Dynamical modeling with application to friction phenomena

Dissertation

zur Erlangung des Doktorgrades
der Mathematisch–Naturwissenschaftlichen Fakultäten
der Georg–August–Universität zu Göttingen

vorgelegt von

Alexander Hornstein

aus Leninabad

Göttingen 2005

D7

Referent	Prof. Dr. Ulrich Parlitz
Korreferent	Prof. Dr. Theo Geisel
Tag der mündlichen Prüfung:	09.11.2005

Contents

1 Introduction **1**

2 Modeling tasks **5**
2.1 Static modeling tasks 5
2.1.1 Regression 5
2.1.2 Classification 6
2.2 Time processing tasks 6
2.2.1 Prediction 7
2.2.2 Cross-prediction 10
2.2.3 System identification/simulation 11
2.2.4 Filtering 11
2.2.5 Dynamical classification 11

3 Static and dynamical modeling **13**
3.1 Static modeling 14
3.1.1 Embedding step 14
3.1.2 Regression step 17
3.1.3 Application of static models 19
3.1.4 Shortcomings of static modeling 20
3.2 Bias in static models 22
3.2.1 Modeling objective 22
3.2.2 Linear system 24
3.2.3 Driven logistic map 27
3.2.4 NARX system 28
3.2.5 Consequences for static models 31
3.3 Dynamical modeling 31

4 Synchronization and modeling **35**
4.1 Preliminaries 36
4.2 Identical synchronization 37
4.2.1 Synchronization manifold and stability 37
4.2.2 Coupling configurations 40
4.3 Generalized synchronization 44

4.3.1 Definition I . . . 44
4.3.2 Definition II . . . 46
4.4 Using synchronization for modeling . . . 47
4.4.1 Reliability . . . 47
4.4.2 Modeling with Lorenz systems . . . 52

5 Dynamical networks 61
5.1 General structure of dynamical networks . . . 62
5.1.1 Networks with multi-dimensional elements . . . 62
5.1.2 Networks with one-dimensional elements . . . 64
5.2 Recurrent Neural Networks . . . 67
5.2.1 Elman/Jordan Networks . . . 67
5.2.2 Locally recurrent globally forward . . . 69
5.2.3 Echo State Networks . . . 70
5.3 Practical aspects of RNNs . . . 72
5.3.1 Stability revised . . . 73
5.3.2 Internal and external mode . . . 78
5.3.3 Using selection methods . . . 87
5.3.4 Optimization of internal connections . . . 90

6 Friction 97
6.1 Friction phenomena and models . . . 98
6.2 Modeling of pre-sliding friction . . . 107
6.2.1 Experimental setup . . . 108
6.2.2 Training and testing . . . 109
6.2.3 Results . . . 111
6.3 Modeling of pre-sliding and sliding friction . . . 113
6.3.1 Training and testing . . . 114
6.3.2 Results . . . 116
6.4 Control . . . 122
6.4.1 Tracking problem . . . 122
6.4.2 Simulation setup . . . 125
6.4.3 Training . . . 126
6.4.4 Testing . . . 129
6.4.5 Results . . . 132

7 Conclusion 137
7.1 Summary . . . 137
7.2 Outlook . . . 139

A Training of black-box models 141
A.1 Cost functions . . . 141
A.2 Optimization . . . 142
A.2.1 Quadratic cost function . . . 143

A.2.2 Fast Orthogonal Search 145
A.3 Overfitting 147

B Miscellaneous **149**
B.1 Biased parameter estimations 149

Chapter 1

Introduction

Every day we are confronted with a multitude of new facts that we have to include in our decision making. Media, like television, radio, or newspapers inform us about current events from politics and economics. We are provided with the latest prices from the stock markets as well as the newest results from sports. In scientific journals we are informed about new research achievements with myriads of data and facts that support the presented theories. In magazines we read about the newest software and hardware issues in the field of computer technology, about new fashion trends, about the latest medical advices, about the best way to invest money, and so on. To refer to all the different kinds of information in a complete manner would simply be impossible. Especially with the growing popularity of the Internet in the last decade, one can say that there is virtually no information that we cannot access.

Since human beings depend on information to plan ahead and decide their course of action, it seems that we live in heavenly times. With the infinite pool of facts that can be accessed day and night everybody should be able to make the best possible decisions. Curiously enough, this is not what is happening. Instead of using the information to their advantage, people are often overwhelmed by it. The sheer amount of facts makes it impossible to filter the important things from the unimportant and to prioritize the results. It is not a question of whether but of how many important decisions in the politics, industry, or in private households are made in the wrong way because vital information could not be found or was simply overlooked.

Clearly, the people need help managing information nowadays, and researchers all over the world are working on solutions. Concepts like 'Data mining' and 'Data warehouse' have become buzzwords in recent years. These concepts do not try to produce new data but are simply meant to manage the data that is already there in a meaningful way and to reveal information that is hidden in the data pool. This is done by searching for common patterns, by interpolating and extrapolating among data points, and by developing models that can extract the essential laws behind them. Although the nomenclature

may vary from decade to decade, the dream behind these concepts is an old one, namely that of artificial intelligence.

The question of how to draw conclusions from information that is already in our possession is very interesting and has been bothering researchers for a very long time. Our brain performs this task every day, sometimes without us even noticing it consciously. When we see a pedestrian cross the street, our brain predicts the most likely trajectory that he is going to follow and we adjust the speed of our vehicle or take evasive action according to it. Our decision making is influenced by facts like if the pedestrian is paying attention to the traffic, if he is a child or an elder, if he limps, and so on. Of course our decision does also depend on our experience, e.g. if we do not know that elder people usually walk slower, we cannot include this information in our strategy. The speed, with which the brain processes information from the world around us and filters it for important data, is simply marvelous and even modern computers have difficulties in matching it in some fields. Especially the processing of audio-visual information seems to be the domain in which the human brain still outperforms computers. However, the limitations of the brain are also obvious. It is inferior to computers in processing abstract data, e.g. the computation of the natural logarithm of 423772418 or the counting of the letter 'a' in the Bible would pose a very time-consuming problem for most of the people while it would be a matter of mere seconds for a computer. The advantages of computers are their superior computational abilities and their sheer limitless memory capacity.

So on the one hand we have the human brain, which is able to find patterns very quickly and link different kinds of information but which also is slow in computing numbers and has a limited capacity. On the other hand we have computers, which are very fast in mathematical operations and have a lossless and virtually infinite memory at their disposal but which lack the ability to draw meaningful conclusions from their 'knowledge'. The straightforward solution to the problem of dealing with too much information is to combine the strengths of both entities to compensate for their individual weaknesses. At the moment there are two different routes to this goal. One is integrating artificial computational implants in the human brain, enhancing its memory capacity and providing superior computational abilities. Although this still sounds like Science Fiction, research on the interface between neurons and computer chips is well on its way. The other route is a little bit older and consists of enabling computers to learn. In this context learning means not simply the storage of new information but the finding of hidden patterns in the data as already described above. Without remotely implying something like consciousness, the second route can be subsumed in the term artificial intelligence.

Eventually at this point the reader might be wondering how this thesis, with its title suggesting a work about modeling applications, is related to

the problem of information processing, which was described in such a grand scope in the previous paragraphs. One part of the answer is simple. On a much smaller scale the development of prediction models is nothing else but making sense of provided information, finding patterns in it, and extracting the common laws by which the data is governed.

The other part of the answer has to do with the different processing strategies of brains and computers. This work was started with a basic question about the mechanisms behind the human brain: How is it possible that the brain can store information and use it to draw new conclusions? To our best knowledge the brain does not store information in some 'memory cells' whose state can be permanently set to specific values, like zeros and ones in computers. Rather, learning takes place by strengthening or weakening connections between neurons. If the brain is represented by a dynamical system, this means that a few changes of parameters enable this system to access the information when stimulated with with a specific external input. So what is the mechanism behind it? Although we will not be able to answer this question, the considerations lead to interesting ideas. We will present two different modeling strategies. One that employs models with a lossless memory and is similar to the digital processing strategy of computers. And another that involves models whose memory is affected by forgetting. Interestingly, the latter is based on the synchronization phenomenon, an effect that is also hypothetically considered being the key mechanism of memory access in in the human brain.

In **Chapter 2** the concept of modeling is specified and discussed in detail. Although modeling can also mean the qualitative description of real phenomena through mathematical models, only the quantitative kind of models, applicable to tasks like prediction or classification, is considered in this thesis. This chapter is intended to make readers familiar with different modeling scenarios and the corresponding notations. Several kinds of modeling tasks are described and the important difference between one-step and free-running predictions is explained.

In **Chapter 3** the static modeling concept is introduced. This approach consists of two steps. In the first one data sequences are transformed into individual patterns lying in an abstract space. In this way information in time, i.e. correlations between data points, can be represented as relations in space. In the second step these relations are tried to be captured by a static function, which is fitted to the patterns with conventional regression techniques. Since the patterns formed from the time series can include values that reach arbitrarily far into the past, the memory of static models is formally lossless. In the chapter the polynomial NARMAX model serves as an example of a static model. It is shown that static modeling has certain weaknesses. An alternative modeling approach is schematically described that can overcome these weaknesses. This approach is named dynamical modeling. It consists of a dynamical system that is driven by an external signal and can employ

its state variables to form a suitable output for the specific modeling task. The internal states are similar to the patterns in the static modeling approach but instead of being artificially constructed in a separate processing step they are an inherent feature of dynamical models. In contrast to static models dynamical models cannot store values from the past with arbitrary precision. For reasons of stability they are based on a fading memory.

The dynamical modeling approach is strongly related to synchronization. Therefore, **Chapter 4** gives a short review of this phenomenon with its focus on identical and generalized synchronization. The latter is used as a basis for developing the concept of reliability, which is a necessary requirement for dynamical modeling. An example of dynamical modeling is presented in a network-like model that uses coupled Lorenz systems as its elemental modules.

Network-like models seem to be best suited for applications in dynamical modeling. Therefore, this type of models is discussed in **Chapter 5** from a broader perspective. After an introduction to notational issues recurrent neural networks are reviewed in more detail as the most prominent representative of models with a network structure. It is shown how these models fit into the concept of dynamical modeling. Leading from that, some ideas for improvements to the usual techniques in modeling with recurrent networks are presented.

In **Chapter 6** the static and the dynamical modeling approach are applied to frictional motion. After introducing the reader to friction phenomena, examples with simulated and measured friction data are presented. It is shown that dynamical models seem to be better suited for describing friction phenomena than static models. In a simulated control application the usage of a dynamical model as controller is demonstrated.

In the last chapter, **Chapter 7**, the main points of this thesis are summarized and some ideas for future work are presented.

Chapter 2

Modeling tasks

Many different modeling problems have been investigated in the past decades. Unfortunately, namings and definitions vary from publication to publication, making it difficult to find a common ground for discussions. In order to clarify things and to allow the reader an easier access, the definitions and meanings of the most important concepts are introduced. In analogy to static and dynamical modeling approaches it makes sense to explicitly differentiate between *static modeling tasks* (Section 2.1) and *time processing tasks* (Section 2.2).

2.1 Static modeling tasks

Naturally, the most important characteristic of static modeling tasks is the lack of reference to concepts like time, sequence, or dynamics. Data are presented in the most abstract way as separate patterns, which have no natural order and can be disarranged without influencing the immanent information. Two of the most important representatives of static modeling tasks are *regression* and *classification*.

2.1.1 Regression

The data patterns for regression tasks are represented as value pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i = 1, \ldots, N$, called *regressor value* $\boldsymbol{x}_i \in \mathbb{R}^m$ and *target value* $\boldsymbol{y}_i \in \mathbb{R}^n$. It is assumed that there exists the same functional relationship $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$ between regressor and target value of every pattern: $\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_i)$, $i = 1, \ldots, N$. The regression task is to find a function $\hat{\boldsymbol{f}}(\cdot)$ that approximates $\boldsymbol{f}(\cdot)$ as best as possible (see Fig. 2.1). With the help of the model function $\hat{\boldsymbol{f}}(\cdot)$ it is possible to interpolate or extrapolate the target values $\boldsymbol{y}_j$ for new query points $\boldsymbol{x}_j$, $j = N + 1, \ldots, N + K$.

Usually, in modeling approaches only the one-dimensional case $n = 1$ is regarded, where the target value is a scalar $y_i \in \mathbb{R}$ and a scalar function $f(\cdot)$ is fitted to the data. However, this is not a serious constraint, as in most

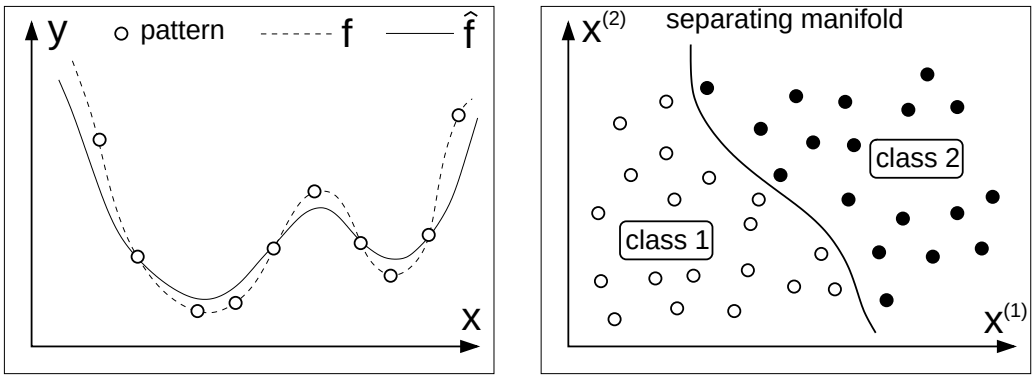

Figure 2.1: Left: In a regression task the goal is to find an approximating function $\hat{f}(\cdot)$ of the relationship $y_i = f(\boldsymbol{x}_i)$ based on the information from the patterns. **Right:** In a classification task the goal is to find a separating manifold, which separates points of class 1 from points of class 2 in the feature space.

cases the results can be easily extended to the multidimensional case $n > 1$ by treating every component of the target value separately. Only when there are strong interdependencies between the components of the target, a special treatment in the context of *multidimensional regression* is necessary.

2.1.2 Classification

Similar to regression the classification task operates with value pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i = 1, \ldots, N$. However, the target value, which in this context is often called *label*, is an element of a discrete finite set: $\boldsymbol{y}_i \in \mathcal{I} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k\}$. Each label $\boldsymbol{y}_i$ attributes its corresponding point $\boldsymbol{x}_i \in \mathbb{R}^m$ to one of the k different classes in a unique way. It is assumed that the labeling of a point $\boldsymbol{x}_i$ can be deduced from its m components or *features* $(x_i^{(1)}, \ldots, x_i^{(m)})$. The classification task is to find a rule that can correctly predict the label of each point.

For the special case of two classes $k = 2$ the classification task reduces to finding a *separating manifold* $\mathcal{M} = \{\boldsymbol{x} \in \mathbb{R}^m \mid f(\boldsymbol{x}) = C\}$, with $C \in \mathbb{R}$, in the features space $\mathbb{R}^m$. The manifold $\mathcal{M}$ separates the points of the two classes from each other (see Fig. 2.1). The labels s_1 and s_2 can be attributed to points $\boldsymbol{x}_i$ according to whether $f(\boldsymbol{x}_i) > C$ or $f(\boldsymbol{x}_i) < C$, respectively.

2.2 Time processing tasks

As the name suggests time processing tasks rely heavily on the concept of time. In contrast to static modeling tasks the data are not represented as separate, independent patterns but as ordered sequences of data values, also called *time series*, or as continuous signals. The reference to time indicates that the source of the data is usually less abstract than in static modeling tasks. Typically,

the data are taken from measurements or knowledge of real (physical) systems. The values provide information about the state of these systems at concrete points in time. Thus, time determines the natural order of the data and any disarrangement results in a change of inherent information.

An interesting topic is the modeling of continuous signals. In contrast to discrete time series their continuous nature renders a direct application of numerical techniques impossible. However, there are two main approaches to treat continuous signals. The usual way is to sample the signal in equidistant time intervals (*sampling time*) and then to perform a numerical processing on the sampled time series. An alternative way is the usage of analogue devices. With their help continuous signals can be processed without relying on discretization as an intermediate step.

Before presenting examples of time processing tasks, a formal remark is necessary. Throughout the thesis we will use the following notation

$$\begin{aligned} \{u_t\}_{t\in\mathbb{I}} \quad &\longrightarrow \quad \text{discrete: } u_t\,, t \in \mathbb{I} \subset \mathbb{Z}\,, \\ &\longrightarrow \quad \text{continuous: } u(t)\,, t \in \mathbb{I} \subset \mathbb{R}\,, \end{aligned} \tag{2.1}$$

with an abstract index set $\mathbb{I}$ if we speak about signals or time series. This shorthand represents an ordered sequence of data points, including both the discrete and the continuous case, whenever the range of the index t is not important for the context. For example in the discrete case the index set can be equal to the set of integers $\mathbb{I} = \mathbb{Z}$ (infinite), or $\mathbb{I} = \mathbb{N}$ (right-infinite), or $\mathbb{I} = \{1, 2, \ldots, N\}$ (finite). Similarly, the shorthand in Eq. (2.1) is used for continuous signals with the index set from the set of real values, e.g. $\mathbb{I} = \mathbb{R}$ (infinite).

In the following sections we limit our attention to the processing of discrete time series. Important time processing tasks are *prediction*, *cross-prediction*, *system identification/simulation*, *filtering*, and *dynamical classification*.

2.2.1 Prediction

Given a time series $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}} = \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_t \in \mathbb{R}^d$, $t \in \mathbb{I}$, we assume that future values are functionally related to values from the past

$$\boldsymbol{u}_{t+1} = \boldsymbol{f}(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots)\,, \tag{2.2}$$

with $\boldsymbol{f} : \mathbb{R}^d \times \mathbb{R}^d \times \ldots \to \mathbb{R}^d$. The prediction task is to estimate future values of the time series by exploiting this functional relationship. Here, we have to differentiate between three different kinds of prediction: the *one-step prediction*, the *multistep prediction*, and the *free-running prediction*.

For the one-step prediction past values of the original time series $\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots$ are used to estimate some future value $\boldsymbol{u}_{t+T}$

$$\hat{\boldsymbol{u}}_{t+T} = \boldsymbol{g}(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots)\,, \tag{2.3}$$

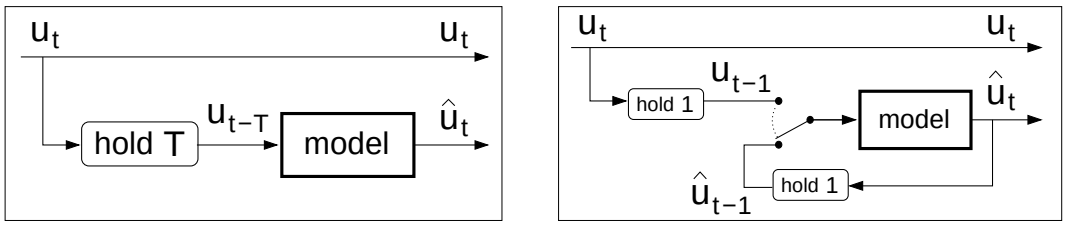

Figure 2.2: Left: In the one-step prediction the model produces an estimate $\hat{\boldsymbol{u}}_t$ of the original value $\boldsymbol{u}_t$ when provided with a delayed version $\boldsymbol{u}_{t-T}$ of the same signal. (Equivalent to predicting $\boldsymbol{u}_{t+T}$ with $\boldsymbol{u}_t$.). **Right:** In a free-running prediction the model is initialized with values from the original time series and then decoupled from the data source. The model's next estimates $\hat{\boldsymbol{u}}_t$ of the original value $\boldsymbol{u}_t$ are based solely on its own previous predictions $\hat{\boldsymbol{u}}_{t-1}$.

with $\boldsymbol{g} : \mathbb{R}^d \times \mathbb{R}^d \times \ldots \to \mathbb{R}^d$ the modeling function and $T \in \mathbb{N}$ the *prediction step* (see Fig. 2.2). Note that 'one-step' does not refer to the value of the prediction step T. It means rather that the estimation of $\boldsymbol{u}_{t+T}$ is performed in one step.

In a multistep prediction the same goal of estimating the future value $\boldsymbol{u}_{t+T}$ is pursued in a different way. Instead of predicting $\boldsymbol{u}_{t+T}$ in one long step, many short step predictions with an intermediate step length Δ, with $T = K \cdot \Delta$, $K \in \mathbb{N}$, are performed in an iterative way. A schematic of a multistep predictions for an intermediate step length $\Delta = 1$ is shown in Fig. 2.3. Based on the values of the original time series a one-step prediction is performed, yielding the values on the first prediction level. These values are used in the next step to perform a 2-step prediction, an so on ... The values on the T-th prediction level represent T-step predictions. That means to predict $\boldsymbol{x}_{t+T}$ in a T-step prediction the following values have to be computed iteratively

$$\begin{aligned} \hat{\boldsymbol{u}}_{t+1} &= \boldsymbol{g}(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots)\,, \\ \hat{\boldsymbol{u}}_{t+2} &= \boldsymbol{g}(\hat{\boldsymbol{u}}_{t+1}, \boldsymbol{u}_t, \ldots)\,, \\ \ldots &\quad \ldots \\ \hat{\boldsymbol{u}}_{t+T} &= \boldsymbol{g}(\hat{\boldsymbol{u}}_{t+T-1}, \hat{\boldsymbol{u}}_{t+T-2}, \ldots)\,. \end{aligned} \tag{2.4}$$

For the prediction of the next value $\boldsymbol{u}_{t+T+1}$ the whole iteration process is repeated after initializing with the values of the original time series

$$\begin{aligned} \hat{\boldsymbol{u}}_{t+2} &= \boldsymbol{g}(\boldsymbol{u}_{t+1}, \boldsymbol{u}_t, \ldots)\,, \\ \hat{\boldsymbol{u}}_{t+3} &= \boldsymbol{g}(\hat{\boldsymbol{u}}_{t+2}, \boldsymbol{u}_{t+1}, \ldots)\,, \\ \ldots &\quad \ldots \\ \hat{\boldsymbol{u}}_{t+T+1} &= \boldsymbol{g}(\hat{\boldsymbol{u}}_{t+T}, \hat{\boldsymbol{u}}_{t+T-1}, \ldots)\,. \end{aligned} \tag{2.5}$$

If the model function $\boldsymbol{g}(\cdot)$ is a perfect approximation of $\boldsymbol{f}(\cdot)$, the T-step predictions on the same level are functionally related to each other, namely by

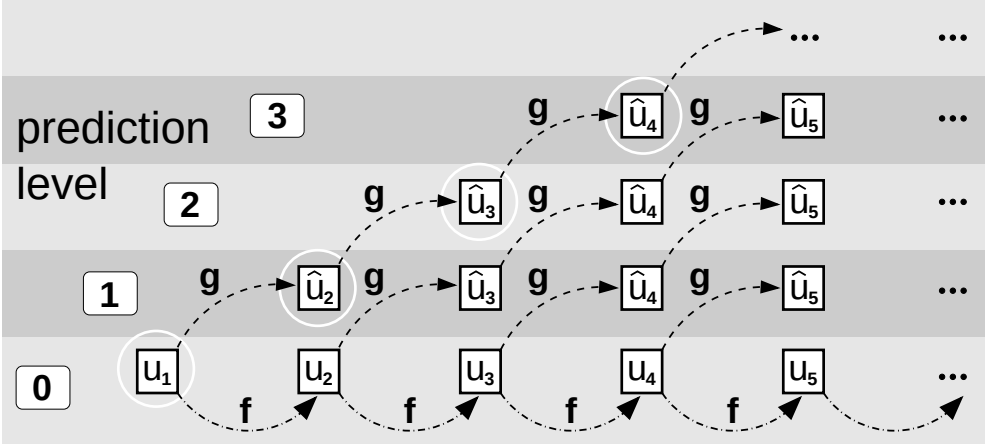

Figure 2.3: Schematics of multistep predictions with a prediction step length $T = 1$. The original time series is produced by the function $\boldsymbol{f}(\cdot)$ on the 0th prediction level. The one-step predictions on the 1st level are produced by the model function $\boldsymbol{g}(\cdot)$ and are based on the original values. The 2-step predictions on the 2nd level are based on the one-step predictions, and so on ... Note that multistep predictions on the same level are not related by any functional relationship. The first value of every prediction level (marked with white circles) represents a value of free-running predictions.

the function $\boldsymbol{f}(\cdot)$. However, in general, small errors of the model function tend to destroy this functional relationship for high values of T.

