>
<td></td>
<td>bell-shape</td>
<td>compression</td>
</tr>
<tr>
<td></td>
<td>bias</td>
<td>concentration</td>
</tr>
<tr>
<td></td>
<td>bias variance trade-off</td>
<td>condition number</td>
</tr>
<tr>
<td></td>
<td>bias-adjusted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>biasing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bidirectional associative memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>black box</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boltzmann machine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPTT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>brain-state-in-a-box</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B-spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B-spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Page Numbers</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>condition number product</td>
<td>114, 123, 128, 135</td>
<td></td>
</tr>
<tr>
<td>consequence</td>
<td>86, 89, 93</td>
<td></td>
</tr>
<tr>
<td>conservatism</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>consistency</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>control</td>
<td>65, 80, 100</td>
<td></td>
</tr>
<tr>
<td>controllability</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>convergence</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>convex</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>cost function</td>
<td>12, 13, 26</td>
<td></td>
</tr>
<tr>
<td>counterpropagation network</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>crisp</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>cross-validation</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>curse of dimensionality</td>
<td>92, 105</td>
<td></td>
</tr>
<tr>
<td>data base</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>defuzzification</td>
<td>88, 106</td>
<td></td>
</tr>
<tr>
<td>domain of interest</td>
<td>15, 16, 71, 77, 81, 92, 178</td>
<td></td>
</tr>
<tr>
<td>dot product</td>
<td>xiv</td>
<td></td>
</tr>
<tr>
<td>double backpropagation</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>DP</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>drastic product</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>dynamic backpropagation</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>dynamic systems</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>errors-in-variables</td>
<td>29, 41, 143, 149</td>
<td></td>
</tr>
<tr>
<td>Euler-Lagrange equation</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>expectation</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>expert knowledge</td>
<td>93, 98, 103</td>
<td></td>
</tr>
<tr>
<td>external dynamics</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>extrapolation</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>feature space mapping</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>feedforward</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>final prediction error</td>
<td>22, 24</td>
<td></td>
</tr>
<tr>
<td>FIR</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>FL</td>
<td>75, 103</td>
<td></td>
</tr>
<tr>
<td>FL modeling</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>flexible hypercube</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>FLS</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>FPE</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>FPEB</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Frobenius norm</td>
<td>xv</td>
<td></td>
</tr>
<tr>
<td>full rule base</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>functional relationship</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>fuzzification</td>
<td>77, 79, 92</td>
<td></td>
</tr>
<tr>
<td>fuzzy controller</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>fuzzy hedge</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>fuzzy inference</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>fuzzy logic</td>
<td>12, 19, 60, 75, 91</td>
<td></td>
</tr>
<tr>
<td>fuzzy model</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>fuzzy operators</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>fuzzy set</td>
<td>76, 78, 82, 86, 88, 90</td>
<td></td>
</tr>
</tbody>
</table>
G

gain scheduling 100
Gaussian machine 60
Gaussian set 104
Gauss-Newton 41
generalization 16, 46, 71, 177, 178, 183, 191
generalized additive FS 20
generalized cross-validation 22
generalized prediction error 24
global minimum 147, 179, 185
GPE 24
gradient 40
gradient descent 40, 97, 154
grid-type clustering 105
growing 24
Gustafson-Kessel 94, 99, 104

H

Hadamar-Shur product xiv
Hamming network 60
Hebb nets 52
Hebbian learning 43
Hessian 41
HIP 25
incremental optimization 70
inference engine 77, 86
inference scheme 86, 88
initialization 14, 38, 91, 98
input noise 187, 195
input-output relationship 18
internal dynamics 61
interpolation 178, 192
iterative 14

J

Jacobian 39, 155
jitter 183

K

knowledge base 20, 77
Kolmogorov 17

L

large data set 154
learning 13
learning rate 38
learning set 46
Levenberg-Marquardt 42, 99, 103, 154, 156, 158
likelihood 28
linear autoassociator 57
linear identification 46
linear in the parameters 19