Multistep predictions are sometimes superior to one-step predictions, yielding smaller errors. Their advantage is that they rely on short step predictions, which usually allows for much simpler model functions $\boldsymbol{g}(\cdot)$ as compared to one-step predictions. A disadvantage of multistep predictions, however, is that errors tend to accumulate in the iteration process.

The third kind of predictions are free-running predictions (see Fig. 2.2 and Fig. 2.3). This approach takes the multistep prediction to its extreme. Instead of restarting and reinitializing the iterative process for every value $\boldsymbol{u}_{t+T}, \boldsymbol{u}_{t+T+1}, \ldots$ the iteration is simply continued. Effectively the prediction step T is incremented for every estimated value, i.e. while $\hat{\boldsymbol{u}}_{t+T}$ is a T-step prediction the next value $\hat{\boldsymbol{u}}_{t+T+1}$ is a $T+1$-step prediction (see Fig. 2.3).

Aside from the initialization, the free-running predictions never refer to values of the original time series and thus deviations of model predictions cannot be corrected. This means that prediction errors will stay in tolerable limits only if the model function $\boldsymbol{g}(\cdot)$ is a good approximation of $\boldsymbol{f}(\cdot)$[1]. In contrast to multistep predictions the free-running predictions are always functionally

[1] If the data source is a chaotic system, the free-running predictions may deviate strongly from the original time series even when the model function $\boldsymbol{g}(\cdot)$ is a perfect approximation of $\boldsymbol{f}(\cdot)$. The reason for this is the inherent instability of chaotic trajectories where small deviations grow exponentially in time. As a consequence a low error of free-running predictions is not suited as a quality criterion for models of chaotic systems.

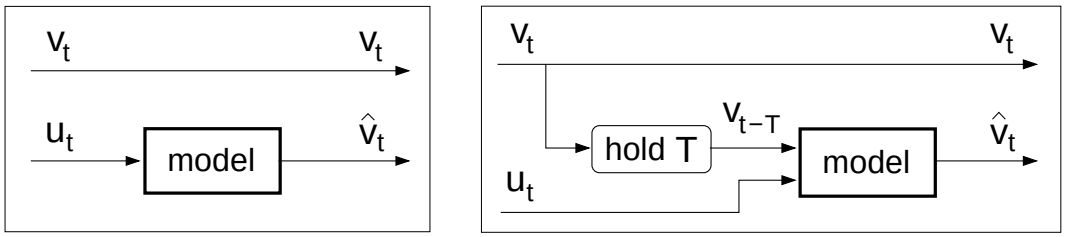

Figure 2.4: **Left:** Pure cross-prediction: Provided with signal $\boldsymbol{u}_t$ the model produces predictions $\hat{\boldsymbol{v}}_t$ of original values $\boldsymbol{v}_t$. **Right:** Mixed cross-prediction: Provided with signal $\boldsymbol{u}_t$ and a delayed version $\boldsymbol{v}_{t-T}$ the model produces predictions $\hat{\boldsymbol{v}}_t$ of the original values $\boldsymbol{v}_t$ (one-step prediction scheme).

related to each other. However, this relationship is grounded on the model function $\boldsymbol{g}(\cdot)$ and not on the original function $\boldsymbol{f}(\cdot)$.

2.2.2 Cross-prediction

Cross-prediction is very similar to prediction. Instead of one, we are given two time series, $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$ and $\{\boldsymbol{v}_t\}_{t\in\mathbb{I}}$. The task is to estimate the value $\boldsymbol{v}_t$ of one time series with the values $\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots$ of the other time series

$$\hat{\boldsymbol{v}}_t = \boldsymbol{g}(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots)\,, \tag{2.6}$$

with $\boldsymbol{g}(\cdot)$ the model function. Cross-predictions into the future or the past $\boldsymbol{v}_{t+T}$, with $T \in \mathbb{Z}$, are subsumed under this case by setting $\tilde{\boldsymbol{v}}_t = \boldsymbol{v}_{t+T}$, $t \in \mathbb{I}$, and using the shifted signal $\{\tilde{\boldsymbol{v}}_t\}_{t\in\mathbb{I}}$ instead of $\{\boldsymbol{v}_t\}_{t\in\mathbb{I}}$. Alternatively a corresponding time shift can be performed on the other time series $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$. We call this approach a *pure cross-prediction* (see Fig. 2.4). Discrimination between one-step or multistep predictions is unnecessary as former estimates are not used for following ones.

Sometimes it is beneficial to consider also past values $\boldsymbol{v}_{t-T}, \boldsymbol{v}_{t-T-1}, \ldots$ of the same time series for the estimation of $\boldsymbol{v}_t$

$$\hat{\boldsymbol{v}}_t = \boldsymbol{g}(\boldsymbol{v}_{t-T}, \boldsymbol{v}_{t-T-1}, \ldots, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots)\,. \tag{2.7}$$

This alternative approach is a mixture of a simple prediction with a prediction step T and a pure cross-prediction. We call it a *mixed cross-prediction* (see Fig. 2.4). As described in the section about prediction, we have to discriminate between a one-step, a multistep, and a free-running prediction.

The cross-prediction task is a very general formulation of a problem, and we will see that all the following tasks are in fact special cases of cross-prediction. Even the previously introduced prediction task can be easily described within this framework.

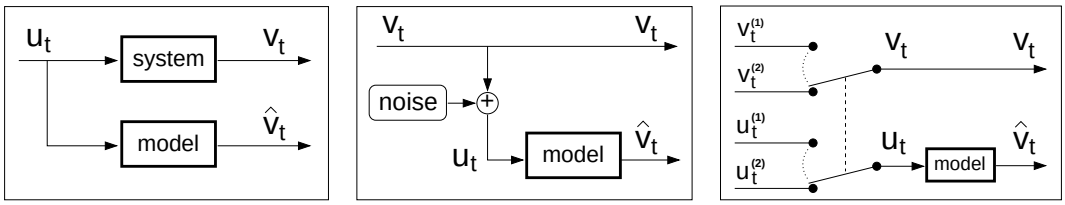

Figure 2.5: Left: In system identification/simulation the model imitates the input-output behavior of the system. Provided with the same input time series $\boldsymbol{u}_t$ the model produces estimates $\hat{\boldsymbol{v}}_t$ of the original system output $\boldsymbol{v}_t$. **Middle:** One typical task in filtering is denoising of corrupted signals. Provided with the noisy signal $\boldsymbol{u}_t = \boldsymbol{v}_t + \boldsymbol{\varepsilon}_t$ the model produces estimates $\hat{\boldsymbol{v}}_t$ of the clean signal $\boldsymbol{v}_t$. **Right:** The model in dynamical classification produces an estimate $\hat{\boldsymbol{v}}_t$ of the label values $\boldsymbol{v}_t = \boldsymbol{v}_t^{(1,2)}$ according to whether the incoming signal $\boldsymbol{u}_t$ origins from data source 1 ($\boldsymbol{u}_t = \boldsymbol{u}_t^{(1)}$) or data source 2 ($\boldsymbol{u}_t = \boldsymbol{u}_t^{(2)}$).

2.2.3 System identification/simulation

In system identification/simulation the input-output behavior of a system is characterized by the input time series $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$ and the output time series $\{\boldsymbol{v}_t\}_{t\in\mathbb{I}}$. The task is to find a model that can mimic the behavior of the system, i.e. provided with the same input time series the model should produce an output time series $\{\hat{\boldsymbol{v}}_t\}_{t\in\mathbb{I}}$ as similar as possible to the output time series of the system (see Fig. 2.5). It is easy to see that this task fits well into the framework of cross-prediction.

2.2.4 Filtering

Filtering always aims at modifying and transforming signals. In the filtering task the model represents a filter with the purpose of transforming an incoming time series $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$ into a desired form $\{\boldsymbol{v}_t\}_{t\in\mathbb{I}}$.

A typical problem is the denoising of corrupted time series. Assume a time series $\{\boldsymbol{s}_t\}_{t\in\mathbb{I}}$ is transferred through a noisy channel that adds random distortions $\boldsymbol{\varepsilon}_t$ to it $\tilde{\boldsymbol{s}}_t = \boldsymbol{s}_t + \boldsymbol{\varepsilon}_t$. The filtering task is to create a model that can reproduce the clean signal $\{\boldsymbol{s}_t\}_{t\in\mathbb{I}}$ when provided with the noisy one. By setting $\boldsymbol{v}_t = \boldsymbol{s}_t$ and $\boldsymbol{u}_t = \tilde{\boldsymbol{s}}_t$, $t \in \mathbb{I}$, the problem can be reformulated as a cross-prediction task (see Fig. 2.5).

2.2.5 Dynamical classification

In static classification tasks, data patterns $(\boldsymbol{x}_i, y_i)$ comprise points $\boldsymbol{x}_i$ and labels y_i. The points belong to a finite number of classes, and each label marks the affiliation of its corresponding point to one class in a unique way. Similarly, in dynamical classification tasks there are signals $\{\boldsymbol{u}_t^{(j)}\}_{t\in\mathbb{I}}$, $j \in \{1, 2, \ldots, J\}$, belonging to a finite number of different data sources. Each one of these

source signals is accompanied by a label signal $\{v_t^{(j)}\}_{t\in\mathbb{I}}$ attributing it to one specific data source. Typically, the label signals are constant in time $v_t^{(j)} = C_j$, and their value is an element from a finite discrete set $C_j \in \{s_1, \ldots, s_j\}$. The modeling task is to create a model that can reproduce the label signal $\{v_t^{(j)}\}_{t\in\mathbb{I}}$ for each corresponding source signal $\{\boldsymbol{u}_t^{(j)}\}_{t\in\mathbb{I}}$ (see Fig. 2.5). Again, the connection to cross-prediction is obvious.

An example for dynamical classification is a monitoring device recording and interpreting a time series $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$ from a system. The system can transit between a fully functional state and a barely functional state, thereby changing the characteristics of signal $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$. The change is formally equivalent to switching the data source of the recorded signal and can be detected by dynamical classification. A possible scenario: If the system if fully functional, the recorded signal is $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}} = \{\boldsymbol{u}_t^{(1)}\}_{t\in\mathbb{I}}$ and the label function $v_t^{(1)} = +1$, $t \in \mathbb{I}$, is signaling a valid system state. When a problem occurs, the signal change $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}} = \{\boldsymbol{u}_t^{(1)}\}_{t\in\mathbb{I}} \rightarrow \{\boldsymbol{u}_t\}_{t\in\mathbb{I}} = \{\boldsymbol{u}_t^{(2)}\}_{t\in\mathbb{I}}$ produces a new label function $v_t^{(2)} = -1$, $t \in \mathbb{I}$, signaling a failure. Catching this signal, the monitoring device can set off an alarm or initiate countermeasures.

Chapter 3

Static and dynamical modeling

In this chapter the reader is introduced to the concepts of *static modeling* and *dynamical modeling*. The first section (Section 3.1) starts with the static modeling approach. Compared to dynamical modeling it is still the favored approach in the fields of science and engineering. Most of the modern modeling procedures are based upon static modeling because it allows the usage of sophisticated regression tools on time processing tasks. The main idea of this approach is to encode time information in static data patterns, thus facilitating the application of memoryless model functions. The advantage of these functions is that they can be adapted to the data patterns by common regression techniques.

One common type of static models are the so called NARMAX models, which are a nonlinear extension of the well known linear ARMAX models[1]. For the concrete case of NARMAX models concepts and shortcomings of the static modeling approach are demonstrated. A subtle problem, which nevertheless can considerably degrade the model performance, occurs for static models if the data are noisy. In this case the structure selection and the parameter estimation in the modeling procedure systematically produce wrong results, a phenomenon referred to as biased estimation. This problem is described in detail in the second section (Section 3.2).

The weaknesses of the static modeling approach lead directly to the formulation of an alternative method in Section 3.3, namely the dynamical modeling approach. In contrast to static modeling the dynamical modeling approach employs models with memory. Despite their greater complexity these models are predestined for time processing tasks because of their internal dynamics. They handle sequential data without relying on an extra encoding step. However, this advantage is also their greatest handicap. Unlike static modeling the dynamic modeling approach has not the mathematical backup of the static regression machinery. Nevertheless, in recent years new concepts and tools, designed for dynamical problems, have been developed that make dynamical

[1] **N**onlinear **A**uto**R**egressive **M**oving **A**verage with e**X**egeneous input

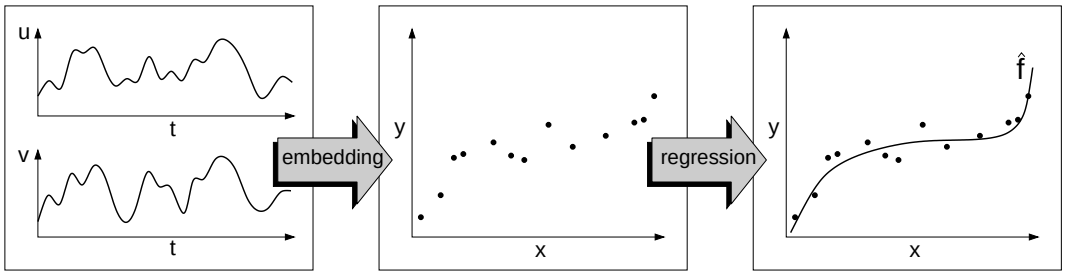

Figure 3.1: Embedding step and regression step in a static modeling approach. In the first step sequential data is transformed into static data patterns. In the second step a function is fitted to the patterns.

modeling attractive for applications.

3.1 Static modeling

The main idea of static modeling is to reduce time processing tasks to static regression tasks and thus to facilitate the application of the many well developed regression tools. For this purpose static modeling comprises two separate steps. In the first step, which we call *embedding step*, time information in sequential data is encoded in regression patterns (see Fig. 3.1). In the second step, called the *regression step*, a static model function is fitted to these patterns. This devision into two distinct processing steps is characteristic for the static modeling approach.

Not associated with the modeling procedure but also very characteristic for static models is the way in which they are applied. Since these models are memoryless functions, which operate on patterns, their application has to be preceded by a preprocessing step, in which the sequential data is translated in an appropriate way.

3.1.1 Embedding step

During the embedding step, data sequences are transformed into data patterns. For a demonstrative example assume that a finite scalar time series $\{u_t\}_{t\in\mathbb{I}} = u_1, \ldots, u_N, \mathbb{I} = \{1, 2, \ldots, N\}$, is given and that every value $u_t \in \mathbb{R}$ is produced deterministically by

$$u_t = f(u_{t-1}, u_{t-2}, \ldots, u_{t-d}) \,, \tag{3.1}$$

with an arbitrary function $f : \mathbb{R}^d \to \mathbb{R}$ and involving $d \in \mathbb{N}$ past values of the same time series from the past. Further, we assume the time processing task to be a one-step prediction into the future. Although this task is dynamical, it can be reformulated as a static regression task. For this purpose the data series is reordered during the embedding step, thereby creating data patterns

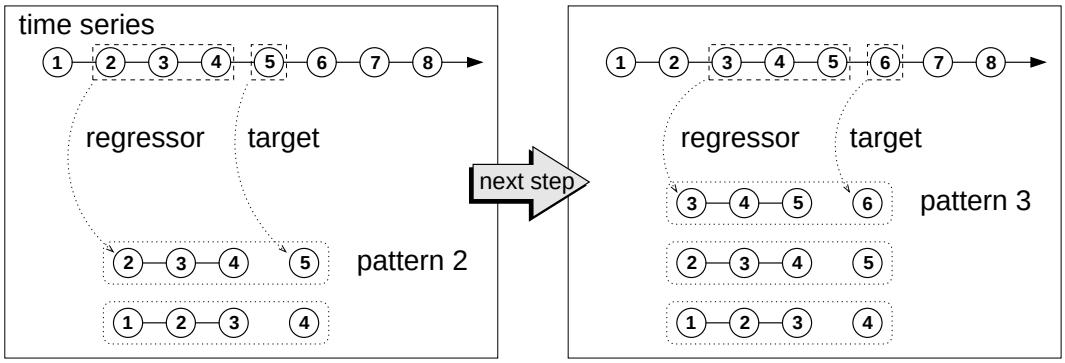

Figure 3.2: The embedding procedure: Step for step a sliding time window selects regressors and corresponding targets from the time series.

$(\boldsymbol{x}_t, y_t)$ with the *regressor* $\boldsymbol{x}_t$ and the *target value* y_t. The targets are simply set equal to the current value of the time series,

$$y_t = u_t\,, \quad t = 1, 2, \ldots, N\,, \tag{3.2}$$

and the regressors are chosen as vectors in the d-dimensional *embedding space*,

$$\boldsymbol{x}_t = (u_{t-1}, u_{t-2}, \ldots, u_{t-d})\,, \quad t = 1, 2, \ldots, N\,, \tag{3.3}$$

filled with previous values corresponding to the target (see Fig. 3.2). The data patterns $(\boldsymbol{x}_t, y_t)$, which are generated during the embedding step, are used in the next step for fitting a static regression function $\hat{y}_t = g(\boldsymbol{x}_t)$. In this regression step the data patterns are treated as being independent and having no natural order. That means a reordering of the patterns has no influence on the outcome of the regression procedure. All time information is encoded within the patterns.

The regressors are effectively windows sliding over the time series. Since they have to be filled with data values, there is a problem at the beginning of the time series. Concretely, the first d regressors, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$, cannot be fully filled because, according to Eq. (3.3), they rely on the values $u_0, u_{-1}, u_{-2}, \ldots$, which are unknown. There are two ways of dealing with this problem. The first one is to drop the first d patterns and using $t = d + 1$ as the first possible index[2]. The second one is called *zero-padding* and consists in setting the unknown values equal to zero. Both variants have their merits, the first one being more accurate the second one being more economical with the data. The reader should keep in mind, though, that embedding in general goes along

[2] In this case it would be mathematically more thorough to introduce an extra index $t' = t - d$ for the patterns with the range $t' \in [1, \ldots, N - d]$. However, such an explicit distinction is more confusing than helpful and will be avoided where the meaning should be clear from the context.

with a decrease in the number of data points or a distortion of data by the introduction of auxiliary patterns. Problems can arise if the time series is very short and there are long-term dependencies. However, in such a case the usage of high dimensional regressors is not advisable anyway, because then the embedding space is too sparsely filled for any reasonable modeling procedure.

In many ways the above example is a simplification of the situation one usually encounters in time processing tasks. It might invoke the false notion that the embedding step is a small formality which can be dealt with in passing. However, this is not the case. The embedding step is very important for the the modeling procedure. The values of many parameters are chosen during this stage, and the success of the following regression step depends heavily on this choice.

For example, the embedding dimension d of the regressor is almost never known a priori and has to be defined in an appropriate way. Choosing a low value, one risks to loose important information from the past. Choosing a high value often introduces redundancy and wastes computational resources. Another important question is: which past elements should be chosen as components for the regressor? For finely sampled signals from continuous systems it often makes no sense to include consecutive values because the informational gain is very small. Typically, a *time delay*, $\tau \in \mathbb{N}$, is introduced to sparsely choose values from the past

$$\boldsymbol{x}_t = (u_{t-\tau}, u_{t-2\tau}, \ldots, u_{t-d\tau})\,. \tag{3.4}$$

However, equidistant values in time may not be optimal for some problems. By allowing individual delays, $\tau_i \in \mathbb{N}$, $i = 1, \ldots, d$, components can be selected from arbitrary times in the past

$$\boldsymbol{x}_t = (u_{t-\tau_1}, u_{t-\tau_2}, \ldots, u_{t-\tau_d})\,. \tag{3.5}$$

The optimization of these parameters is not trivial as the number of combinations increases dramatically with the embedding dimension.

Vector valued time series, $\{\boldsymbol{u}_t\}_{t\in\mathbb{I}}$, $\boldsymbol{u}_t \in \mathbb{R}^m$, $\mathbb{I} = \{1, \ldots, N\}$, are treated in analogy to scalar time series. However, now all dimensions have to be considered in the regressor

$$\boldsymbol{x}_t = (u^{(1)}_{t-1}, u^{(1)}_{t-2}, \ldots, u^{(1)}_{t-d}, u^{(2)}_{t-1}, \ldots, u^{(m)}_{t-d})\,. \tag{3.6}$$

Again, individual time delays can be introduced if necessary, making the optimization of parameters arbitrarily complicated. Similarly, past values of other time series, $\{\boldsymbol{v}_t\}_{t\in\mathbb{I}}$, e.g. for cross-predictions, can be incorporated into the regressor. All these measures lead effectively to higher embedding dimensions and increase modeling complexity and computational costs.

3.1.2 Regression step

During the regression step a model function

$$\hat{y}_t = g(\boldsymbol{x}_t \,|\, w_1, \ldots, w_M)\,, \tag{3.7}$$

is fitted to the patterns $(\boldsymbol{x}_t, y_t)$, $t = 1, \ldots, N$, that were produced in the embedding step. Typically, the model depends on a finite number of parameters w_i, $i = 1, \ldots, M$, which have to be adapted to the data. The functional relationship between the regressors $\boldsymbol{x}_t$ and the targets y_t is in general nonlinear. Therefore, the model function $g(\cdot)$ has to include nonlinearities in some way, and techniques from nonlinear regression have to be applied. The specific procedure can be chosen from many possible approaches like NARMAX models, Support Vector models, Neural Networks, local models, and so on [49, 30].

NARMAX models

A relatively simple approach are additive NARMAX models, which are a nonlinear extension of the well known, linear ARMAX models [17, 3]. The model function is formulated as a linear superposition of nonlinear basis functions

$$g(\boldsymbol{x}_t | w_1, \ldots, w_M) = \sum_{i=1}^{M} w_i g_i(\boldsymbol{x}_t)\,. \tag{3.8}$$

The advantage of this approach is twofold. On the one hand, the nonlinearity of the model function is controllable by the number M and the type of the basis functions. On the other hand, the parameters w_i in Eq. (3.8) enter linearly into the model, making an estimate in the scope of a least squares fitting very simple.

The usage of basis functions can be interpreted as a nonlinear transformation of the regressors $\boldsymbol{x}_t \in \mathbb{R}^m$ from the linear into the nonlinear feature space (see Fig. 3.3). In the nonlinear feature space the new regressors

$$\tilde{\boldsymbol{x}}_t = (g_1(\boldsymbol{x}_t), \ldots, g_M(\boldsymbol{x}_t))\ , \tag{3.9}$$

are the transformed versions of the original regressors $\boldsymbol{x}_t$. Even if the original regressors $\boldsymbol{x}_t \in \mathbb{R}^m$ are nonlinearly related to the corresponding target values y_t, the functional relationship of the new regressors $\tilde{\boldsymbol{x}}_t \in \mathbb{R}^M$ can turn out to be linear. However, this depends on the right choice of the basis functions.

Possible classes of basis functions are monomials or polynomials [3, 19], rational functions [72, 20], radial functions [71, 73], or wavelets [12]. For example, monomials are products of the components in vector valued inputs $\boldsymbol{z} = (z_1, z_2, \ldots, z_d)$

$$g(\boldsymbol{z}) = \prod_{j=1}^{d} z_j^{p_j}\ , \tag{3.10}$$

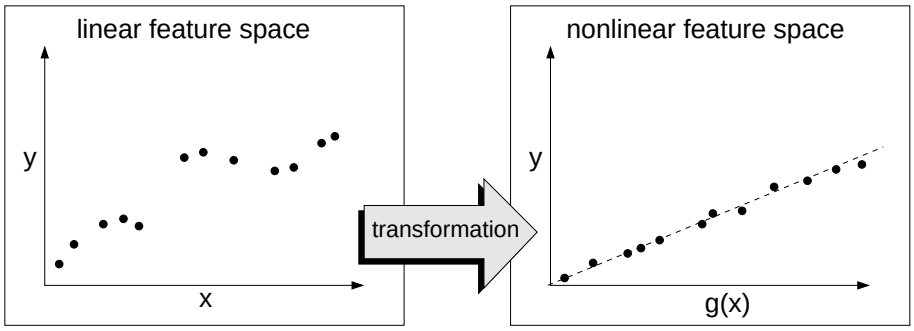

Figure 3.3: The features (components) of the regressors $\boldsymbol{x}_t$ are often nonlinearly related to the corresponding target values y_t. An appropriate transformation of the features with nonlinear basis functions may yield an almost linear relationship between the new features $\tilde{\boldsymbol{x}}_t = (g_1(\boldsymbol{x}_t), \ldots, g_M(\boldsymbol{x}_t))$ and the original target values y_t.

with the exponents $p_j \in \mathbb{N}$. The sum of the exponents

$$\max \sum_{j=1}^{d} p_j = p\,, \tag{3.11}$$

defines the *degree* (or *order*) of the monomial.

The most common method to estimate the parameters or weights w_i in Eq. (3.8) is the *least squares minimization* (see Section A.2.1). The main idea is to minimize the squared deviations of the model outputs from the original target values. The least squares minimization yields the solution

$$\hat{\boldsymbol{w}} = \left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}\right)^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{y}\,, \quad \hat{\boldsymbol{y}} = \tilde{\boldsymbol{X}} \hat{\boldsymbol{w}}\,, \tag{3.12}$$

with the target vector $\boldsymbol{y} = (y_1, \ldots, y_N)^T$, the model output $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_N)^T$, the estimated weight vector $\hat{\boldsymbol{w}} = (\hat{w}_1, \ldots, \hat{w}_M)^T$, and the regressor matrix

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} g_1(\boldsymbol{x}_1) & \cdots & g_M(\boldsymbol{x}_1) \\ \vdots & \ldots & \vdots \\ g_1(\boldsymbol{x}_N) & \cdots & g_M(\boldsymbol{x}_N) \end{pmatrix}. \tag{3.13}$$

In order to avoid the inclusion of redundant basic functions, the computation of the model parameters is usually combined with a selection technique, e.g. the Fast Orthogonal Search (FOS) [43, 44, 18] (see also Section A.2.2). Iteratively basis function are selected from a pool of possible candidates into the model. Every iteration step only the best function from the pool is included. The quality of each function is measured by its contribution to the model performance on the training data. In this way a parsimonious model is created.

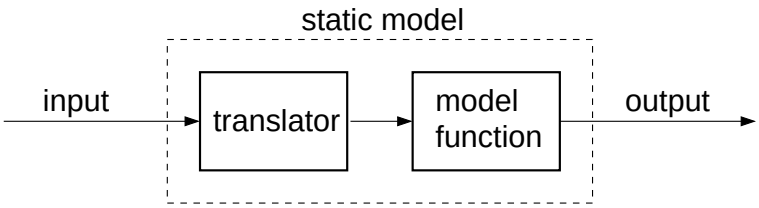

Figure 3.4: The model function in static modeling cannot process sequential data directly. Instead, the input is preprocessed by a translator, which transforms the data sequence into data patterns and passes the latter to the model function.

However, since the selection is a sequential process, which considers only one basis function at a time, it does not lead necessarily to the smallest possible model. Usually, a suboptimal solution is found. Nevertheless, the FOS is a relatively fast algorithm that can be relied on to produce adequate models with good performances.

Although the naming may imply otherwise, the reader should keep in mind that the regression step in the static modeling approach is in most cases more than a mere estimation of parameters. While the embedding step determines how time information in the data is encoded, the regression step is responsible for developing a model structure that can handle the data. This involves the choice of parameters and nonlinearities, the complexity and such things as stability of the model.

3.1.3 Application of static models

The final goal of time processing tasks is a model that can process sequential data from the same source as the training data in an appropriate way. This assignment is achieved in static modeling in an indirect way. During training the modeling task is altered from a time processing task to a static modeling task, and the model function is adapted to process static data patterns. The embedding step is a formal trick to simplify the modeling procedure. However, the price one has to pay for this simplification is the inability of the model function to process sequential data directly. A characteristic aspect of static modeling is that the model function depends on a translator that converts time series into patterns (see Fig. 3.4).

The translator uses the parameters from the embedding step to rearrange the time series into patterns. In this way the translator is a necessary key, giving the model function access to the data. The final applicable model in static modeling consists of two important parts: the translator and the model function.

3.1.4 Shortcomings of static modeling

One of the shortcoming of static modeling was already indirectly mentioned. It is the separation of the embedding and the regression into two different steps. The separation is a weak spot of static modeling because time information in the data and the structure of the models have to be individually adjusted and thus cannot be coordinated adequately. In the end, this leads to a try and error approach, where different combinations of embeddings and model structures have to be tested. For complicated dependencies on time and for multiple input signals it is often not possible to test every possible embedding, leading often to suboptimal solutions.

Another weakness that concerns specifically the NARMAX models but is inherently present in most of the static approaches is the way in which the model structure and the parameters of the model are determined. All selection methods, including the FOS, are based on the MSE cost function

$$\mathrm{MSE}_1(\boldsymbol{w}) \equiv \frac{1}{N}\sum_{t=1}^{N}\left(y_t - \hat{y}_t(\boldsymbol{w})\right)^2 , \tag{3.14}$$

the index 1 indicating that the model outputs $\hat{y}_t$ are one-step predictions (see Section 2.2.1). The selection procedure finds the best suited basis functions that lower the cost function in Eq. (3.14) the most. In parallel the weights $\boldsymbol{w} = (w_1, \ldots, w_M)$ are defined by the global minimum of Eq. (3.14). This least squares estimation leads to the best possible model for one-step predictions.

However, a problem arises if instead of one-step predictions the modeling goal involves free-running predictions. In this case lowering cost function MSE_1 in Eq. (3.14) does not result in an optimal model. Models that perform well on one-step predictions are not automatically the best suited for free-running predictions. Therefore, if the goal of the modeling procedure is a free-running model, it makes more sense to use the following cost function

$$\mathrm{MSE}_\infty(\boldsymbol{w}) \equiv \frac{1}{N}\sum_{t=1}^{N}\left(y_t - \hat{y}_t(\boldsymbol{w})\right)^2 . \tag{3.15}$$

Formally, Eq. (3.15) looks the same as Eq. (3.14). The difference, indicated by the index ∞, is that in this case the model outputs $\hat{y}_t$ are based on free-running predictions. Lowering MSE_∞ results in the best possible free-running model. However, the difficulty now is that the minimization problem is not convex as before. By iteratively using previous predictions in a free-running scheme (see Fig. 2.2), the dependence of cost function MSE_∞ on the model weights $\boldsymbol{w}$ is strongly nonlinear. Hence, a nonlinear optimization method has to be applied, which cannot guarantee a globally optimal solution.

Even worse, iterative selection schemes, like the FOS algorithm, are not applicable for cost function MSE_∞. This is schematically depicted in Fig. 3.5.

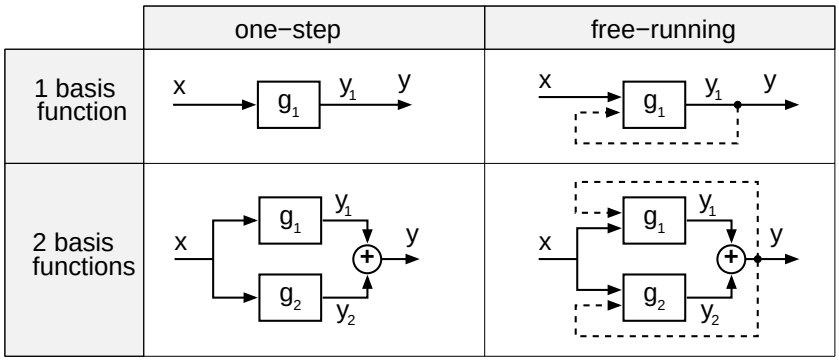

Figure 3.5: The difference between one-step and free-running predictions has consequences for the selection of basis functions into the static NARX model. For one-step predictions the performance of basis function g_1 is independent of basis function g_2. After the additional selection of g_2 the output y_1 of g_1 remains the same. This is different for free-running predictions, where the basis functions have an extra feedback (dashed line). This feedback for g_1 is different if g_2 is selected into the model. Thus, the performance of g_1 cannot be estimated independently from g_2.

If the cost function is based on one-step predictions, like MSE_1 in Eq. (3.14), the contributions of the different basis functions to the reduction of the cost function are mutually independent. Thus, the best basis functions can be selected one by one. Unfortunately, this is not possible for MSE_∞. Free-running models utilize their previous outputs for their current outputs. With this feedback loop a mutual dependence of the basis functions in the model is established. Thus they cannot be selected individually.

Since the sequential selection of individual basis functions is not possible in the scope of cost function MSE_∞, there are two alternatives for the user. The first approach is the brute-force method. One abandons the idea of forward selecting and simply tries out every possible combination of basis functions from the pool. For small function pools this is possibly the best procedure. Nevertheless, it is clear that for greater pools the brute-force procedure is absolutely impossible, as the number of combination grows exponentially with the number of basis functions.

The second alternative is a compromise. First, the selection is done with MSE_1. Afterwards, the estimation of the weights is performed with the MSE_∞ cost function. This latter approach has the advantage that it is still feasible for greater function pools. However, it has also its shortcomings. Cost function MSE_1 does not take into account that small errors tend to accumulate in free-running predictions. Especially if the model is strongly nonlinear, instabilities are often the result. Sometimes it is beneficial to use a simpler model even if it performs worse in one-step application, simply because it is more stable for free-running predictions. Therefore, cost function MSE_1 is not a suitable indicator for the quality of free-running models. Another concern is noisy data.

As will be described in the next section, distortions in the data often cause the model parameters to deviate systematically from the optimal values.

3.2 Bias in static models

In the training phase of a modeling procedure the parameters of a model are adjusted according to a specified task. Assuming that the model structure is appropriate, the parameters can take on values for which the model is considered optimal. In black-box modeling, information about the correct values has to be extracted from the training data. This extraction usually proves difficult because the information is limited by two factors: finiteness and noisiness of the data[3]. Different procedures for estimating the correct parameters can be classified by the way they deal with these two limiting factors.

Suppose a model with one free parameter $a \in \mathbb{R}$ is optimal for the parameter value $a = a_0$. A data set $\mathcal{T}$ with finite and noisy data is used for training the model and a method M is applied for estimating the model parameter. Due to the limitations of the training set $\mathcal{T}$ the estimated value of a probably shows a deviation from the correct value

$$\mathrm{M}(a \,|\, \mathcal{T}) = a_0 + \varepsilon \,. \tag{3.16}$$

The magnitude of the deviation $\varepsilon \in \mathbb{R}$ depends on the estimation method M and on the training data set $\mathcal{T}$. A statistical statement can be made about the estimation method M by looking at the average value of the estimated value for (infinitely) many training sets $\mathcal{T}_i$

$$\langle\, \mathrm{M}(a \,|\, \mathcal{T}_i) \,\rangle_i = a_0 + \langle\, \varepsilon_i \,\rangle_i \,, \tag{3.17}$$

with $\langle\, \cdot \,\rangle_i$ the average over all index values $i \in \mathbb{N}$. The average deviation $\langle\, \varepsilon_i \,\rangle_i$ is called the *bias* of the estimator M. If it vanishes the estimator is referred to as *unbiased*. Otherwise it is a *biased estimator*. Although formally incorrect, the categorization is often transferred to the model as well, which is then called biased or unbiased, respectively.

3.2.1 Modeling objective

To give the discussion some focus the effects of noisy data on modeling is considered for the special case of a system simulation, as depicted in Fig. 3.6. An arbitrary system is driven by an input signal $\{u_t\}_{t\in\mathbb{I}}$ and produces a response signal $\{y_t\}_{t\in\mathbb{I}}$. The system can either be discrete or continuous. For the

[3]Of course, there is also the case that the data is incomplete. Sometimes important variables are not observable and cannot be used in the modeling procedure. In this situation the partial lack of information has to be compensated in some way. However, this case is not regarded here. We assume that all needed variables are present.

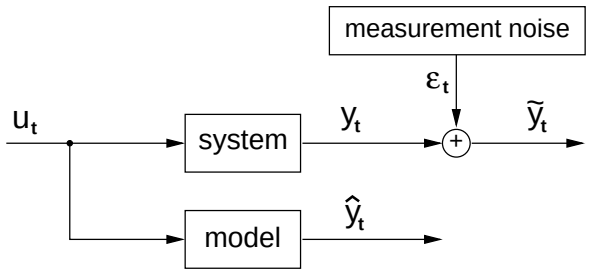

Figure 3.6: Driven by an input $\{u_t\}_{t\in\mathbb{I}}$, the system produces an output $\{y_t\}_{t\in\mathbb{I}}$. In contrast to the input signal, the output signal is not known and has to be measured, leading to imprecise values $\tilde{y}_t = y_t + \varepsilon_t$. The modeling task is to predict the original signal $\{y_t\}_{t\in\mathbb{I}}$ from the input $\{u_t\}_{t\in\mathbb{I}}$.

latter case $\{u_t\}_{t\in\mathbb{I}}$ and $\{y_t\}_{t\in\mathbb{I}}$ are assumed to be sampled versions of the corresponding continuous signals. The modeling objective is to develop a black-box model that simulates the behavior of the system, i.e. provided with the input signal $\{u_t\}_{t\in\mathbb{I}}$ it generates an output signal $\{\hat{y}_t\}_{t\in\mathbb{I}}$, which resembles the original response signal $\{y_t\}_{t\in\mathbb{I}}$ as closely as possible.

For a more realistic modeling scenario two assumptions are made. The first one concerns the model structure and is motivated by the purpose of the model. In experimental setups measurements on systems are often costly and complicated. In fact, one of the reasons for developing a model is to replace costly measurements by cheap numerical predictions. Consequently, this means that no measurements are performed during the application of the model and that predictions $\hat{y}_t$ of the original values y_t cannot rely on the knowledge of previously measured output values $\tilde{y}_{t-1}, \tilde{y}_{t-2}, \ldots$ We adopt this viewpoint for our models and restrict them in such a way that during the application phase the predictions of y_t are based solely on the knowledge of the input values $u_t, u_{t-1}, \ldots$

The second assumption concerns noise present in the input and the output signal. Since the input is usually provided by the user, it is save to assume that the values u_t of the input signal are known with sufficient accuracy, i.e. they are noise-free. However, the output signal $\{y_t\}_{t\in\mathbb{I}}$ of the system has to be measured in some way and a realistic treatment of the subject ought to include *measurement noise* (or *additional noise*). Therefore, measurements on the system produce the output values

$$\tilde{y}_t = y_t + \varepsilon_t \,, \tag{3.18}$$

which are the original output values y_t distorted by a noise signal $\{\varepsilon_t\}_{t\in\mathbb{I}}$. If not stated otherwise, we will assume the measurement errors to be normally distributed $\varepsilon_t \sim \mathrm{N}(\mu,\, \sigma_\varepsilon^2)$ with zero mean $\mu = 0$ and a finite variance $\sigma_\varepsilon^2 \in \mathbb{R}^+$.

3.2.2 Linear system

A very basic example, showing the effects of measurement noise, is the identification of a driven linear system

$$y_t = a_0 y_{t-1} + b_0 u_t \,. \tag{3.19}$$

The parameter $b_0 \in \mathbb{R}$ can be chosen arbitrarily. However, parameter $a_0 \in \mathbb{R}$ is obviously restricted to $|a_0| < 1$ for reasons of stability. Since additive measurement errors are assumed, the output is contaminated by iid[4] noise

$$\tilde{y}_t = y_t + \varepsilon_t = a_0 y_{t-1} + b_0 u_t + \varepsilon_t \,, \tag{3.20}$$

with normally distributed errors $\varepsilon_t \sim \mathrm{N}(0,\, \sigma_\varepsilon^2)$.

The model function is chosen to have the same form as the system function

$$\hat{y}_t = a\tilde{y}_{t-1} + bu_t \,, \tag{3.21}$$

with two free parameters a and b. The parameters have to be estimated on a finite data set consisting of measured output values $\tilde{y}_1, \ldots, \tilde{y}_N$ and known input values $u_1, \ldots, u_N$ during the training phase of the model.

The estimation of the parameters is done within the scope of an optimization problem by minimizing cost function MSE_1 from Eq. (3.14)

$$\begin{aligned} \mathrm{MSE}_1(a,b,N) &= \frac{1}{N}\sum_{i=1}^{N} (\tilde{y}_t - \hat{y}_t)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} \left((a_0-a)y_{t-1} - a\varepsilon_{t-1} + (b_0-b)u_t + \varepsilon_t\right)^2 \end{aligned} \tag{3.22}$$

which is an average over all squared one-step prediction errors. By looking at the cost function in the limes $N \to \infty$, statistical properties of the correlations between the signals can be exploited:

$$\begin{aligned} \mathrm{MSE}_1(a,b) &\equiv \lim_{N\to\infty} \mathrm{MSE}_1(a,b,N) \\ &= (a_0-a)^2 C_{yy}(0) + a^2\sigma_\varepsilon^2 + (b_0-b)^2 C_{uu}(0) \\ &\quad +(a_0-a)(b_0-b)C_{yu}(1) + \sigma_\varepsilon^2 \,, \end{aligned} \tag{3.23}$$

with σ_ε^2 being the variance of the noise signal $\{\varepsilon_t\}_{t\in\mathbb{I}}$.

In Eq. (3.23) the usual notation for values of the correlation function between two signals $\{v_t\}_{t\in\mathbb{I}}$ and $\{w_t\}_{t\in\mathbb{I}}$ is used:

$$C_{vw}(\tau) = \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{N} v_t w_{t+\tau} \,, \tag{3.24}$$

[4] **i**ndependently **i**dentically **d**istributed

which has the time shift τ between the signals as its only argument. From here on the explicit mentioning of the time shift will be omitted for better readability. Instead of $C_{uu}(0)$ and $C_{yy}(0)$ for the auto correlations, we will write C_{uu} and C_{yy}, respectively. The cross correlation $C_{yu}(1)$ will be expressed as $C_{\bar{y}u}$. Here, the bar over y denotes that the cross-correlation is computed for the time shifted output signal

$$C_{\bar{y}u} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} y_{t-1} u_t \,. \tag{3.25}$$

With $y_t = a_0 y_{t-1} + b_0 u_t$ it follows that

$$C_{\bar{y}u} = b_0 \sum_{j=1}^{\infty} a_0^{j-1} C_{uu}(j) \,, \tag{3.26}$$

which is a weighted summation (since $|a_0| < 1$ it is a finite geometric series) of all auto-correlation values of the input signal $\{u_t\}_{t\in\mathbb{I}}$ with a time shift j greater zero. The magnitude of parameter a_0 determines how fast the influence of long term correlations is decreased.

The cost function in Eq. (3.23) is a quadratic function, and thus it has a unique minimum. The necessary conditions for this extremum are vanishing derivations

$$\frac{\partial \mathrm{MSE}_1(a,b)}{\partial a} = -2(a_0 - a)C_{yy} + 2a\sigma_\varepsilon^2 - 2(b_0 - b)C_{\bar{y}u} \stackrel{!}{=} 0 \,, \tag{3.27}$$

$$\frac{\partial \mathrm{MSE}_1(a,b)}{\partial b} = -2(b_0 - b)C_{uu} - 2(a_0 - a)C_{\bar{y}u} \stackrel{!}{=} 0 \,. \tag{3.28}$$

From these equations it follows for the model parameters a and b that

$$a = \frac{C_{uu}C_{yy} - C_{\bar{y}u}^2}{C_{uu}C_{yy} - C_{\bar{y}u}^2 + C_{uu}\sigma_\varepsilon^2} \cdot a_0 \,, \tag{3.29}$$

and

$$b = b_0 + \frac{C_{\bar{y}u}^2}{C_{uu}}(a_0 - a) \,, \tag{3.30}$$

From the Cauchy-Schwartz inequality it follows that $C_{\bar{y}u}^2 \leq C_{uu}C_{yy}$. That means that generally the magnitude of parameter a is underestimated, while that of parameter b has a constant bias depending on a_0.