I

identification 91, 93
inconsistency 78, 148
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear regression</td>
<td>19</td>
<td>metropolis algorithm</td>
<td>44</td>
</tr>
<tr>
<td>linear system</td>
<td>18</td>
<td>MEV</td>
<td>24</td>
</tr>
<tr>
<td>linguistic hedge</td>
<td>82</td>
<td>Mexican Hat network</td>
<td>60</td>
</tr>
<tr>
<td>linguistic modeling</td>
<td>93, 94, 98</td>
<td>MIMO</td>
<td>86, 89, 178, 193, 198</td>
</tr>
<tr>
<td>linguistic rule</td>
<td>77, 81, 102</td>
<td>minimization</td>
<td>38, 96</td>
</tr>
<tr>
<td>linguistic variable</td>
<td>78</td>
<td>minimum eigenvalue criterion</td>
<td>24</td>
</tr>
<tr>
<td>LM</td>
<td>39, 42</td>
<td>Minkowski</td>
<td>36</td>
</tr>
<tr>
<td>local minimum</td>
<td>39, 41, 99</td>
<td>min-max</td>
<td>83, 84, 86, 88</td>
</tr>
<tr>
<td>loglikelihood</td>
<td>37</td>
<td>MISO</td>
<td>86, 88</td>
</tr>
<tr>
<td>LRSI</td>
<td>25</td>
<td>MLP</td>
<td>55</td>
</tr>
<tr>
<td>LS cost function</td>
<td>26, 59, 107, 146, 152, 160, 177</td>
<td>model</td>
<td>12, 17, 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model classes</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model complexity</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model parameter</td>
<td>13, 38, 91, 92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model order</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modeling</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modeling scheme</td>
<td>13, 107, 161</td>
</tr>
<tr>
<td></td>
<td></td>
<td>momentum term</td>
<td>39, 103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>monte carlo</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSE</td>
<td>xviii, 152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multilayer perceptron</td>
<td>54, 201</td>
</tr>
<tr>
<td>madaline</td>
<td>53</td>
<td>M Narf</td>
<td>154</td>
</tr>
<tr>
<td>Mamdani</td>
<td>105, 106</td>
<td>Marquardt-factor</td>
<td>42</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>69</td>
<td>maximum likelihood estimator</td>
<td>28</td>
</tr>
<tr>
<td>MAP</td>
<td>24</td>
<td>Maxnet network</td>
<td>60</td>
</tr>
<tr>
<td>Marquardt-factor</td>
<td>42</td>
<td>McCullogh-Pitts</td>
<td>51</td>
</tr>
<tr>
<td>maximum likelihood estimator</td>
<td>28</td>
<td>MCMC</td>
<td>25</td>
</tr>
<tr>
<td>Maxnet network</td>
<td>60</td>
<td>mean BB mapping</td>
<td>xviii, 181</td>
</tr>
<tr>
<td>McCullogh-Pitts</td>
<td>51</td>
<td>mean of maximum</td>
<td>87</td>
</tr>
<tr>
<td>MCMC</td>
<td>25</td>
<td>mean square error</td>
<td>xviii, 152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurement noise</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurement variance</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurements</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>membership degree</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>membership function</td>
<td>79, 86, 89</td>
</tr>
<tr>
<td>NBJ</td>
<td>63</td>
<td>NARMA</td>
<td>62</td>
</tr>
<tr>
<td>NARMAX</td>
<td>63</td>
<td>NARMAX</td>
<td>62</td>
</tr>
<tr>
<td>NARX</td>
<td>51, 62, 200</td>
<td>NBJ</td>
<td>63</td>
</tr>
<tr>
<td>neocognitron</td>
<td>50</td>
<td>mexican hat network</td>
<td>60</td>
</tr>
<tr>
<td>neocognitron network</td>
<td>60</td>
<td>model</td>
<td>12, 17, 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model complexity</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model parameter</td>
<td>13, 38, 91, 92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>model order</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modeling</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modeling scheme</td>
<td>13, 107, 161</td>
</tr>
<tr>
<td></td>
<td></td>
<td>momentum term</td>
<td>39, 103</td>
</tr>
<tr>
<td>mean square error</td>
<td>xviii, 152</td>
<td>monte carlo</td>
<td>44</td>
</tr>
<tr>
<td>measurements</td>
<td>14</td>
<td>mean BB mapping</td>
<td>xviii, 181</td>
</tr>
<tr>
<td>membership degree</td>
<td>81</td>
<td>mean of maximum</td>
<td>87</td>
</tr>
<tr>
<td>membership function</td>
<td>79, 86, 89</td>
<td>mean square error</td>
<td>xviii, 152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurement noise</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurement variance</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measurements</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>membership degree</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>membership function</td>
<td>79, 86, 89</td>
</tr>
</tbody>
</table>
network information criterion 24
neural networks 12, 19, 49
neurofuzzy 19, 22, 38, 90, 106
Newton 41
NFIR 51, 62, 200
NIC 24
NN identification 203
NOE 63
noise 40, 177
noisy inputs 148
nonlinear 61, 91, 143
norm xv, 84
normal 81, 92, 95, 99, 104
not-operator 83
output noise 195
overfitting 23, 46, 159, 179, 181
parallel model 61
parameter variance 179
parameter vector 77
parameters 38, 178
parametric 12
pattern recognition 53, 55, 57
penalty factor 22
penalty function 23
perceptron 50, 54, 162, 188
PID 102
polynomial 18
posterior distribution 24
powered hedge 82
pp spline 18
predictor 64
premise 86, 93
prior (Gaussian) 31
prior (Laplace) 31
prior distribution 24
probability 27
probability distribution 35
process noise 146
product-sum 83, 84, 86, 88, 89, 105
pruning 24, 25, 98
optimization 14, 38, 69, 91, 93, 99
optimization rate 38
orthogonal least squares 24
R

radial basis function 20, 31, 50, 55
random walk 44
RBF 20
RBFN 20
real-time recurrent learning 71
recurrent 57, 70
recurrent backpropagation 73
recursive 14
redundancy 78, 91, 92, 106
regularization 30, 187, 189
regularization parameter 190
regulizer 47
relative sensitivity index 25
ridge value 192
robustness 33
rule base 77, 78, 81, 92, 98, 105

Shibata’s model selector 22
shifted hedge 82
simulated annealing 44, 60
simulation 64
singleton 82, 90, 107
SISO 99, 105
small data set 157
s-norm 84, 85, 91, 105
sparse data 16, 182
splines 18
spurious data 46
SSPE 14
stability 22, 33, 38, 45, 48
steepest descent 103
stopped training 46
stopping criterion 46
structure identification 93
supervised 69, 100
support vector machines 22

S

saliency 25
saturation 101
scaled hedge 82
t-conorm 85
Schwartz’ criterion 22
TDNN 62
self-organizing map 94
test set 47, 48
sensitivity 203
thin-plate 56
sensitivity model 71, 72
Tikhonov 189
sensitivity term pruning 25
Tikhonov regulizer 32
sensitivity value 25
t-norm 84, 85, 91, 105
series-parallel model 61
topology-preserving map 58
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>transient</td>
<td>64</td>
</tr>
<tr>
<td>trapezoid</td>
<td>82</td>
</tr>
<tr>
<td>trapezoidal set</td>
<td>104</td>
</tr>
<tr>
<td>travelling salesman</td>
<td>59</td>
</tr>
<tr>
<td>triangular set</td>
<td>80, 82, 89, 92, 95, 104</td>
</tr>
<tr>
<td>true inputs</td>
<td>xviii</td>
</tr>
<tr>
<td>true model</td>
<td>xviii</td>
</tr>
<tr>
<td>true outputs</td>
<td>xviii</td>
</tr>
<tr>
<td>true parameters</td>
<td>xviii</td>
</tr>
<tr>
<td>truth factor</td>
<td>86</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>86</td>
</tr>
<tr>
<td>weight decay</td>
<td>30</td>
</tr>
<tr>
<td>weight decay rate</td>
<td>30</td>
</tr>
<tr>
<td>weight smoothing regularization</td>
<td>32</td>
</tr>
<tr>
<td>weighted least squares</td>
<td>27</td>
</tr>
<tr>
<td>width</td>
<td>77</td>
</tr>
<tr>
<td>windowing</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>unfolding in time</td>
<td>73</td>
</tr>
<tr>
<td>universal approximation</td>
<td>17, 50, 54, 83, 186</td>
</tr>
<tr>
<td>universe of discourse</td>
<td>76</td>
</tr>
<tr>
<td>unsupervised</td>
<td>69</td>
</tr>
<tr>
<td>update vector</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>validation</td>
<td>14, 47</td>
</tr>
<tr>
<td>validation set</td>
<td>46</td>
</tr>
<tr>
<td>Vapnik-Chervonenkis</td>
<td>23</td>
</tr>
<tr>
<td>variability</td>
<td>xviii, 179, 180, 182, 191</td>
</tr>
<tr>
<td>variance</td>
<td>xviii, 151, 179, 181, 182</td>
</tr>
<tr>
<td>variance backpropagation</td>
<td>198</td>
</tr>
<tr>
<td>VC theory</td>
<td>23</td>
</tr>
</tbody>
</table>