For the special case that uncorrelated white noise is used as input signal $u_t \sim \mathrm{N}(0,\, \sigma_u^2)$, it follows from Eq. (3.26) that $C_{\bar{y}u} = 0$. Since the input signal has zero mean, the auto-correlation coefficients are equal to the variances of the signals ($C_{uu} = \sigma_u^2$ and $C_{yy} = \sigma_y^2$) and the cost function in Eq. (3.23) becomes

$$\mathrm{MSE}_1(a,b) = (a_0 - a)^2\sigma_y^2 + a^2\sigma_\varepsilon^2 + (b_0 - b)^2\sigma_u^2 + \sigma_\varepsilon^2 \,. \tag{3.31}$$

The equations for the model parameters simplify to

$$a = \frac{\sigma_y^2}{\sigma_y^2 + \sigma_\varepsilon^2} \cdot a_0 \,, \tag{3.32}$$

and

$$b = b_0 \,. \tag{3.33}$$

That means that if an uncorrelated input signal is used, the estimation of parameter b is unbiased. Unfortunately, this is not the case for parameter a. Depending strongly on the relation between the output variance σ_y^2 and the noise variance σ_ε^2, the value of a deviates systematically from the true value.

The problem with biased estimations arises from the fact that cost function MSE_1 is based on one-step predictions[5]. By using cost function MSE_∞, which is based on free-running predictions, the bias can be reduced, as shown analytically in Section B.1. A numerical simulation demonstrates this phenomenon clearly. The system in Eq. (3.19) was numerically simulated with $a_0 = 0.8$ and $b_0 = 1.0$ as system parameters. In the simulation a random sequence $u_t \sim \mathrm{N}(0,\, 1)$ with normally distributed values ($N = 10000$) was used as an input signal, producing the same amount of output values $\{y_t\}_{t\in\mathbb{I}}$. The measurement errors $\{\varepsilon_t\}_{t\in\mathbb{I}}$ were chosen as normally distributed random values $\varepsilon_t \sim \mathrm{N}\big(0,\, 0.64\sigma_y^2\big)$.

On the left hand side of Fig. 3.7 cost function MSE_1 is depicted for different values of the model parameters a and b. As can be clearly seen, the minimum of the cost function lies far away from the true values $a_0 = 0.8$ and $b_0 = 1.0$. Since the position values of the minimum are taken as estimations for the model parameters a and b, the figure shows how much the model parameters deviate from the true values. While the deviation for parameter b is very small, parameter a deviates dramatically from $a_0 = 0.8$. This reflects the fact that a is biased while b is unbiased. On the right hand of Fig. 3.7 the values of cost function MSE_∞ are shown. Here, the minimum lies near the true values. That means that in this case both parameters a and b are unbiased.

[5]Actually, the deeper reason behind the biased estimate is that the least squares method is based on the assumption that the regressors are noise free. An extension of the least squares method, the so called *Total Least Squares* (TLS), can indeed cope with noise in the regressors and thus yields unbiased estimates. However, TLS is mostly limited to linear problems because the noise distribution of each individual component in the regressor has to be known. Nonlinear regressions that are made pseudo-linear with a regression matrix as in Eq. (3.13) tend to combine and transform the noise distributions of the input values in various ways, which would have to be determined before application. Therefore, TLS is unpractical for most but the simplest nonlinear problems.

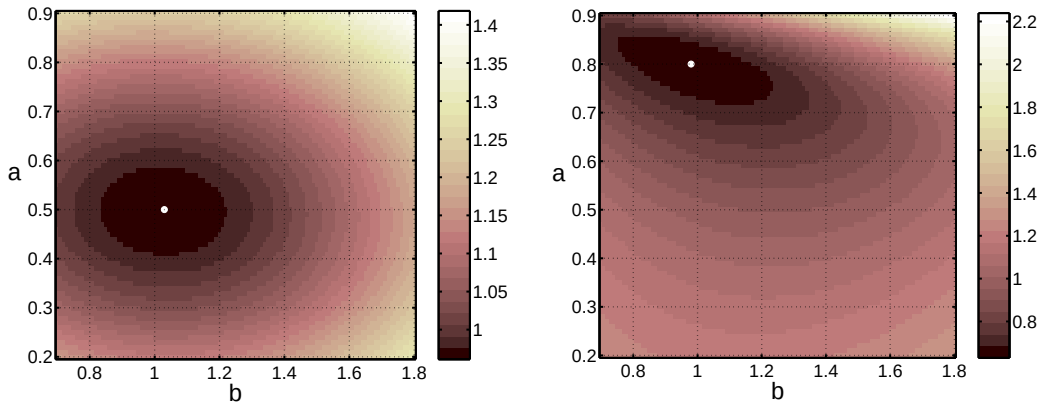

Figure 3.7: The value of the cost functions MSE_1 (**left**) and MSE_∞ (**right**) for different model parameters a and b. Note that the logarithm of the cost functions is used for better resolution. The minimum of both cost functions is marked with a white, filled circle. The true values of the linear system were $a_0 = 0.8$ and $b_0 = 1.0$.

3.2.3 Driven logistic map

In the next example the linear system from the last section is replaced by the driven logistic map

$$y_t = (1-\alpha)\lambda y_{t-1}(1-y_{t-1}) + \alpha u_t\,, \tag{3.34}$$

with the parameters α and λ. Parameter λ is the canonical parameter of the logistic map, taken from the interval $[0, 4]$. The parameter α is chosen from the interval $[0, 1]$. It plays the role of the coupling strength of the system to the input signal. For the input signal we use $u_t \sim \mathrm{U}\,[0,\,1]$, i.e. uniformly distributed random numbers from the interval $[0, 1]$.

The nonlinear system in Eq. (3.34) can be described by a NARX model

$$\hat{y}_t = a\tilde{y}_{t-1} + b\tilde{y}_t^2 + cu_t\,, \tag{3.35}$$

with three adjustable parameters a, b, and c. The optimal values for these three parameters would be $a_0 = (1-\alpha)\lambda$, $b_0 = -(1-\alpha)\lambda$, and $c_0 = \alpha$. If the data are noise-free, the optimal values are estimated correctly. Otherwise, the estimation is biased.

For a numerical simulation we have chosen the parameters $\alpha = 0.5$ and $\lambda = 4.0$. It should be noted that the value of $\alpha = 0.5$ implies a non-chaotic logistic map. A random sequence $u_t \sim \mathrm{U}\,[0,\,1]$ with uniformly distributed values in the interval $[0, 1]$ with $N = 10000$ values was used as an input signal. The measurement noise $\varepsilon_t \sim \mathrm{N}(0,\,\sigma_\varepsilon)$ was simulated with normally distributed random values. To pronounce the effects a strong noise signal was employed with the variance equal to the variance of the output signal, $\sigma_\varepsilon = \sigma_y$.

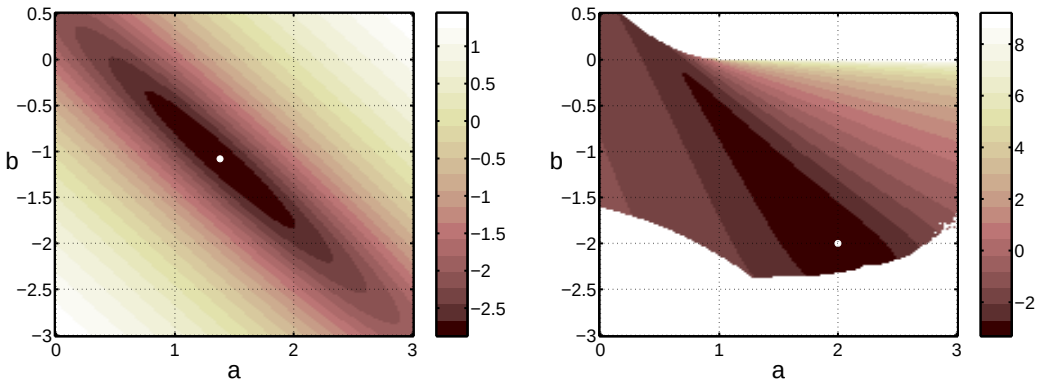

Figure 3.8: The values of the cost functions MSE_1 (**left**) from Eq. (3.36)) and MSE_∞ (**right**) for different values of a and b. For better resolution the logarithmic value of the cost functions is depicted. The minimum of the cost functions is marked with a filled white circle. The correct values were $a_0 = 2$ and $b_0 = -2$. The white areas on the right hand side show the regions, where the free-running model is unstable, leading to infinite cost values.

In Fig. 3.8 the values of cost function

$$\mathrm{MSE}_1(a, b, c = c_0) = \sum_{i=1}^{N} (\tilde{y}_t - \hat{y}_t)^2 \,, \tag{3.36}$$

are presented for various values of the parameters a and b. For a better graphical presentation in two dimensions the third parameter was held fixed at the correct value $c = c_0 = 0.5$. Similar to the linear system, the position of the minimum deviates strongly from the true values if the cost function is based on one-step predictions. For cost function MSE_∞, which is based on free-running predictions, the minimum lies exactly at the correct values $a = a_0 = 2$ and $b = b_0 = -2$.

3.2.4 NARX system

As our last example the following NARX system is considered

$$y_t = a_0 y_{t-1} + \frac{b_0}{1 + |y_{t-2}|} + c_0 u_t \,. \tag{3.37}$$

It is similar to the first, linear example in Section 3.2.2. However, with the second addend, Eq. (3.37) includes an additional nonlinearity that cannot be fully described by a polynomial NARX model. This is in contrast to both former examples in Section 3.2.2 and Section 3.2.3.

Here, the model can only approximate the nonlinear system in Eq. (3.37). There are *optimal model parameters* that guarantee the best possible approximation. However, there are no *correct model parameters* that could be compared to the parameters of the original system. This is the example that shows

the typical situation of black-box modeling: The parameters of the model have no or just an unknown analogon in the parameters of the original system. In almost all cases a black-box model solely approximates a system behavior without being able to perfectly describe it. Therefore, it is interesting to see how noise in the training data affects models that are imperfect from the beginning.

The modeling of the system in Eq. (3.37) was done with a polynomial NARX model. The FOS algorithm was applied to a pool of potential basis functions, consisting of all monomials with maximum degree $p = 5$ and the possible components $u_t, u_{t-1}, u_{t-2}, y_{t-1}, y_{t-2}$.

A training set was generated with $N = 10000$ data points. The system in Eq. (3.37) was simulated with the parameters $a_0 = 0.8$, $b_0 = -0.2$, and $c_0 = 1.0$. As an input signal a random sequence of normally distributed values with zero mean and a standard deviation of one, $u_t \sim \mathrm{N}(0, 1)$, was used. With the same parameter settings a second, independent test set was produced with the same amount of data.

The FOS algorithm yielded the following model structure

$$\begin{aligned} \hat{y}_t &= w_0 + w_1 y_{t-1} + w_2 y_{t-2} + w_3 u_t + \\ &\quad w_4 y_{t-2}^2 + w_5 y_{t-2}^3 + w_6 y_{t-2}^4 + w_7 y_{t-2}^5 \,, \end{aligned} \tag{3.38}$$

with the parameters

$$\begin{aligned} &w_0 = -0.4801,\ w_1 = 0.7970,\ w_2 = -0.0664,\ w_3 = 9.9923, \\ &w_4 = 0.1136,\ w_5 = 0.0191,\ w_6 = -0.0055,\ w_7 = -0.0009 \,. \end{aligned} \tag{3.39}$$

On the test set the model performed with a normalized mean squared error

$$\mathrm{NMSE}_\infty \equiv \frac{100\% \cdot \mathrm{MSE}_\infty}{\sigma_y^2} = 6.11\% \,, \tag{3.40}$$

meaning that the model was able to capture the system characteristics fairly well but was by no means very precise. As such this model was an ideal representative of an imperfect model, and the model structure in Eq. (3.38) was taken for further investigations concerning the influence of noise on parameter estimations.

To show the effects of noise another approach had to be taken this time. Since the model has a different structure from the system and there are no perfect parameters, systematic deviations from them cannot be shown. Instead we resorted to the model performance. Models with biased parameters show an inferior performance when simulating the input-output behavior of the given system. Thus, the following approach was taken: The models were trained on noisy data. After the training phase the models were used for simulation, and their output compared to the noise-free output of the given system. If the deviations were small, the model captured the behavior of the original,

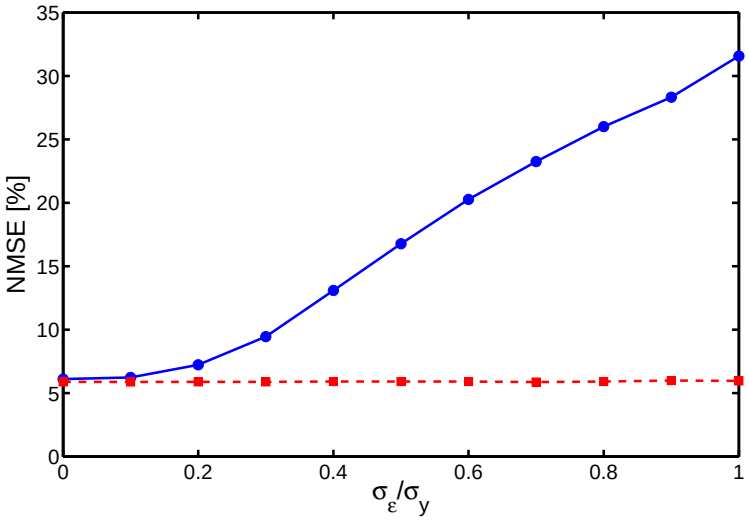

Figure 3.9: The plot shows the influence of noise in the training data on the performance of the model. The x-axis denotes the level of noise that was present in the training data. The y-axis denotes the performance of the model measured with NMSE_∞ on the noise-free test set. The filled circles represent the performance of models, trained with cost function MSE_1, while the filled squares represent the performance of models trained with MSE_∞.

undisturbed system well. In this case the noise in the training data did not introduced a bias in the model. Otherwise, if the performance were bad, the estimation of the model parameters in the training phase could be considered biased.

In the numerical experiment we utilized the training and test data set that were already employed for developing the model structure in Eq. (3.38). Random sequences with normally distributed values, $\varepsilon_t \sim \mathrm{N}(0,\, \sigma_\varepsilon)$, were added to the output time series $\{y_t\}_{t\in\mathbb{I}}$ of the training data to simulate noisy data, while the test set was left untouched. Eleven different modeling procedures were performed subsequently with different levels of noise

$$\sigma_\varepsilon = i \cdot 0.1\sigma_y\,, \quad i = 0, 1, \ldots, 10\,, \tag{3.41}$$

with σ_y being the standard deviation of the original undisturbed output series. Each modeling procedure included the adjustment of the weights w_i in Eq. (3.38) based on the minimization of the cost function MSE_1. An additional model was developed by using the alternative cost function MSE_∞. In the latter case a nonlinear optimization method was applied (**fminsearch**, Matlab 6.5, based on Nelder-Mead simplex method). Afterwards each model was tested on the clean test data set and the performance measured with the NMSE_∞ criterion, as defined in Eq. (3.40). The results can be seen in Fig. 3.9.

Obviously the models based on the minimization of one-step predictions in cost function MSE_1 were strongly influenced by the noise present in the

training data. The performance on the test set deteriorated dramatically for higher noise levels in the training set. Not the imperfect model structure was the cause for this impairment but the model bias. This can be seen by comparing the results with the performance of the models that were gained on the basis of optimizing cost function MSE_∞. Using a cost function based free-running predictions sustained the quality of the model even for very high levels of noise in the training set.

3.2.5 Consequences for static models

It was shown that noise in the training data leads to biased estimations of the parameters. In [34] a recursive method was presented that can reduce this effect. It is based on including the noise dependencies into the modeling procedure. However, this method is very time consuming. Here, another approach was introduced for the special case of system simulation. If the estimations of the parameters are done with cost functions that are based on free-running predictions instead of one-step predictions bias in the estimates can be avoided or at least reduced.

Unfortunately this method is not applicable to the selection process of the basis functions. For reasons already mentioned the selection procedures still can only be performed in the scope of cost function MSE_1, which is based on one-step predictions. Consequently, the selection process itself is always biased for noisy data. In the next section an alternative modeling approach is presented that can employ MSE_∞ for the development of the model structure.

3.3 Dynamical modeling

The problems of static modeling can be used to formulate the requirements for a better modeling strategy. The main problems are the separation of the embedding and regression step and the dependency of the selection process on the MSE_1 cost function. The former inhibits a proper coordination between the processing of time information in the data and the structure of the model. The latter leads to models that are not optimal for free-running applications and, if the data is noisy, to a biased selection and estimation of the model parameters.

Concluding from these problems, an approach is needed that can

a) merge the embedding step and the regression step into one optimization procedure.

b) employ the MSE_∞ directly for the selection of the model structure and parameter estimation.

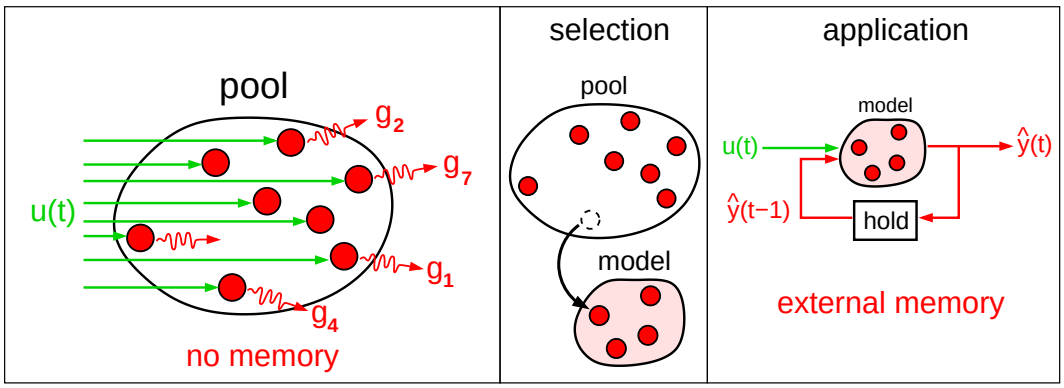

Figure 3.10: In the static modeling approach basis functions $g_i(\cdot)$ are selected from a pool of potential candidates into the model. Since the basis function are mutually independent, the ones lowering cost function MSE_1 the most can be chosen for the model in the selection process. For a free-running application a mutual dependency between the selected functions is introduced by an external feedback loop.

In the static modeling approach a model is created by choosing the best candidates from pool of basis functions (see Fig. 3.10). Since the time information is hidden in the embedding structure of the regressor patterns, these basis functions are memoryless transformation mappings. As such, the benefit of each function for the model performance can be estimated independently of the other ones. However, this performance can only be assessed for cost function MSE_1, which is based on one-step predictions. The performance for cost function MSE_∞ cannot be evaluated at this stage because an external feedback loop is used in free-running applications and this feedback loop introduces mutual dependencies between the selected basis functions in the model. Because of the external feedback loop cost functions MSE_1 and MSE_∞ have different values.

The dilemma is that one has to know all basis functions in the model before the external feedback loop can be reasonably applied and the performance of free-running predictions evaluated. Therefore an iterative selection process based on MSE_∞ is not feasible. There are three possible solutions to this problem. The first solution is to keep the external feedback loop and approximate MSE_∞ by MSE_1. This path is taken by static modeling approach. It produces often good models but suffers from the described problems.

The second solution is to avoid the external feedback loop during the free-running application. This has the advantage that cost functions MSE_1 and MSE_∞ are equivalent. However, in many cases the prediction performance suffers considerably if the feedback is removed, rendering this strategy only as a last resort.

The third solution is to incorporate the positive effects of feedback loops from the beginning into the pool so that an additional external feedback is not

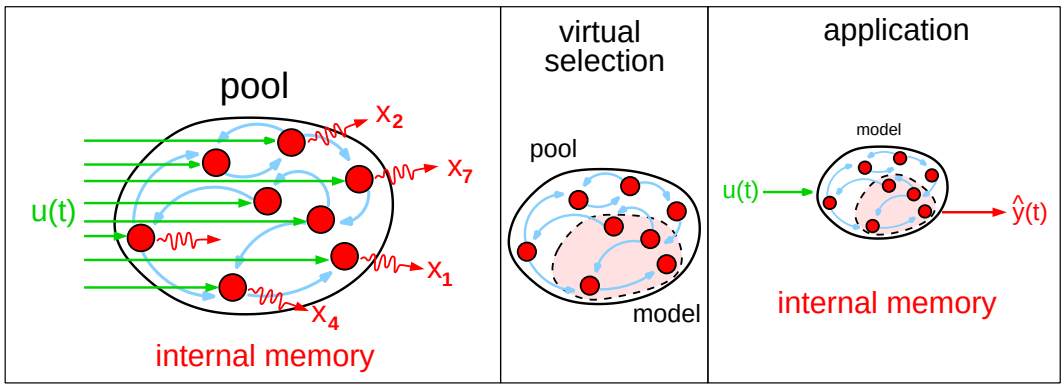

Figure 3.11: In the dynamical modeling approach a state space model is regarded as a pool of different response dynamics to a driving input signal. In a virtual selection process the state dynamics x_i that lower the cost function MSE_∞ the most are chosen to represent the model output. Because of the internal memory there is no need for an external feedback loop in free-running applications.

needed in later applications of the model. Similar to the second solution, the cost functions MSE_1 and MSE_∞ are equal. However, the prediction performance does not degrade because there are still feedback loops present. This path is taken by the dynamical modeling approach and shall be discussed here in more detail (see also Fig. 3.11).

The dynamical modeling approach includes internal feedback loops between the basis functions in the pool from the start. Since no additional external loop is applied later, the basis functions already unfold the dynamics that they also show during the applications of the model. In effect, this leads to the equivalence between one-step and free-running predictions, so that consequently $\mathrm{MSE}_\infty = \mathrm{MSE}_1$. For this reason the benefit of each basis function for free-running predictions can be assessed independently of the others. To implement the internal feedback structure, an internal memory in form of a state vector has to be introduced. The output of all basis functions in the pool is stored in this internal state vector for one time step. Since the basis functions influence each other, the state vector is used in each iteration to produce the output for the next time step.

Formally, the pool of basis functions in the dynamical modeling approach is a dynamical system that is driven by an external input signal $\{u_t\}_{t\in\mathbb{I}}$

$$\boldsymbol{x}_t = \boldsymbol{g}(\boldsymbol{x}_{t-1}, u_t)\,, \tag{3.42}$$

with $\boldsymbol{x}_t \in \mathbb{R}^M$ the state vector, storing the output of M basis functions in the pool at time step t, and $\boldsymbol{g}(\cdot) \equiv (g_1(\cdot), \ldots, g_M(\cdot))$ the vector function that includes all basis functions $g_i : \mathbb{R}^M \times \mathbb{R} \to \mathbb{R}$. The output of the model is set

in analogy to the static models as a superposition of the basis functions

$$\hat{y}_t = \sum_{i=1}^{M} w_i g_i(\boldsymbol{x}_{t-1}, u_t)\,. \tag{3.43}$$

Only the basis functions that contribute the most to a good prediction performance are needed for the model output. Since the cost functions MSE_1 and MSE_∞ are equivalent, the best ones can be chosen in a selection process similar to the static models (see Fig. 3.11). One should note, however, that this selection process is only virtual, because the whole pool is needed to create the desired dynamics. The weights w_i of the basis functions that are not selected are simply set to zero in Eq. (3.43). This is different from the static approach, where the selection process really reduced the size of the model. This is the downside of the dynamical modeling approach: The size of dynamical models usually exceeds the size of static models by far.

Nevertheless, both requirements from the beginning of the chapter are met by the dynamical modeling approach. Since an external feedback loop is not employed, the bias problem of the static models does not occur for dynamical models. Besides, an additional embedding step is not needed because the internal state $\boldsymbol{x}_t$ stores implicitly the history of the input $\{u_t\}_{t\in\mathbb{I}}$. By adjusting the parameters of the basis functions and the internal feedback loops between them, time information in the input data and model structure are optimized at the same time.

The creation of recurrent elements in the function pool through internal feedbacks is by no means trivial. In order to produce reliable response signals to an input signal the dynamics of the pool has to meet some stability criteria. This problem is discussed in more detail from the point of view of synchronization in the next chapter.

Chapter 4

Synchronization and modeling

When two systems are coupled to each other, they can sometimes assimilate their behavior. This phenomenon is called *synchronization*. Its discovery is accredited to Huygens (1629–1695), who observed that two pendulum clocks adjusted their oscillations when they were attached to the same beam [61]. Since then many other examples of synchronization were found ranging from organ pipes over biological clocks to glowworms. Especially with the upcoming of radio communication in the beginning of the 20th century, synchronization became an essential concept receiving more and more attention from engineering, mathematics, and physics.

In the last decades of the 20th century synchronization was linked with the popular chaos theory. It was found numerically and experimentally that chaotic systems could be synchronized (e.g. [27], [60], [58]). This new aspect of synchronization was spectacular because it went against all intuition concerning chaos. In chaotic systems even slight differences of initial conditions lead to completely different trajectories and it was astonishing that such systems could be brought to assimilate their dynamics and to stay locked in this state in a stable manner. Many scientists of the nonlinear dynamics community seized the opportunity to explore the new field of chaotic synchronization; and the combination of chaos theory and synchronization proved fruitful for research. Nowadays synchronization is among the mostly investigated nonlinear phenomena with many applications in fields like control, communication, or neuroscience.

In this chapter we want to give a short introduction to concepts regarding synchronization. Since there are so many of them, we will restrict the topic to the most important ones for this work. These are the *identical synchronization* in Section 4.2 and the *generalized synchronization* in Section 4.3. For broader reviews see e.g. [13], [61], [56], or [53].

In the last section (Section 4.4) the dynamical modeling approach, which was introduced in Section 3.3, is examined from the perspective of synchronization. The concept of reliability is developed, which is based primarily on

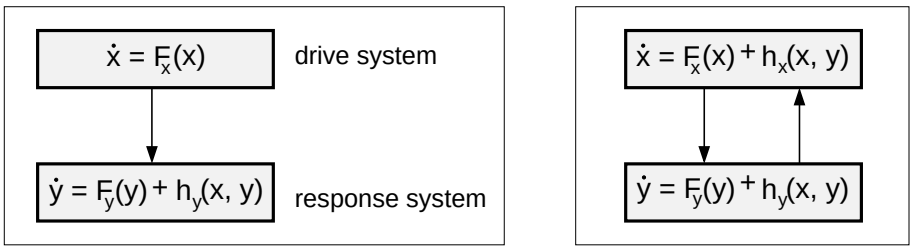

Figure 4.1: Left: Unidirectional coupling scheme between two dynamical systems. How the state of the *response system* evolves is influenced by the state of the *drive system*. **Right:** Bidirectional coupling scheme. Because of the mutual influence there is no distinction between drive and response system.

generalized synchronization. In a practical example we show how a synchronized system can be used for dynamical modeling. In this case the model system consists of interconnected Lorenz systems.

4.1 Preliminaries

Before going into detail about synchronization, a short notion on coupling schemes is required. For the general case of two coupled dynamical systems we have

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}) + \boldsymbol{h}_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{y}) \,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{F}_{\boldsymbol{y}}(\boldsymbol{y}) + \boldsymbol{h}_{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y}) \,, \end{aligned} \tag{4.1}$$

with $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$ as state vectors and $\boldsymbol{F}_{\boldsymbol{x}} : \mathbb{R}^m \to \mathbb{R}^m$ and $\boldsymbol{F}_{\boldsymbol{y}} : \mathbb{R}^n \to \mathbb{R}^n$ as vector fields of the two systems. The functions $\boldsymbol{h}_{\boldsymbol{x}} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ and $\boldsymbol{h}_{\boldsymbol{y}} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ define the mutual coupling between both systems. This coupling scheme is called *bidirectional*. In the special case that one of the coupling functions is zero, the coupling is called *unidirectional* (see Fig. 4.1). In contrast to bidirectional coupling, where both systems mutually influence each other, the information flows only in one way for the unidirectional case. The system providing the information is called *drive* or *master system*. The system receiving the information is called *response* or *slave system*.

Although the unidirectional and the bidirectional coupling schemes have similarities, they also differ in many ways and there is no general method to transfer results from one to the other. Compared to the bidirectional coupling the unidirectional coupling is a docile scenario. Since the drive system is not influenced by the response system, it sets the course of the dynamics in the mutual state space. The response system is more or less restricted to the choice of following or not following. In the bidirectional case the dynamics is not dictated by one system. Rather, both systems compete for dominance

and the dynamics in the mutual state space is the result of this struggle. The bidirectional coupling produces a whole new system with its own dynamics and thus proves more difficult for analysis. In our work bidirectional coupling plays only a minor role and if not explicitly referred to, we will restrict our attention on the unidirectional case.

All definitions in this chapter refer to the case of two coupled continuous systems. However, with minor modifications they can also be transfered to multiple coupled systems. In the case of discrete systems the ordinary differential equations in Eq. (4.1) have to be replaced with difference equations

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{x}_t) + \boldsymbol{h}_{\boldsymbol{x}}(\boldsymbol{x}_t, \boldsymbol{y}_t)\,, \\ \boldsymbol{y}_{t+1} &= \boldsymbol{f}_{\boldsymbol{y}}(\boldsymbol{y}_t) + \boldsymbol{h}_{\boldsymbol{y}}(\boldsymbol{x}_t, \boldsymbol{y}_t)\,. \end{aligned} \tag{4.2}$$

Apart from that definitions and concepts for the continuous and the discrete case are generally the same. Exceptions will be explicitly pointed out to the reader.

4.2 Identical synchronization

Identical synchronization (IS)[1] is the simplest and most intuitive case of synchronization. We speak of IS when two coupled systems assimilate their dynamics and evolve in exactly the same way. This occurs only if two identical systems are coupled to each other. As a consequence IS is a mathematical abstraction because generally it is impossible to guarantee the identity of systems in real physical environments. Nevertheless, IS can be experimentally approximated and more importantly it serves as a good starting point for the understanding of synchronization.

4.2.1 Synchronization manifold and stability

In the special case where the vector fields in Eq. (4.1) coincide $\boldsymbol{F}_{\boldsymbol{x}} = \boldsymbol{F}_{\boldsymbol{y}} = \boldsymbol{F}$ IS is possible. A possible definition is the following:

Definition 4.1 *In the unidirectional coupling scheme the drive system and the response system*

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}(\boldsymbol{x})\,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{F}(\boldsymbol{y}) + \boldsymbol{h}_{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y})\,, \end{aligned} \tag{4.3}$$

with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{F} : \mathbb{R}^m \to \mathbb{R}^m$, *and* $\boldsymbol{h}_{\boldsymbol{y}} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$, *are considered* **identically synchronized** *iff the two following conditions are met:*

[1] Alternative names for IS often found in literature are *complete synchronization* or *conventional synchronization*.

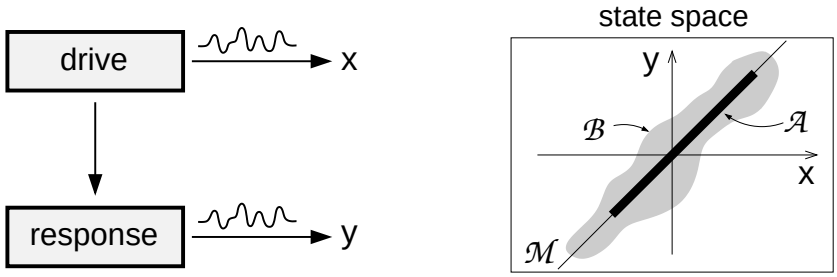

Figure 4.2: Identical synchronization occurs if two identical dynamical systems are coupled in an appropriate way. In the mutual state space the dynamics is then restricted to the synchronization manifold $\mathcal{M}$ which is a plane through the origin. The attractor $\mathcal{A}$ of the coupled system lies inside $\mathcal{M}$ and is surrounded by a basin of attraction $\mathcal{B}$.

(i) There exists a compact, invariant manifold

$$\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^m \times \mathbb{R}^m \,|\, \boldsymbol{x} = \boldsymbol{y}\}\,, \tag{4.4}$$

containing the attractor $\mathcal{A}$ of the combined system in the mutual state space

$$\mathcal{A} \subset \mathcal{M} \subset \mathbb{R}^m \times \mathbb{R}^m\,. \tag{4.5}$$

(ii) There exists a basin of attraction $\mathcal{B}$ around the attractor $\mathcal{A} \subset \mathcal{B}$ such that

$$\begin{aligned} &\forall(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^m \times \mathbb{R}^m \,: \\ &(\boldsymbol{x}(t=0), \boldsymbol{y}(t=0)) = (\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathcal{B} \;\Rightarrow\; \lim_{t\to\infty} \|\boldsymbol{x}(t) - \boldsymbol{y}(t)\| = 0\,. \end{aligned} \tag{4.6}$$

In the above definition the stability of the manifold $\mathcal{M}$ plays a crucial role. It establishes synchronization as a robust phenomenon demanding that small perturbations of the system states should not lead to a breakdown of the synchronized motion.

The stability property is closely linked to the coupling between the systems. In the generalized notation the coupling signal was assumed to be a function of the drive and the response state $\boldsymbol{h}_{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y})$. In many coupling functions a proportionality factor is utilized

$$\boldsymbol{h}_{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y}) = c \cdot \tilde{\boldsymbol{h}}_{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y})\,, \tag{4.7}$$

with $c \in \mathbb{R}$ referred to as the *coupling strength*. The coupling strength determines how much the dynamics of the response system is influenced by the drive system and it is essential for the stability of synchronization. Consider two identical systems with a vanishing coupling strength, $(c = 0)$, starting from the same initial conditions. Although they may evolve in the synchronization manifold $\mathcal{M}$ in Eq. (4.4), the systems cannot be considered synchronized.

Even for small perturbations of the systems the evolving state would leave the manifold $\mathcal{M}$ and especially for chaotic systems its distance from $\mathcal{M}$ would grow exponentially. As a consequence a finite coupling strength, ($|c| > 0$), is a necessary requirement for synchronization. Depending on the coupling configuration the coupling strength has typically even to exceed a critical value to initiate synchronization.

Conditional Lyapunov exponents

The stability of the synchronization manifold $\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \,|\boldsymbol{x} = \boldsymbol{y}\}$ in Eq. (4.4) can be investigated by looking at the *synchronization error* $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{x}$. The evolution of $\boldsymbol{e}$ is governed by

$$\dot{\boldsymbol{e}} = \boldsymbol{F}(\boldsymbol{x} + \boldsymbol{e}) + \boldsymbol{h_y}(\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{e}) - \boldsymbol{F}(\boldsymbol{x})\,. \tag{4.8}$$

These equations can be linearized for small synchronization errors near the manifold where $\boldsymbol{y} = \boldsymbol{x}$

$$\dot{\boldsymbol{e}} \approx \boldsymbol{D_F}(\boldsymbol{x}) \cdot \boldsymbol{e}\,, \tag{4.9}$$

with $\boldsymbol{D_F}$ being the Jacobian of the vector field $\boldsymbol{F}$. The Jacobian tells us about the stability properties at a specific point $\boldsymbol{x}$ in the state space. The eigenvalues of the symmetric matrix

$$\boldsymbol{\Lambda}(\boldsymbol{x}) = \boldsymbol{D_F}(\boldsymbol{x})^T \cdot \boldsymbol{D_F}(\boldsymbol{x})\,, \tag{4.10}$$

determine whether synchronization errors grow or shrink. In order to make statements about all points on the manifold the local properties have to be averaged for the whole trajectory $\{\boldsymbol{x}\}$. Starting at $\boldsymbol{x}(0)$ at time $t = 0$ one defines the averaging matrix $\boldsymbol{Y}$ as

$$\boldsymbol{Y}(0) = \boldsymbol{I}\,, \tag{4.11}$$

and integrates it along the trajectory

$$\dot{\boldsymbol{Y}}(t) = \boldsymbol{D_F}(\boldsymbol{x}(t))\boldsymbol{Y}(t)\,. \tag{4.12}$$

The eigenvalues μ_i, $i = 1, \ldots, m$, of the symmetric matrix

$$\boldsymbol{\Lambda} = \lim_{t\to\infty} \left[\boldsymbol{Y}(t)^T \cdot \boldsymbol{Y}(t)\right]^{1/2t}\,, \tag{4.13}$$

determine the stability properties of the synchronization manifold [28]. The logarithms of the eigenvalues, $\lambda_i = \ln(\mu_i)$, $i = 1, \ldots, m$, are called *Lyapunov exponents* (LEs)[2]. A necessary condition for synchronization is that all LEs

[2]With appropriate modifications Lyapunov exponents can also be computed for discrete systems or can be derived in case the state equations are unknown purely from samples of time series. For a nice review of different methods see [28].

are negative $\lambda_i < 0$. Then $\mathcal{M}$ is on the average stable. The addition 'on the average' is important as the LEs provide information only about the averaged stability behavior of a system following a typical trajectory on the attractor. Even for negative LEs there may still be local instabilities present in the manifold $\mathcal{M}$ disrupting possible synchronization for small periods of time. This is the reason why negative LEs are a necessary but not sufficient condition for synchronization [13].

For unidirectionally coupled identical systems it makes a difference whether the LEs are computed along the trajectory of the driver or the one of the driven system [63]. If the LEs are computed along the trajectory $\{\boldsymbol{x}\}$ of the driver, as described in Eq. (4.9), they are called *transverse Lyapunov exponents* (TLEs). Otherwise, if they are computed along trajectory $\{\boldsymbol{y}\}$ of the response system, they are called *conditional Lyapunov exponents* (CLEs).

The CLEs are a very general concept and define the stability properties of any non-autonom system, which is driven by an external signal. The word 'conditional' refers to the fact that the stability of the response system depends on the driving signal. If all CLEs are negative, the response system has no degrees of freedom left and its trajectory is fully determined by the trajectory of the drive system. Independent of the initial conditions, the driven system evolves in the same way for the same driving signal after the transient behavior is decayed (see also Section 4.3).

Lyapunov function

Even though the Lyapunov exponents can tell us about the local stability of the synchronization manifold, it is not possible to derive global stability from them. The latter can be proved with the help of a so called *Lyapunov function*. A Lyapunov function $\mathcal{L}(\boldsymbol{e})$ with the synchronization error $\boldsymbol{e}$ as its argument provides sufficient conditions for synchronization. It is defined as a continuously differentiable real valued function having the following two properties

(i) $\forall \boldsymbol{e} \in \mathbb{R}^m \,:\, \boldsymbol{e} \neq 0 \;\Rightarrow\; \mathcal{L}(\boldsymbol{e}) > 0 \quad \text{and} \quad \boldsymbol{e} = 0 \;\Rightarrow\; \mathcal{L}(\boldsymbol{e}) = 0,$

(ii) $\forall \boldsymbol{e} \in \mathbb{R}^m \,:\, \boldsymbol{e} \neq 0 \;\Rightarrow\; \frac{\mathrm{d}\mathcal{L}(\boldsymbol{e})}{\mathrm{d}t} < 0.$

If such a function can be found for a coupled system, a stable synchronization is ensured (see for example [42]). Compared to conditional LEs the analysis via Lyapunov function is more difficult to apply because the construction of the latter is not standardized for arbitrary nonlinear systems.

4.2.2 Coupling configurations

Many researchers have developed standard techniques for coupling two system in such a way as to guarantee synchronization. Three of the most important

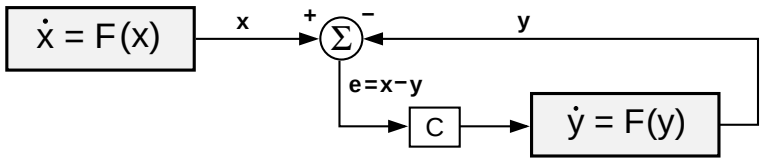

Figure 4.3: In the diffusive coupling scheme the synchronization error $\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{y}$ is used as a feedback to the response system. Whenever the mutual state $(\boldsymbol{x}, \boldsymbol{y})$ leaves the synchronization manifold $\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \,|\, \boldsymbol{y} = \boldsymbol{x}\}$, the feedback is activated, adding a dissipative component into the system dynamics. Through this dissipative mechanism the synchronization of the coupled systems is stabilized.

ones are *diffusive coupling* [10], *Pecora-Carroll decomposition* [58], and *active-passive decomposition* [41].

Diffusive coupling (DC)

The concept of DC[3] is very simple: A coupling term proportional to the synchronization error $\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{y}$ is added to the equations governing the dynamics of the response systems

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}(\boldsymbol{x})\,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{F}(\boldsymbol{y}) + \boldsymbol{C}(\boldsymbol{x} - \boldsymbol{y})\,, \end{aligned} \tag{4.14}$$

with the proportionality $m \times m$ matrix $\boldsymbol{C}$ (see Fig. 4.3). The idea is to introduce an additional dissipation into the dynamics countering possible instabilities of the synchronization manifold $\mathcal{M}$ in Eq. (4.4). If the state $(\boldsymbol{x}, \boldsymbol{y})$ in the mutual state space is evolving inside $\mathcal{M}$, the dissipation term vanishes. However, every time the evolving state tries to escape the manifold $\mathcal{M}$ due to instabilities or perturbations, the dissipation term is activated forcing the response system back.

Dissipative coupling can be analyzed with the help of conditional Lyapunov exponents (CLEs) as described in Section 4.2.1. The linearized equations for the synchronization error read

$$\dot{\boldsymbol{e}} = (\boldsymbol{D}_{\boldsymbol{F}}(\boldsymbol{x}) - \boldsymbol{C})\,\boldsymbol{e}\,. \tag{4.15}$$

Usually the Jacobi matrix $\boldsymbol{D}_{\boldsymbol{F}}$ will have positive conditional Lyapunov exponents defying synchronization. However, now they can be neutralized by an appropriate choice for the coupling matrix $\boldsymbol{C}$.

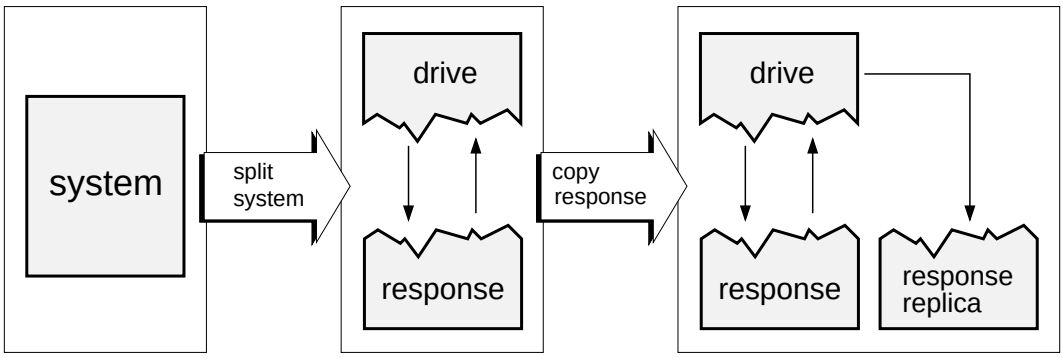

Figure 4.4: For the PCD a dynamical system is split into a drive and a response subsystem. The response system is copied and the replica system is driven in the same way as the response system. If the stability conditions are met, IS can be observed between the response and the replica system.

Pecora-Carroll decomposition (PCD)

Pecora and Carroll suggested in [58] a coupling method that involves the splitting of a system into a drive and a response subsystem (see Fig. 4.4)

$$\begin{aligned} \dot{\boldsymbol{z}} &= \boldsymbol{F}(\boldsymbol{z}) \\ &\downarrow \\ \dot{\boldsymbol{x}} &= \boldsymbol{G}_{\boldsymbol{x}}(\boldsymbol{x},\boldsymbol{y})\,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{G}_{\boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y})\,. \end{aligned} \tag{4.16}$$

After the splitting a replica, i.e. an exact copy, of the response system is built and driven by the same signal as the original response system

$$\dot{\boldsymbol{y}}' = \boldsymbol{G}_{\boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y}') \tag{4.17}$$

Provided that the conditional Lyapunov exponents are all negative the response and the replica system assimilate their dynamics yielding identical synchronization $\boldsymbol{y} = \boldsymbol{y}'$.

One disadvantage of the PCD is that the number of possible drive-response subsystems for low dimensional systems is rather limited and that the number of stable response system is even smaller.

Active-passive decomposition (APD)

The APD was suggested by Kocarev and Parlitz in [41] as a very general method for constructing coupling configurations between two systems. It starts

[3] An alternative name for diffusive coupling is dissipative coupling.

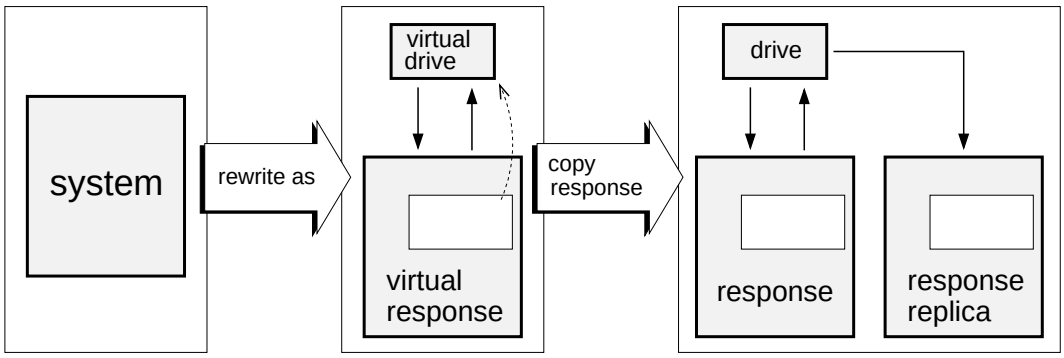

Figure 4.5: For the APD an autonomous dynamical system is rewritten as a non-autonomous system. The driving signal is ascribed to a virtual, external drive system (active part). The virtual response system, called the passive part, is copied and the replica system is driven in the same way as the virtual response system. If stability conditions hold, IS can be observed between the response and the replica system.

with a formal trick by rewriting an autonomous dynamical system as a non-autonomous one (see Fig. 4.5)

$$\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}) \quad \longrightarrow \quad \dot{\boldsymbol{x}} = \boldsymbol{G}(\boldsymbol{x}, \boldsymbol{s}(t)) \,, \tag{4.18}$$

with the driving signal $\boldsymbol{s}(t)$ defined as $\boldsymbol{s} = \boldsymbol{h}(\boldsymbol{x})$ or $\dot{\boldsymbol{s}} = \boldsymbol{H}(\boldsymbol{x})$. Even though the driving signal is a functional part of the dynamical system, it is treated as coming from a virtual drive system that influences a virtual response system. In contrast to the PCD the virtual drive system is not necessarily a subsystem of the original dynamical system but rather a function of its state. Thus APD is a more general concept with many possible coupling configurations including the ones from diffusive coupling and PCD.

Similar to PCD a replica of the virtual response system is built

$$\dot{\boldsymbol{y}} = \boldsymbol{G}(\boldsymbol{y}, \boldsymbol{s}(t)) \,, \tag{4.19}$$

and driven with the same signal $\boldsymbol{s}(t)$. Provided the synchronization manifold $\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \,|\, \boldsymbol{y} = \boldsymbol{x}\}$ is stable, identical synchronization between the virtual response and the replica system occurs.

Analyzing the stability of the synchronization for the APD approach, can either be finding a Lyapunov function or investigating the conditional Lyapunov exponents (CLEs). In the latter case the linearized equations read

$$\dot{\boldsymbol{e}} = \boldsymbol{D}_{\boldsymbol{G}}(\boldsymbol{x}, \boldsymbol{s})\boldsymbol{e} \,, \tag{4.20}$$

governing the evolution of small synchronization errors $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{x}$. The stability can be deduced from the signs of the CLEs in this equation. Negative CLEs ensure a stable synchronization between virtual response and replica system. In this case the virtual response system is acting as a passive system that is driven by an active system (virtual drive system). Hence the name active-passive decomposition.

4.3 Generalized synchronization

The research concerning identical synchronization (IS) led to a new concept called *generalized synchronization* (GS), which is broader in its scope and includes the IS as a special case [2], [65]. While IS is concerned with coupled identical systems, GS focuses mainly on the coupling between systems with different dynamical properties. As a consequence, if GS occurs, the relationship between the state variables of the drive and the response system is not as simple as $\boldsymbol{y} = \boldsymbol{x}$ anymore. Some of the concepts of IS can be transfered directly to GS. However, as a more general concept GS features many new aspects of synchronization.

In the simplest case GS involves a synchronization manifold that is a nonlinear functional relationship $\boldsymbol{y} = \boldsymbol{\Psi}(\boldsymbol{x})$ between drive and response states. Depending on the systems the function $\boldsymbol{\Psi}(\cdot)$ can either be only slightly nonlinear or it can be very complex in its shape. It can either be smooth or even non-differentiable. However, in some cases the relationship between drive and response states is multi-valued, making it impossible to define any function $\boldsymbol{\Psi}(\cdot)$ at all. Taking these difficulties into account, it is by no means a trivial task to identify GS for two unidirectional coupled systems. Two main detection methods have been established, which differ in their approach. One is based on the detection of a functional relationship between drive and response system. The other method investigates the behavior of replica systems and is related to conditional Lyapunov exponents. Since the definition of GS depends on the detection method, the two approaches lead to concepts of GS that are not quite congruent [56], [54].

4.3.1 Definition I

Consider two unidirectionally coupled dynamical systems

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}(\boldsymbol{x}) \,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{G}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) \,, \end{aligned} \tag{4.21}$$

with the state vectors $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$ and the vector fields $\boldsymbol{F}$ and $\boldsymbol{G}$. The first system is driving the second one with the coupling signal $\boldsymbol{h}(\boldsymbol{x})$. For this coupling the definition of GS follows Definition(4.1) of IS except for the characterization of the synchronization manifold $\mathcal{M}$. While for IS the manifold is simply a plane through the origin $\boldsymbol{y} = \boldsymbol{x}$, the manifold for GS can be arbitrarily shaped

$$\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^m \times \mathbb{R}^n | \boldsymbol{y} = \boldsymbol{\Psi}(\boldsymbol{x})\} \,, \tag{4.22}$$

with the $\boldsymbol{\Psi} : \mathbb{R}^m \to \mathbb{R}^n$ representing the functional relationship between the state variables $\boldsymbol{x}$ and $\boldsymbol{y}$. IS is included in the concept of GS as a special case where $\boldsymbol{\Psi}(\cdot) = \boldsymbol{id}(\cdot)$.

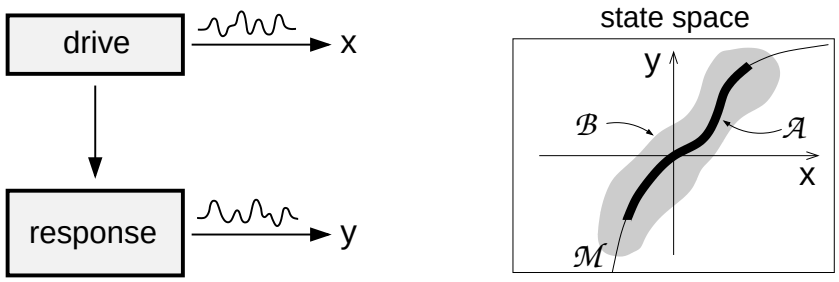

Figure 4.6: Generalized synchronization occurs if two (possibly) nonidentical dynamical systems are coupled in an appropriate way. In the mutual state space the dynamics is then restricted to the synchronization manifold $\mathcal{M}$, which can be an arbitrarily shaped hyperplane. The attractor $\mathcal{A}$ of the coupled system lies inside $\mathcal{M}$ and is surrounded by a basin of attraction $\mathcal{B}$.

Definition 4.2 *In the unidirectional coupling scheme the drive system and the response system*

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}(\boldsymbol{x})\,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{G}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x}))\,, \end{aligned} \tag{4.23}$$

with $\boldsymbol{x} \in \mathbb{R}^m$,$\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{F} : \mathbb{R}^m \to \mathbb{R}^m$, *and* $\boldsymbol{h} : \mathbb{R}^m \to \mathbb{R}^k$, *have the property of generalized synchronization iff the two following conditions are met:*

(i) There exists an attracting synchronization set

$$\mathcal{M} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^m \times \mathbb{R}^n \mid \boldsymbol{y} = \boldsymbol{\Psi}(\boldsymbol{x})\}\,, \tag{4.24}$$

containing the attractor $\mathcal{A}$ *of the combined system in the mutual state space*

$$\mathcal{A} \subset \mathcal{M} \subset \mathbb{R}^m \times \mathbb{R}^m\,. \tag{4.25}$$

(ii) There exists a basin of attraction $\mathcal{B}$ *around the attractor* $\mathcal{A} \subset \mathcal{B}$ *such that*

$$\begin{aligned} &\forall (\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^m \times \mathbb{R}^n \ : \\ &(\boldsymbol{x}(t=0), \boldsymbol{y}(t=0)) = (\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathcal{B} \ \Rightarrow \ \lim_{t\to\infty} \|\boldsymbol{y}(t) - \boldsymbol{\Psi}(\boldsymbol{x}(t))\| = 0\,. \end{aligned} \tag{4.26}$$

The definition above was used for example in [42] and [35] and explicitly formulated in [56]. Note that it makes no statement about the properties of $\boldsymbol{\Psi}(\cdot)$. Depending on the investigated drive and response systems the function may be smooth or even non-differentiable at all [63], [35]. For detecting GS in the above sense nearest neighbors statistics were suggested in [65] and [59]. These methods use the fact that for synchronized systems points in a small

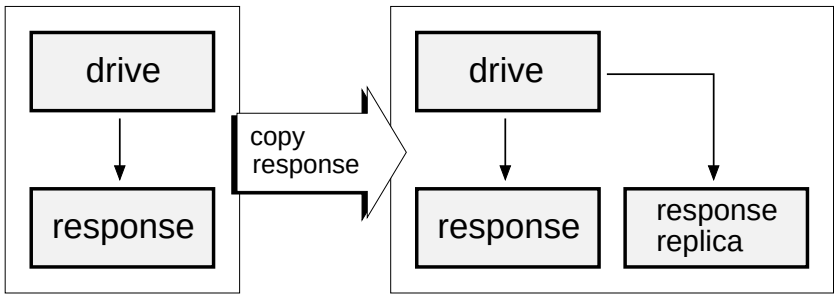

Figure 4.7: A replica system (also auxiliary system) is simply an exact copy of the response system. During an application the original system and the replica are driven by the same driving signal.

ε-region in the state space of the drive system are mapped by $\boldsymbol{\Psi}(\cdot)$ to points in the state space of the response system which lie also very near to each other. In case that no synchronization is established, these points lie arbitrarily scattered in the state space of the response system.

4.3.2 Definition II

Another method for detecting GS is the application of a so called *auxiliary* or *replica system* as described in [1] (see Fig. 4.7). The replica system is an exact copy of the response system

$$\dot{\boldsymbol{y}}' = \boldsymbol{G}(\boldsymbol{y}', \boldsymbol{h}(\boldsymbol{x}))\,. \tag{4.27}$$

The purpose of the replica is to test whether the response system retains any degrees of freedom or whether its dynamics is totally determined by the driving signal. For the former case the replica system $\boldsymbol{y}'$ and the original response system $\boldsymbol{y}$ behave differently even when driven by the same signal $\boldsymbol{h}(\boldsymbol{x})$. In the latter case, when GS is established, both systems asymptotically behave in the same way. That means in spite of possibly different initial conditions their trajectories assimilate in the course of time

$$\lim_{t\to\infty} \|\boldsymbol{y}(t) - \boldsymbol{y}'(t)\| = 0\,. \tag{4.28}$$

This behavior can be interpreted as the response system forgetting its initial state. Following [56] the second definition for GS can be written as

Definition 4.3 *In the unidirectional coupling scheme the drive system and the response system*

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{F}(\boldsymbol{x})\,, \\ \dot{\boldsymbol{y}} &= \boldsymbol{G}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x}))\,, \end{aligned} \tag{4.29}$$

with $\boldsymbol{x} \in \mathbb{R}^m$,$\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{F} : \mathbb{R}^m \to \mathbb{R}^m$, and $\boldsymbol{h} : \mathbb{R}^m \to \mathbb{R}^k$, have the property of generalized synchronization iff the following condition is met:

(i) There exists an open basin of attraction $\mathcal{B} \subset \mathbb{R}^m \times \mathbb{R}^n$ *such that*

$$\forall (\boldsymbol{x}_0, \boldsymbol{y}_0), (\boldsymbol{x}_0, \boldsymbol{y}_0') \in \mathcal{B} \Rightarrow \lim_{t\to\infty} \|\boldsymbol{y}(t) - \boldsymbol{y}'(t)\| = 0\,. \tag{4.30}$$

The second definition of GS is more general than the first one because it includes also the case where there is no functional relationship between the states of the drive and the response system due to multivaluedness.

4.4 Using synchronization for modeling

At the end of the last chapter the concept of dynamical modeling was formulated (see Section 3.3). The basic idea was that a dynamical system is used as a pool of various response signals to an external driving signal. The response signals can be combined to model any deterministic input-output relationship that consists between the driving signal and another output signal. In this section the requirements for dynamical modeling are investigated in the scope of synchronization.

Dynamical modeling relies on the stability of the driven dynamical system but also on a property that we refer to as *reliability*. Reliability is intimately related to the concept of Generalized Synchronization and is discussed in detail in Section 4.4.1. In Section 4.4.2 we show how a dynamical model can be employed for a prediction task. The example is based on the driven Lorenz-Oscillator, which was used in [41] to demonstrate the Active-Passive-Decomposition.

4.4.1 Reliability

A dynamical system that is driven by an input signal $\{u_t\}_{t\in\mathbb{I}}$ as in

$$\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}, u_t) \quad \text{or} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), u(t))\,, \tag{4.31}$$

is said to be *BIBO-stable*[4] if the output values of the system

$$y_t = g_{\text{out}}(\boldsymbol{x}_t)\,, \tag{4.32}$$

with an arbitrary output function $g_{\text{out}} : \mathbb{R}^m \to \mathbb{R}$ are finite for finite values of the input, i.e. in mathematical formalism $\forall C_u \in \mathbb{R}^+ \exists C_y \in \mathbb{R}^+ : |u_t| < C_u\,, \forall t \in \mathbb{I} \Rightarrow |y_t| < C_y\,, \forall t \in \mathbb{I}$. Since such a behavior is desirable for modeling purposes, a dynamical model is usually required to be BIBO-stable[5]. However,

[4] **B**ounded **I**nput **B**ounded **O**utput

[5] Sometimes, when it is known that the magnitude of all input values will never exceed a certain bound $C_{\max} \in \mathbb{R}^+$, a weaker version of BIBO-stability can be utilized: $\forall C_u < C_{\max} \exists C_y \in \mathbb{R}^+ : |u_t| < C_u\,, \forall t \in \mathbb{I} \Rightarrow |y_t| < C_y\,, \forall t \in \mathbb{I}$. Models with this property can become unstable if the input unexpectedly exceeds the threshold $C_{\max}$, which can happen for example if the input is noisy.

this requirement is not enough. Another important point is that the response of the driven system is always the same to a specific input signal $\{u_t\}_{t\in\mathbb{I}}$. If this were not the case, the dynamical system could not be used as a model for tasks like prediction because it would produce unreliable output and could not be trained. The response has to be independent of the initial state of the system, although there may be a transient phase in which the system adjusts to the input. We call this property *reliability* and define it as follows

Definition 4.4 *A driven discrete dynamical system*

$$\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}, u_t)\,, \tag{4.33}$$

with $\boldsymbol{x}_t \in \mathbb{R}^m$ *and* $f : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ *is said to be* reliable *iff*

$$\forall\{u_t\}_{t\in\mathbb{N}}\,\forall\boldsymbol{x}_0, \boldsymbol{x}_0' \in \mathbb{R}^m\,\forall\varepsilon > 0\,\exists T \in \mathbb{N}\,\forall t \in \mathbb{N}: \\ t > T \Rightarrow \|\boldsymbol{x}_t - \boldsymbol{x}_t'\| < \varepsilon\,. \tag{4.34}$$

In analogy a continuous dynamical system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), u(t))\,, \tag{4.35}$$

with $\boldsymbol{x}(t) \in \mathbb{R}^m$ *and* $f : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ *is called* reliable *iff*

$$\forall\{u(t)\}_{t\in\mathbb{R}^+}\forall\boldsymbol{x}_0, \boldsymbol{x}_0' \in \mathbb{R}^m\,\forall\varepsilon > 0\,\exists T \in \mathbb{R}^+\,\forall t \in \mathbb{R}^+: \\ t > T \Rightarrow \|\boldsymbol{x}(t) - \boldsymbol{x}'(t)\| < \varepsilon\,, \tag{4.36}$$

with $\boldsymbol{x}_0$ *and* $\boldsymbol{x}_0'$ *the initial states at time step* $t = 0$.

Comparing the definition of reliability with the one of Generalized Synchronization (GS) in 4.3 we find them almost identical except for two points. The first difference is that the source of the input signal is not explicitly given for reliability. That means that a reliable dynamical system has to synchronize to every possible input signal after a transient phase. The second difference is that the initial states are not limited to a basin. Since the input source is not given, practical considerations make it necessary to define reliability as a global property.

The difference between generalized synchronization and reliability is demonstrated with a simple, one-dimensional example. Consider the driven Gaussian map

$$x_t = \exp(-a^2[x_{t-1} - b]^2) + u_t\,, \tag{4.37}$$

with the parameters $a = 3.5$ and $b = 0.5$. In [54] this system was examined for the special case that the input signal $u_t = sz_t$, $s \in \mathbb{R}$, is a scaled version of the tent map

$$z_t = 1 - 2|z_{t-1} - 0.5|\,. \tag{4.38}$$

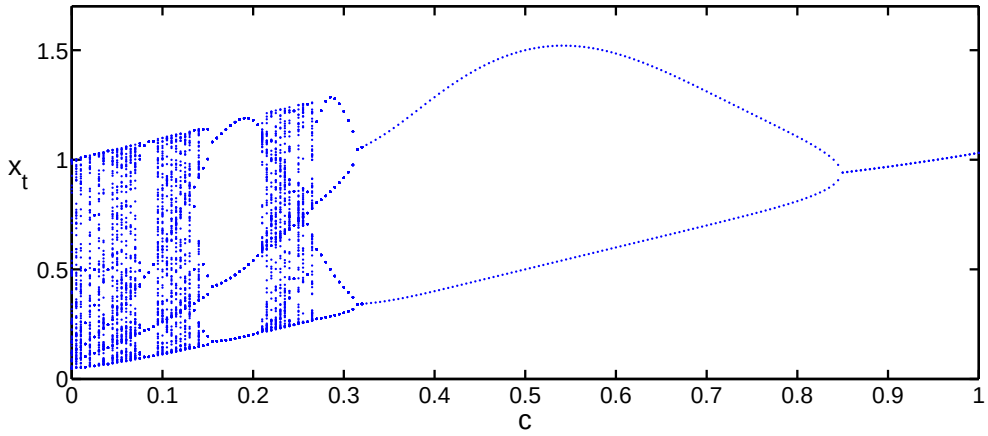

Figure 4.8: Bifurcation diagram of the driven Gaussian map in Eq. (4.37) for constant input signals $\{u_t = c\}_{t\in\mathbb{I}}$. The magnitude c of the input signal is used as a bifurcation parameter. For every magnitude $c \in [0, 1]$ the points mark 100 sequential states x_t of the Gaussian map in a stationary mode.

If the scaling factor is set to $s = 0.4$, the Gaussian map exhibits generalized synchronization [54]. In this case the system responds always in the same way to the same input signal. Nevertheless, the system is not reliable because the generalized synchronization is not indifferent to the input signals but occurs only if the latter have specific properties. For constant inputs $u_t = c$, $c \in \mathbb{R}$, the stationary behavior of the Gaussian map is shown in Fig. 4.8. One can see that the spectrum of different responses ranges from stable fixed points over n-cycles to chaotic behavior, depending on the magnitude of the input. The signal from the tent map alternates between these regimes and manages to stabilize the Gaussian map enough to invoke generalized synchronization. However, for other input signals the reliability of the Gaussian map cannot be guaranteed.

From the Gaussian map we can draw an important conclusion. A necessary criterion for the system in Eq. (4.33) to be reliable is the existence of one stable fixed point for a constant external input, i.e.

$$\forall\{u_t = c\}_{t\in\mathbb{I}}\,, c \in \mathbb{R}\,\exists\boldsymbol{x}_s \in \mathbb{R}^m : \lim_{t\to\infty} \|\boldsymbol{x}_t - \boldsymbol{x}_s\| = 0\,. \tag{4.39}$$

If this requirement is not met, e.g. if the system has multiple fixed points, the state space is divided into different basins of attraction and the response from the system depends on the initial state of the system. Another example is a system with a stable n-cycle. Although such a system has formally only one basin, it has n different ways ('phases') to run through the n-cycle. Thus, the state space is effectively divided into n sub-basins. Starting in two different sub-basins means that the system responds differently to the same input

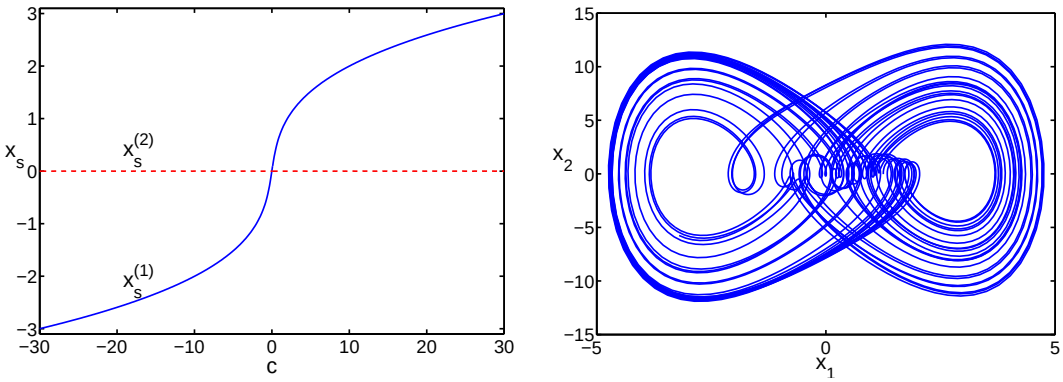

Figure 4.9: Left: Fixed point $\boldsymbol{x}_s$ of the Duffing oscillator in Eq. (4.40) for constant input $u(t) = c$ in the range $c \in [-30, 30]$. While the second component of the fixed point (dashed red line) remains at zero, the first component (blue line) changes its value according to the input. **Right:** Chaotic Duffing attractor. If the Duffing oscillator is driven by the sinus signal $u(t) = 27\sin(1.4t)$, it displays chaotic behavior.

signal[6]. The worst case for a BIBO stable system is chaotic behavior for a constant input. There is no way to get a reliable response from a chaotically system since even the smallest deviation from initial conditions already leads to totally different output signals. Note that the requirement in Eq. (4.39) and analogous arguments also apply to the continuous case.

Although an attractive fixed point for constant input signals is an important criterion, it is not sufficient to define a reliable system. Consider for example the driven single-well Duffing oscillator

$$\dot{x}_1 = x_2 \tag{4.40}$$
$$\dot{x}_2 = -x_1 - x_1^3 - 0.2x_2 + u(t)\,. \tag{4.41}$$

For a constant input $u(t) = c$, $c \in \mathbb{R}$, $t \in \mathbb{I}$, the dynamics settle on a fixed point as shown in Fig. 4.9 for the range $c \in [-30, 30]$. However, it was shown in [57] and [52] that the dynamics becomes chaotic if the oscillator is driven by an alternating signal $u(t) = a\sin(\omega t)$ with $a = 27$ and $\omega = 1.4$ (see rhs of Fig. 4.9). Therefore, even though the Duffing oscillator has a single attractive fixed point for all constant input signals, it is an unreliable system.

The Gaussian map and the Duffing example showed that the concept of reliability is similar but also distinct from generalized synchronization and global stability. A practical way of proving the reliability of a dynamical system is by finding a suitable Lyapunov function. Consider for example the

[6] An example for a stable fixed point and a coexisting 2-cycle is discussed in detail for a dynamical system of the type $\boldsymbol{x}_t = \tanh(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{b}u_t)$ in Section 5.3.1. It is shown that the basin structure depends on the input values. Thus, the same initial state can belong to different basins depending on the input signal.

Lorenz system

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2 - x_1) \\ \dot{x}_2 &= r x_1 - x_2 - x_1 x_3 \,, \\ \dot{x}_3 &= x_1 x_2 - b x_3 \end{aligned} \tag{4.42}$$

with the parameters $\sigma, b > 0$ and $r \in \mathbb{R}$. In [42] it was shown the coupling scheme

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2 - x_1) \\ \dot{x}_2 &= r u(t) - x_2 - u(t) x_3 \,, \\ \dot{x}_3 &= u(t) x_2 - b x_3 \end{aligned} \tag{4.43}$$

leads to generalized synchronization (GS) of the Lorenz system to any input signal $\{u(t)\}_{t\in\mathbb{I}}$. The difference $\boldsymbol{e}(t) = \boldsymbol{y}(t) - \boldsymbol{y}'(t)$ between trajectories of the system in Eq. (4.43) and a replica system is considered and a Lyapunov function for this variable can be constructed in the following way

$$L(\boldsymbol{e}) = (e_1^2/\sigma + e_2^2 + e_3^2)/2 \,. \tag{4.44}$$

Using the relations in Eq. (4.43) one can show further that

$$\begin{aligned} \dot{L}(\boldsymbol{e}) &= e_1 \dot{e}_1/\sigma + e_2 \dot{e}_2 + e_3 \dot{e}_3 \\ &= e_1 e_2 - e_1^2 - e_2^2 - u e_2 e_3 + u e_2 e_3 - b e_3^2 \\ &= -(e_1 - e_2/2)^2 - 3e_2^2/4 - b e_3^2 < 0 \,. \end{aligned} \tag{4.45}$$

Therefore, the trajectories of every replica system has to assimilate to the original system independent of the input signal. According to our definition such a system is called reliable.

Along with the coupling scheme in Eq. (4.43) an even simpler one was considered for the Lorenz system in [42], which is

$$\begin{aligned} \dot{x}_1 &= -\sigma x_1 + u(t) \\ \dot{x}_2 &= r x_1 - x_2 - x_1 x_3 \,. \\ \dot{x}_3 &= x_1 x_2 - b x_3 \end{aligned} \tag{4.46}$$

Driving a Lorenz system in this way also leads to GS for all possible input signals. This can be easily seen if we consider the difference $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{y}'$ between the system in Eq. (4.46) and a replica like before. For the first component it follows with $\dot{e}_1 = -\sigma e_1$ that it vanishes in the course of time. That leaves the subsystem

$$\begin{aligned} \dot{e}_2 &= -e_2 - x_1 e_3 \\ \dot{e}_3 &= x_1 e_2 - b e_3 \,, \end{aligned} \tag{4.47}$$

for which the Lyapunov function $L = e_2^2 + e_3^2$ can be defined. Following from this is

$$\dot{L} = 2(-e_2^2 - x_1 e_2 e_3 + x_1 e_2 e_3 - b e_3^2) = -2(e_2^2 + b e_3^2) < 0\,. \tag{4.48}$$

Again, this is proof that the system in Eq. (4.46) is reliable.

As an example for discrete dynamics we can use the following system, which is known from discrete-time recurrent neural networks,

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{b}u_t\right)\,, \tag{4.49}$$

with the state vector $\boldsymbol{x}_t \in \mathbb{R}^m$, the connection matrix $\boldsymbol{C} \in \mathbb{R}^{m\times m}$, the input connection vector $\boldsymbol{b} \in \mathbb{R}$, and the input $u_t \in \mathbb{R}$ (see Section 5.2 for detailed description). The nonlinear transformation function $\tanh(\cdot)$ in Eq. (4.49) is applied separately to each component of its vector valued argument. It has the property

$$|\tanh(x) - \tanh(y)| \leq |x - y|\,, \quad \forall x, y \in \mathbb{R}\,, \tag{4.50}$$

which we will use in the following proof[7]. Similar to the continuous case the difference $\boldsymbol{e}_t = \boldsymbol{x}_t - \boldsymbol{x}'_t$ between system and replica is considered with the following Lyapunov function $L(\boldsymbol{e}_t) = \|\boldsymbol{e}_t\|^2$. It follows that

$$\begin{aligned} \Delta L(\boldsymbol{e}_t) &\equiv L(\boldsymbol{e}_t) - L(\boldsymbol{e}_{t-1}) \\ &= \left\|\tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{b}u_t\right) - \tanh\left(\boldsymbol{C}\boldsymbol{x}'_{t-1} + \boldsymbol{b}u_t\right)\right\|^2 - \|\boldsymbol{e}_{t-1}\|^2\,, \\ &\leq \|\boldsymbol{C}\boldsymbol{e}_{t-1}\|^2 - \|\boldsymbol{e}_{t-1}\|^2 \end{aligned} \tag{4.51}$$

with the last inequality the result of Eq. (4.50). If the connection matrix $\boldsymbol{C}$ has a shrinking effect on vectors, i.e. if its norm is smaller one, $\sigma(\boldsymbol{C}) < 1$, it follows that $\Delta L(\boldsymbol{e}_t) < 0$, which proves that the system in Eq. (4.49) is reliable (for detailed discussion see also Section 5.3.1).

4.4.2 Modeling with Lorenz systems

In the last section it was shown that a driven dynamical system has to be reliable in order to use it for dynamical modeling. Two coupling schemes for the Lorenz system were presented that are suitable candidates for such a modeling approach. In this section we want to use the second coupling scheme in Eq. (4.46), which is only slightly modified in the first component

[7] In fact, the transformation function is not restricted to $\tanh(\cdot)$. Any function can be used as long as it complies to the inequality in Eq. (4.50), which is equivalent to the function being Lipschitz with the Lipschitz number 1.

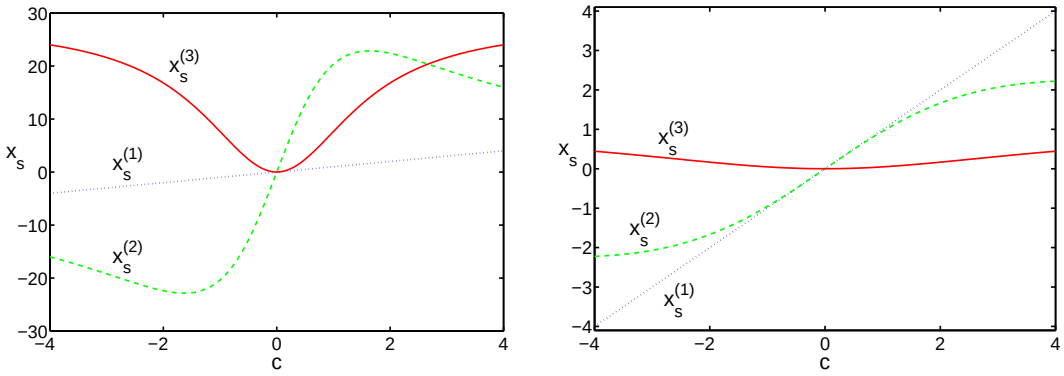

Figure 4.10: Fixed point $\boldsymbol{x}_s$ of Lorenz system in Eq. (4.52) for constant input signals $\{u(t) = c\}_{t\in\mathbb{I}}$ in the range $c \in [-4, 4]$. Changes in the first component of $\boldsymbol{x}_s$ are marked as blue dotted line, second component as green dashed line and third as red line. **Left:** parameter set: $\sigma_1 = 10$, $r_1 = 28$, $b_2 = 2.666$, $\alpha_1 = 1.0$ **Right:** parameter set: $\sigma_2 = 10$, $r_2 = 1$, $b_2 = 20$, $\alpha_2 = 1.0$.

$(u(t) \to \sigma u(t))$ and has an additional parameter $\alpha > 0$, as a basis for developing a dynamical model:

$$\begin{aligned} \dot{x}_1 &= \alpha\sigma(u(t) - x_1) \\ \dot{x}_2 &= \alpha(rx_1 - x_2 - x_1x_3) \\ \dot{x}_3 &= \alpha(x_1x_2 - bx_3) \end{aligned} \quad . \tag{4.52}$$

Note that all numerical simulations in this section were performed with an integration time $t_i = 0.1$, which was also used as the sampling time.

The Lorenz system in Eq. (4.52) is reliable. That means that for a constant input $u(t) = c$, $c \in \mathbb{R}$, the system dynamics settles on a fixed point $\boldsymbol{x}_s$. In Fig. 4.10 the position of this point is shown in relation to the magnitude $c \in [-4, 4]$ of the constant input signal for two different sets of parameters: $\sigma_1 = 10$, $r_1 = 28$, $b_2 = 2.666$, $\alpha_1 = 1.0$ and $\sigma_2 = 10$, $r_2 = 1$, $b_2 = 20$, $\alpha_2 = 1.0$. Leaving out all dynamical effects, i.e. for a very slow dynamics of the input signal $u(t)$, we can say that the first component of the driven Lorenz in Eq. (4.52) simply follows the external input signal, the second component transforms it nonlinearly in a sigmoid fashion, and the third performs a quadratic transformation.

Taking into account also the dynamical effects, we are able to produce different responses to an arbitrary input signal $u(t)$ by changing the free parameters σ, r, b, and α. If we combine the individual response signals $x_i(t)$, $i = 1, 2, 3$, in Eq. (4.52), we can form an output signal $\hat{y}(t)$

$$\hat{y}(t) = \sum_{i=1}^{3} w_i x_i(t) + w_0 \,, \tag{4.53}$$

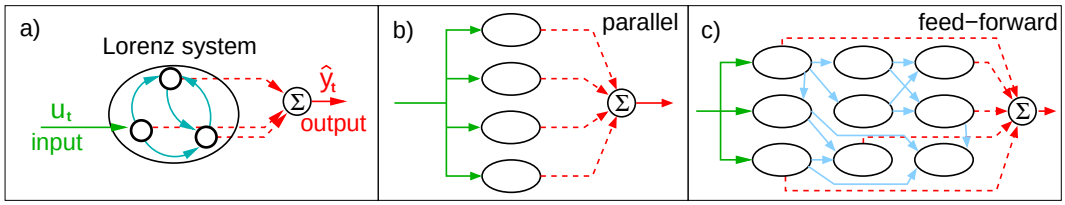

Figure 4.11: Different strategies for building a dynamical model with Lorenz systems. **a)** Driven Lorenz system as a basis module for the three different strategies. The input signal drives the system. The response signals from the three state variables are linearly combined to form the model output. **b)** Parallel strategy: Different Lorenz systems are driven with the same input signal. Their individual outputs (three each) is combined in the model output. There is no interconnection between the individual systems. **c)** Feed-forward strategy: Lorenz systems are arranged in layers. The Lorenz systems can get their input from the systems of all preceding layers. System in the same layer can be connected only in one way, so that no loops occur. Only the first layer is connected to the external input.

with the linear weights $w_i \in \mathbb{R}$, $i = 1, 2, 3$, and the constant offset $w_0 \in \mathbb{R}$ (see Fig. 4.11). In this way the Lorenz system is used as a reservoir or pool of dynamical responses, which is the basic idea of dynamical modeling (see Section 3.3 for details).

Of course a reservoir with three different response signals is not rich enough to be of any use for any but the simplest modeling tasks. An option is to extend the Lorenz system with additional state variables, i.e. to create a high-dimensional nonlinear systems that is able produce a great diversity of response signals to an input. However, in most cases it is practically not possible to guarantee reliability of such systems, as the dynamics becomes more and more complex with each dimension and there is no general rule to find a suitable Lyapunov function[8]. An alternative way is the usage of many Lorenz systems with different parameter sets in parallel (see Fig. 4.11). A finite number M of Lorenz systems represent the dynamical model. All of these systems are driven with the same input signal and their $3M$ response signals are linearly combined in the model output. This dynamical modeling approach is referred to as *parallel strategy*.

The parallel strategy was tested for the following cross-prediction task. The Rössler system

$$\begin{aligned} \dot{z}_1 &= 2 + z_1(z_2 - 4) \\ \dot{z}_2 &= -z_1 - z_3 \,, \\ \dot{z}_3 &= z_2 + 0.45 z_3 \end{aligned} \tag{4.54}$$

was integrated for a time span $t = 1000$ and the first component was used as an input signal to the Lorenz systems $u(t) = z_1(t)$. The desired model output

[8]One of the exceptions are recurrent neural networks, which are discussed in Section 5.2.

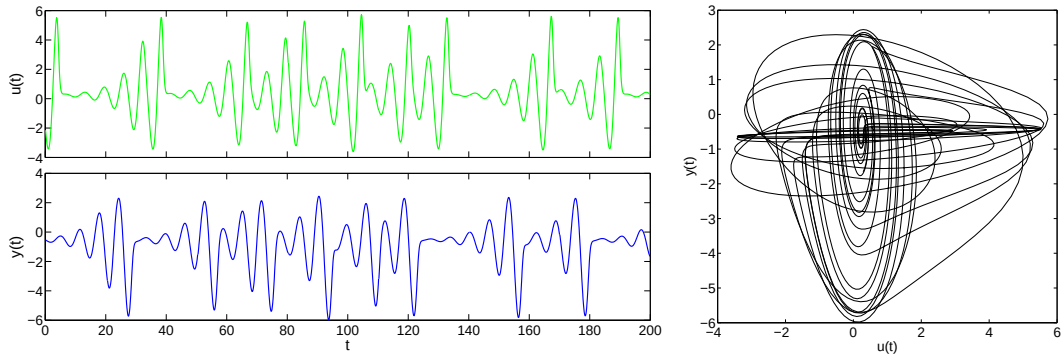

Figure 4.12: The cross-prediction task consists of predicting the second component of the Rössler system in Eq. (4.54) lying a time span $T = 10$ in the future from the current value of the first component. **Left:** A sample of the input time series $u(t) = z_1(t)$ and the output time series $y(t) = z_2(t + 10)$ for the time period $t \in [0, 200]$. With a sampling time of $t_s = 0.1$ each time series has 2000 data points. **Right:** Plotting input against output, a complicated nonlinear relationship becomes visible.

was set to $y(t) = z_2(t+T)$ with $T = 10$, i.e. the model had to perform a cross-prediction from the first component to the second component, lying a time step $T = 10$ in the future (see Fig. 4.12). As can be seen the time span $T = 10$ corresponds roughly to one and a half oscillations of the Rössler system.

The first component $z_1(t)$ of the Rössler system in Eq. (4.54) was used to drive $M = 32$ parallel Lorenz systems, having each different sets of parameters σ_i, r_i, b_i, α_i, $i = 1, \ldots, M$. The output of the model was the weighted sum of the Lorenz state variables

$$\hat{y}(t) = \sum_{i=1}^{M} \sum_{j=1}^{3} w_j^{(i)} x_j^{(i)}(t) + w_0 \,, \tag{4.55}$$

with $x_j^{(i)}$ being the j-th component of the i-th Lorenz system. With a sampling time of $t_s = 0.1$ the integration yielded 10000 data points for the learning phase of the modeling process. The first 1000 data points were discarded as transient, while the last 3000 data points were used as a validation set (see Section A.3). The other 6000 data points were used for training the $3M+1$ weight parameters in Eq. (4.55). This was done by minimizing the mean squared error

$$\text{MSE} = \sum_{t=1001}^{6000} (y_t - \hat{y}_t)^2 \,, \tag{4.56}$$

on the training set, which is a quadratic problem with a simple solution scheme (see Section A.2.1).

Since randomly picked parameter sets of the Lorenz systems were not likely to produce good results, the parameters were additionally optimized in a Simulated Annealing (SA) procedure, similar to the one described in Section 5.3.4.

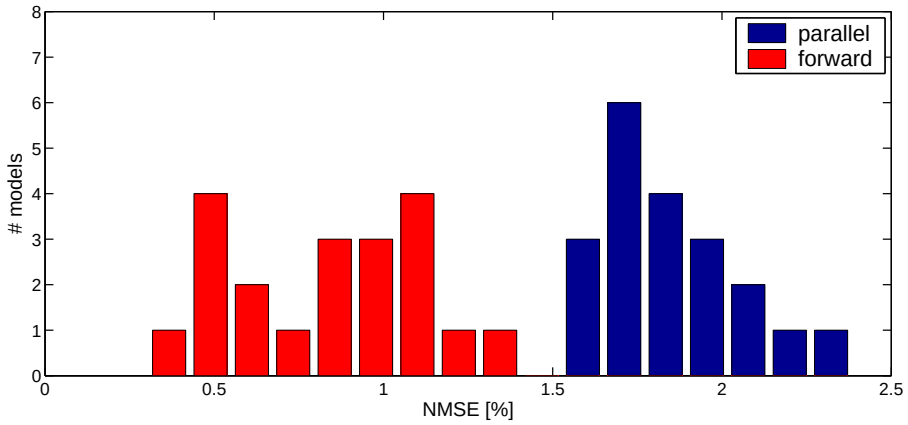

Figure 4.13: Evaluation results of the parallel (left) and the feed-forward strategy (right) on the test set. The NMSE value for 20 different modeling runs for each strategy. The average value for the parallel strategy is NMSE = 1.9. The results of the forward strategy group around NMSE = 0.9.

Random changes were performed iteratively on the $4M$ parameters σ_i, r_i, b_i, and α_i. If these changes led to a lower MSE on the validation set, they were automatically accepted. If they led to a higher MSE, they were accepted with a certain probability p that was lowered from iteration to iteration. The number of iterations was set to 400. The initial parameters were chosen from the uniform distributions: $\sigma_i \sim \mathrm{U}\,[0,\,20]$, $r_i \sim \mathrm{U}\,[0,\,40]$, $b_i \sim \mathrm{U}\,[0,\,10]$, and $\alpha_i \sim \mathrm{U}\,[0.8,\,1.2]$.

After the training process the model was evaluated on a fresh test set consisting of 10000 data points. The quality measure was the normalized mean squared error

$$\mathrm{NMSE} = \frac{100\,\%}{\sigma_y^2} \sum_{t=1001}^{10000} \left(y_t - \hat{y}_t\right)^2 \,, \tag{4.57}$$

with σ_y^2 being the variance of the target signal. Again the first 1000 data points were left out to avoid influence of transient behavior. This modeling procedure was repeated for 20 different models. The results are displayed in Fig. 4.13 (lhs). Although not a perfect model for cross-prediction, the average value of NMSE = 1.9 indicates a result that can be called satisfactory. In Fig. 4.14 an example of a model cross-prediction on the test set is shown for 1000 data points from the time span $t \in [250, 350]$.

The problem with the parallel strategy is that the response signals from the different Lorenz systems do not differ enough. Different parameter sets may cause slower or faster reactions to the input signal and also different transformations (see Fig. 4.10). However, the response signals still show a significant correlation with the input signal. A mechanism is needed that can introduce

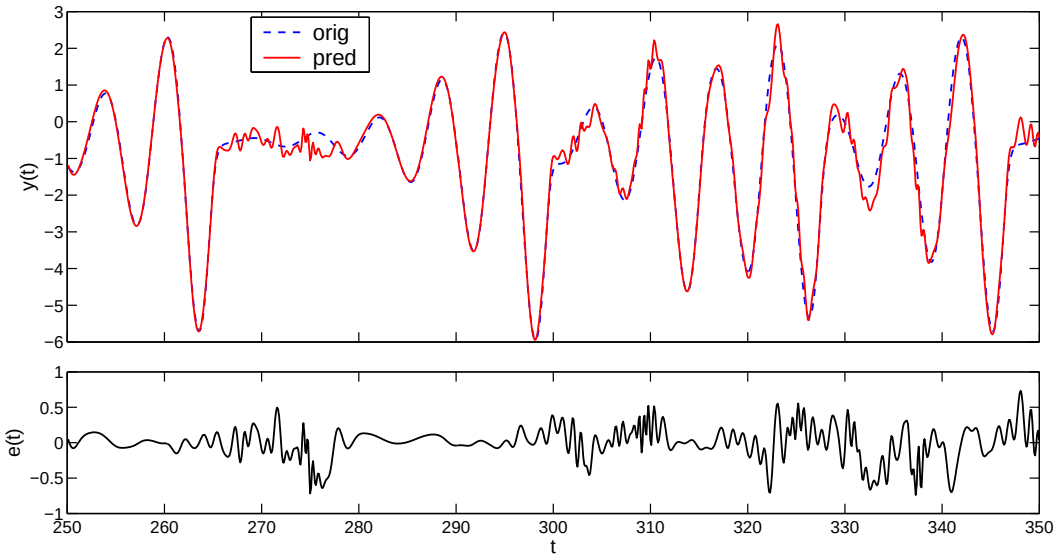

Figure 4.14: The original signal $y(t)$ (blue dashed line) and the predicted signal $\hat{y}(t)$ (red line) of the dynamical model consisting of parallel arranged Lorenz systems for a sample time span $t \in [250, 350]$ taken from the test set. Below, the corresponding model error $e(t) = \hat{y}(t) - y(t)$ is plotted.

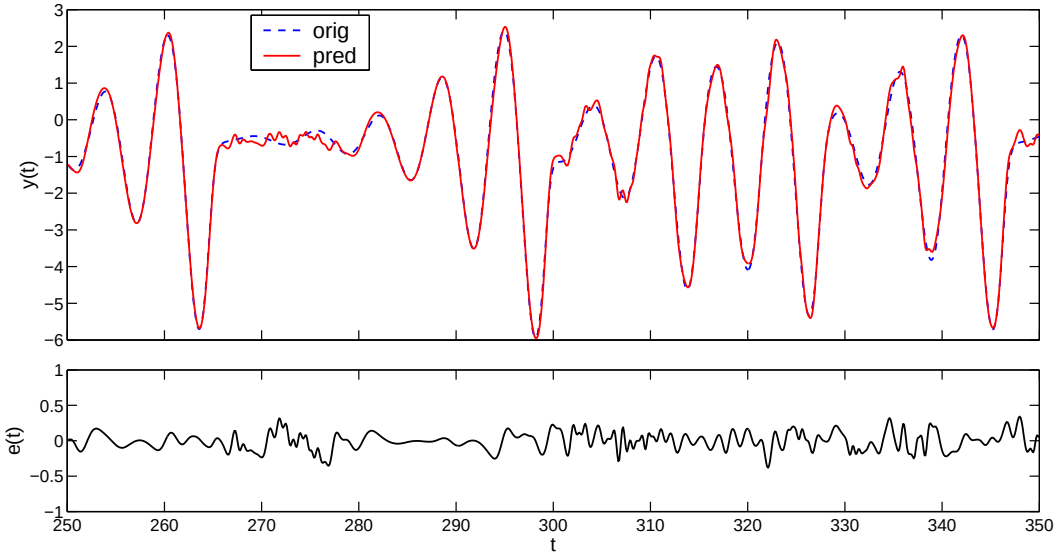

Figure 4.15: The original signal $y(t)$ (blue dashed line) and the predicted signal $\hat{y}(t)$ (red line) are shown for the same time span $t \in [250, 350]$ as in Fig. 4.14. However, the dynamical model employed follows the feed-forward strategy. Below, the corresponding model error $e(t) = \hat{y}(t) - y(t)$ is plotted.

greater delays between the responses and produce internal combinations of them. Such a mechanism can be found by coupling the Lorenz systems instead of using them separately.

We set the output of the Lorenz system in Eq. (4.52) to be

$$\xi^{(i)}(t) = \frac{1}{r_i} x_2^{(i)}(t)\,, \quad i = 1, \ldots, M\,. \tag{4.58}$$

If two systems are uni-directionally coupled to each other, the output of the driving system is used as the input $u(t)$ to the driven system in Eq. (4.52). If two Lorenz systems are coupled to one system, the input $u(t)$ is a weighted sum of the two outputs. More generally the input to the i-th Lorenz system can be written as

$$u^{(i)}(t) = \sum_{j=1}^{M} c_{ij}\xi^{(i)}(t) + b_i u^{\text{ext}}(t)\,, \tag{4.59}$$

with $c_{ij} \in \mathbb{R}$ being the connection weight form the j-th system to the i-th system and $b_i \in \mathbb{R}$ the connection weight to the external input $u^{\text{ext}}(t)$. Since recurrent loops might lead to unreliable systems the coupling scheme has to be restricted to feed-forward connections (see Fig. 4.11). Formally this means that the connection matrix $\boldsymbol{C} \in \mathbb{R}^{M\times M}$ with the elements equal to the connection weights c_{ij} is restricted to a lower left matrix[9].

Similar to the parallel strategy the forward strategy was tested on the same cross-prediction task as described above. Everything was done in the same way as for the parallel strategy except that the $M = 32$ Lorenz systems were arranged in 4 layers with 8 systems each. The systems in the first layer were driven by the external input signal. The other systems were driven by internal coupling signals. The initial coupling scheme was chosen in such a way that each Lorenz system was coupled to exactly one system in a previous layer. The coupling weights were randomly chosen from a uniform distribution in $c_{ij} \in [-1.5, 0.5] \cup [0.5, 1.5]$. Additionally 30 randomly placed connections were introduced with the same uniform distribution and according to the forward scheme.

The SA procedure was also done similarly as in the parallel strategy. However, instead of spending all 400 iteration on optimizing the parameters of the Lorenz systems, only 200 steps were employed. The other 200 iterations were used to optimize the connection matrix $\boldsymbol{C}$. New connections were added, old connections were cut or their value was rescaled. The resulting model was evaluated with the same NMSE criterion from Eq. (4.57). The results for 20 different runs are shown in Fig. 4.13 (rhs). With an average value of NMSE = 0.9 the performance of this network-like model for the cross-prediction task is very good. In Fig. 4.15 an example of the model prediction is shown for the time

[9]If the notations seem too confusing, the reader is advised to read Section 5.1 first, where the coupling structure is explained in a more general but also more systematic way.

sample $t \in [250, 350]$ of the test set. Compared with the parallel strategy, the forward strategy is superior.

Recurrent connections in the connection matrix $\boldsymbol{C}$ would contribute even more to the diversity of the different response signals. However, in the case of coupled Lorenz systems the reliability cannot be guaranteed if the coupling scheme deviates from the feed-forward form. With the Recurrent Neural Networks an example system is presented in Section 5.2 that can employ recurrent loops, because in this case there exists an easy criterion that can ensure reliability.

Chapter 5

Dynamical networks

In the Chapter 2 the concept of dynamical modeling has been introduced to the reader. It was shown that it offers interesting features, which can prove beneficial for some modeling tasks. In this chapter we want to shift our attention to the types of models that can be employed for dynamical modeling. It was already mentioned in Chapter 4 that it is advantageous to use models, comprising elements that are coupled to each other, because they strike a good balance between simplicity and complexity, i.e. the diversity of the internal responses is high but reliability can still be ensured. They are able to exhibit a very complex behavior even if the included elements themselves have a simple dynamics. Since this is exactly the property that makes them interesting for dynamical modeling, the focus of this chapter are models with an internal network structure. We call these models *dynamical networks*, referring to the internal dynamics of the individual elements and the network-like coupling scheme[1].

In Section 5.1 we start by introducing the general structure of dynamical networks. The reader is familiarized with notation issues and some useful terms. In Section 5.2 the most prominent representatives of dynamical networks are presented: Recurrent Neural Networks. As a dynamical extension of static Feedforward Neural Networks, they are much more versatile but also much more complex to handle. Because of this complexity the simpler Feedforward Neural Networks still enjoy a greater popularity, dominating in applications and scientific publications. Nevertheless, much has been done to change the neglected status of Recurrent Neural Networks. The most important ideas are summarized in Section 5.2. Some of these ideas are picked up in Section 5.3 where we take a closer look at the application of recurrent networks for modeling. In numerical experiments strengths and weaknesses of recurrent networks are revealed and some techniques are introduced that can improve

[1] In the literature the term *dynamical network* is sometimes applied to artificial neural networks with a connection matrix that is not constant in time. However, this usage is not adopted in this thesis.

the performance of these networks.

5.1 General structure of dynamical networks

Dynamical networks have many different realizations. There are rings of logistic maps, chains of Lorenz Oscillators, recurrent neural networks, and so on. Despite their obvious differences, which lie mostly in the internal dynamics of the elements, these systems have a coupling structure that can be described in a common framework. All of the mentioned models represent dynamical networks, and the network-like structure is the same for every one of them. The purpose of this section is to show the common background of dynamical networks and to introduce a compact notation for further usage.

5.1.1 Networks with multi-dimensional elements

Consider a dynamical model with M elements. Every element has an internal state that can be described by a finite number of real values, merged into one *state vector*. If the state vector has just one component, the element is called *one-dimensional*, otherwise it is called *multi-dimensional*. The update equation for the state vector $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^{d^{(i)}}$ of the $d^{(i)}$-dimensional i-th element[2] at time step $t \in \mathbb{I} \subset \mathbb{N}$ can generally be written as

$$\boldsymbol{x}_t^{(i)} = \boldsymbol{f}^{(i)} \left(\boldsymbol{x}_{t-1}^{(i)}, I_t^{(i)} \right) , \quad i = 1, \ldots, M,\ t \in \mathbb{I} , \tag{5.1}$$

with $I_t^{(i)} \in \mathbb{R}$ being a value of the scalar *internal input signal*, $\{I_t^{(i)}\}_{t \in \mathbb{I}}$, which drives the i-th element (see also Fig. 5.1). The *transfer* or *update function* $\boldsymbol{f}^{(i)} : \mathbb{R}^{d^{(i)}} \times \mathbb{R} \to \mathbb{R}^{d^{(i)}}$ in Eq. (5.1) is in most cases a nonlinear function. In continuous models the elements change their states not abruptly in discrete time steps but continuously, as defined by the *vector field*. Instead of the difference equations in Eq. (5.1) we have ordinary differential equations (ODEs)

$$\dot{\boldsymbol{x}}^{(i)}(t) = \boldsymbol{f}^{(i)} \left(\boldsymbol{x}^{(i)}(t), I^{(i)}(t) \right) , \quad i = 1, \ldots, M,\ t \in \mathbb{I} \subset \mathbb{R} , \tag{5.2}$$

with $\boldsymbol{f}^{(i)} : \mathbb{R}^{d^{(i)}} \times \mathbb{R} \to \mathbb{R}^{d^{(i)}}$, here playing the role of a vector field.

All elements produce *internal output signals*, which are functions of their state variables

$$z_t^{(i)} = g^{(i)} \left(\boldsymbol{x}_t^{(i)} \right) , \quad i = 1, \ldots, M,\ t \in \mathbb{I} , \tag{5.3}$$

and for continuous networks

$$z^{(i)}(t) = g^{(i)} \left(\boldsymbol{x}^{(i)}(t) \right) , \quad i = 1, \ldots, M,\ t \in \mathbb{I} . \tag{5.4}$$

[2] In this section an index in parenthesis, like $^{(i)}$, is being used to indicate that the indexed variable is associated with the i-th element.

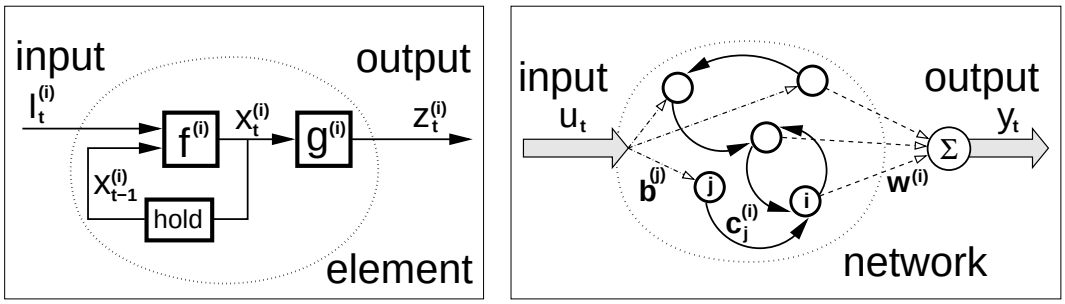

Figure 5.1: Left: Driven by the internal input signal $\{I_t^{(i)}\}_{t\in\mathbb{I}}$, the i-th element changes its state $\boldsymbol{x}_t^{(i)}$ according to Eq. (5.1) and produces an output $z_t^{(i)}$ according to Eq. (5.3). **Right:** The network consists of elements that are mutually connected by weights $c_j^{(i)}$. It is driven by the external input signal $\{u_t\}_{t\in\mathbb{I}}$ and produces an output signal $\{y_t\}_{t\in\mathbb{I}}$ according to Eq. (5.7).

with a possibly nonlinear *output function* $g^{(i)} : \mathbb{R}^{d^{(i)}} \to \mathbb{R}$. For even greater generality the internal output signals could also be chosen multi-dimensional. However, this would imply many parallel connection paths between the elements, which would complicate the notation in an unnecessary way. For simplification we consider only the one-dimensional case here.

The elements of dynamical networks are internally connected to each other. That means that an element can receive internal output signals of other elements and vice versa send its own output signal to them. All signals, influencing the i-th element, are summed up in the *internal input signal* $\{I_t^{(i)}\}_{t\in\mathbb{I}}$ from Eq. (5.1). It consist of the following addends

$$I_t^{(i)} = \sum_{k=1}^{M} c_k^{(i)} z_{t-1}^{(k)} + c_0^{(i)} + b^{(i)} u_t \,, \quad i = 1, \ldots, M,\ t \in \mathbb{I}\,. \tag{5.5}$$

Again, in the continuous case the analogy to Eq. (5.5) can simply be produced by exchanging the discrete time series for the continuous signals. The parameters $c_j^{(i)} \in \mathbb{R}$, $i, j = 1, \ldots, M$, in Eq. (5.5) are called *internal connection parameters*. Specifically, $c_j^{(i)}$ represents a connection from the j-th element to the i-th element. A nonzero value of $c_j^{(i)}$ means that the output $z_{t-1}^{(j)}$ of the j-th element contributes to the internal input $I_t^{(i)}$ of the i-th element and thus influences the update of the latter's state[3]. The magnitude of $c_j^{(i)}$ reveals the strength of the unidirectional influence. A special role is reserved for parameter $c_0^{(i)} \in \mathbb{R}$, $i = 1, \ldots, M$. It is usually referred to as the *bias* of the i-th element and can be regarded as the connection strength to a virtual, constant signal $\{z_t^{(0)}\}_{t\in\mathbb{I}}$, with $z_t^{(0)} = 1$, $\forall t \in \mathbb{I}$.

[3]This includes also possible *auto-connections* or *self-connections* for $j = i$.

Similar to the internal connections, the parameters $b^{(i)} \in \mathbb{R}$, $i = 1, \ldots, M$ in Eq. (5.1) represent connections of the elements to the *external input signal* $\{u_t\}_{t\in\mathbb{I}}$, which drives the dynamical model. In case the *external connection parameter* $b^{(i)}$ is nonzero the state $\boldsymbol{x}^{(i)}$ of the i-th element is influenced by the external signal. In case the external input signal is multi-valued with p different channels, $\boldsymbol{u}_t = (u_{1t}, \ldots, u_{pt})^T$, the external connection parameter is a vector-valued parameter, $\boldsymbol{b}^{(i)} = (b_1^{(i)}, \ldots, b_p^{(i)})^T$, with an extra component for every input channel.

The *external output* of a dynamical model is a function of its internal states

$$y_t = g_{\text{out}}\left(\boldsymbol{x}_t^{(1)}, \ldots, \boldsymbol{x}_t^{(M)}\right) \quad \text{or} \quad y(t) = g_{\text{out}}\left(\boldsymbol{x}^{(1)}(t), \ldots, \boldsymbol{x}^{(M)}(t)\right) . \tag{5.6}$$

If direct links from the external input to the external output are necessary, they can also be included into the output function $g_{\text{out}}(\cdot)$. A typical output function is a simple weighted sum over all state variables of the network

$$y_t = \sum_{i=1}^{M} (\boldsymbol{w}^{(i)})^T \cdot \boldsymbol{x}_t^{(i)} + w_0 \,, \tag{5.7}$$

with *output parameters* $\boldsymbol{w}^{(i)} = \left(w_1^{(i)}, \ldots, w_{d^{(i)}}^{(i)}\right)^T \in \mathbb{R}^{d^{(i)}}$ for every element and an additional bias value $w_0 \in \mathbb{R}$ that represents a constant offset in the output signal. If the external output is multi-valued, $\boldsymbol{y}_t = (y_{1t}, \ldots, y_{qt})^T$, Eq. (5.7) applies to every output channel with an extra set of output parameters $\boldsymbol{w}_j^{(i)}$, $i = 1, \ldots, M$, $j = 1, \ldots, q$.

5.1.2 Networks with one-dimensional elements

In Section 5.1.1 networks with multi-dimensional elements are described on a very general level to include an extensive class of dynamical models with a network-like structure and point out common ideas and properties of its members. In this way, different models, like rings of logistic maps or chains of Chua oscillators, fit in the introduced scheme. However, due to generalization the presented equations lack a certain compactness and, in their current form, are cumbersome for further usage. In this section we loosen our claim for generality and state the equations of dynamical networks for the prevalent case of discrete networks with one-dimensional elements and the usual settings that are used in this thesis.

For one-dimensional elements, the update equations in Eq. (5.1) can be written as

$$x_t^{(i)} = f^{(i)}(x_{t-1}^{(i)}, I_t^{(i)}) \,, \quad i = 1, \ldots, M, \; t \in \mathbb{I} \,, \tag{5.8}$$

with the nonlinear transfer function $f^{(i)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. In many cases the transfer function depends only on the internal input signal

$$f^{(i)}(x_{t-1}^{(i)}, I_t^{(i)}) = h^{(i)}(I_t^{(i)}) \,, \quad i = 1, \ldots, M, \; t \in \mathbb{I} \,, \tag{5.9}$$

with a so called *transformation function* $h^{(i)} : \mathbb{R} \to \mathbb{R}$, which transforms the internal input signal $\{I_t^{(i)}\}_{t\in\mathbb{I}}$ in a nonlinear way. For example, in artificial neural networks this function is usually sigmoid, like $\tanh(\cdot)$. It limits the amplitude of the input signal and is often referred to as *squashing function*.

Elements whose update function is represented by a transfer function as in Eq. (5.9) are able to describe time series from discrete systems very well but sometimes lack to capture smoother dynamics. For continuous signals the transformation function in Eq. (5.9) can be augmented by an additional term

$$f^{(i)}(x_{t-1}^{(i)}, I_t^{(i)}) = a^{(i)} x_{t-1}^{(i)} + h^{(i)}(I_t^{(i)}), \quad i = 1, \ldots, M,\ t \in \mathbb{I}, \tag{5.10}$$

with the so-called *relaxation parameter* $a^{(i)} \in (-1, 1) \subset \mathbb{R}$. For positive relaxation parameters, $a^{(i)} > 0$, the additional term in Eq. (5.10) has an averaging effect on the dynamics of the element, producing a smoother state transition. For negative values, $a^{(i)} < 0$, the dynamics can be made even more erratic. In this way the relaxation parameter offers a leverage for the smoothness of the element dynamics.

For multi-dimensional elements the output function $g^{(i)} : \mathbb{R}^{d^{(i)}} \to \mathbb{R}$ of the i-th element is a function of its $d^{(i)}$ state variables. Usually it is chosen to produce linear combinations of the $d^{(i)}$ state variables, reducing in this way a multi-dimensional signal to a scalar signal that can be distributed among the other elements. This task is superfluous for one-dimensional elements. Here, the only useful task of output functions $g^{(i)}(\cdot)$ is to transform the scalar states of the elements. However, since nonlinear transformation of internal signals is already performed by the functions $h^{(i)}$, the application of the output functions $g^{(i)}(\cdot)$ is mostly redundant. For this reason, output functions are not used[4], setting them effectively equal to identical functions, $g^{(i)}(\cdot) = \mathrm{id}(\cdot)$, $i = 1, \ldots, M$. In this way the output of a one-dimensional element is equal to its state variable

$$z_t^{(i)} = x_t^{(i)}, \quad i = 1, \ldots, M,\ t \in \mathbb{I}, \tag{5.11}$$

[4] In some publications the roles of the functions $g^{(i)}(\cdot)$ and $h^{(i)}(\cdot)$ are reversed. The output functions $g^{(i)}(\cdot)$ are responsible for internal transformations and the transfer functions $h^{(i)}(\cdot)$ are not used. In this case the update equations read

$$x_t^{(i)} = \sum_j c_j^{(i)} g^{(i)}(x_{t-1}^{(j)}),$$

instead of

$$x_t^{(i)} = h^{(i)}\Big(\sum_j c_j^{(i)} x_{t-1}^{(j)}\Big).$$

For invertible functions it can be shown that both forms are equivalent [33]. However, this does not apply to arbitrary functions.

and the internal input signal from Eq. (5.5) transfers to

$$I_t^{(i)} = \sum_{k=1}^{M} c_k^{(i)} x_{t-1}^{(k)} + c_0^{(i)} + b^{(i)} u_t \,, \quad i = 1, \ldots, M,\ t \in \mathbb{I} \,, \tag{5.12}$$

which we can express without usage of the internal output signals $\{z_t^{(i)}\}_{t\in\mathbb{I}}$.

The equations Eq. (5.10) and Eq. (5.12) completely describe the deterministic behavior of driven networks with one-dimensional elements. However, we can now derive a very compact matrix notation. Since M one-dimensional elements are used, the state of the whole network is defined in one M-dimensional vector $\boldsymbol{x}_t = (x_t^{(1)}, \ldots, x_t^{(M)})^T$, which includes all individual states of the elements as components. Taking the before mentioned specifications into account, the update equation for the network state can be written in a compact matrix form

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \vec{\boldsymbol{h}}(\boldsymbol{I}_t) \,, \quad t \in \mathbb{I} \,, \tag{5.13}$$

with the *relaxation matrix* $\boldsymbol{A}$ being a diagonal matrix with the relaxation parameters $a^{(i)}$ in the diagonal. The vector-valued input signal is defined as

$$\boldsymbol{I}_t = (I_t^{(1)}, \ldots, I_t^{(M)})^T = \boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t \,, \quad t \in \mathbb{I} \,, \tag{5.14}$$

with the *connection matrix* $\boldsymbol{C}$, whose entries equal the connection parameters, $C_{ij} = c_j^{(i)}$. The *bias vector* $\boldsymbol{c}_0 = (c_0^{(1)}, \ldots, c_0^{(M)})^T$, and the *external connection vector* $\boldsymbol{b} = (b^{(1)}, \ldots, b^{(M)})^T$ are similarly defined by the corresponding parameters from Eq. (5.12). The arrow above the vector-valued transformation function $\vec{\boldsymbol{h}}(\cdot)$ in Eq. (5.13) is a special abbreviation in the notation. It denotes that each transformation function $h^{(i)} : \mathbb{R} \to \mathbb{R}$ in $\boldsymbol{h} = (h^{(1)}, \ldots, h^{(M)})$ is applied individually to the corresponding component of the vector-valued argument, i.e. $h^{(1)}$ is applied to $I_t^{(1)}$, $h^{(2)}$ to $I_t^{(2)}$, and so on. If all transformation functions are equal, $h^{(i)}(\cdot) = h(\cdot)$, $i = 1, \ldots, M$, we write Eq. (5.13) as

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + h(\boldsymbol{I}_t) \,, \quad t \in \mathbb{I} \,, \tag{5.15}$$

which means that function $h : \mathbb{R} \to \mathbb{R}$ is applied to each component of its vector valued argument separately.

The output of the network in Eq. (5.7) reduces to a simple linear sum

$$y_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 = \sum_{i=1}^{M} w^{(i)} x_t^{(i)} + w_0 \,, \tag{5.16}$$

with the output coefficients $\boldsymbol{w} = (w^{(1)}, \ldots, w^{(M)})^T$ and the bias $w_0 \in \mathbb{R}$.

5.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are the best example of models with internal dynamics and a network-like structure. Formally an extension to Feedforward Neural Networks (FNNs), they transcend the latter in their ability to approximate the behavior of diverse systems. Their internal memory makes RNNs well suited for the identification of patterns in time as well as in space and is the basis for their computational superiority. However, the downside of the internal dynamics is that RNNs are difficult to handle in applications. In contrast to simple input-output models, like FNNs, they represent full dynamical systems on their own. Thus, issues like stability have to be dealt with.

There are many different approaches to RNNs and a complete review of them would probably fill many books. Coarsely, RNNs can be divided in two classes: the ones with a constant input and the ones with a time varying input [31]. The former class is suitable for static modeling tasks like pattern recognition, where the RNNs are used as associative memory. The latter class is applied to temporal processing of input-output relationships and is suited for dynamical problems like prediction or simulation (see Section 2 for detailed definitions). Since this work centers around dynamical applications, only the second class of RNNs is of interest to us. Therefore relaxation networks like Hopfield networks or Boltzman machines, which represent a great part of the usual RNN literature, will not be discussed here (for a broad review see [31]).

5.2.1 Elman/Jordan Networks

Coming from the field of automatic language processing, Jordan developed one of the earliest RNNs in 1986 [39], followed by Elman in 1990 with a slightly different approach [25]. Both network types are named after their creators; and today, Elman and Jordan networks rank among the classical architectures of RNNs. Their internal structure only slightly deviates from the traditional structure of FNNs. Like an ordinary feedforward network, Elman and Jordan networks consist of one input layer, one hidden layer, and one output layer (see Fig. 5.2). However, what sets them apart is an additional layer, which is called the *context layer*. The context layer is used as an internal memory that (partially) stores the previous state of the network. Every time step the context layer provides its contents as an additional input next to the external input. Thus, every external input value is evaluated in the context of the previous state of the network.

The difference between Elman and Jordan networks lies in what part of the previous state of the network is stored. The context layer of Elman networks stores the contents of the hidden layer while the context layer of Jordan networks stores the contents of the output layer. Also, all neurons in the context

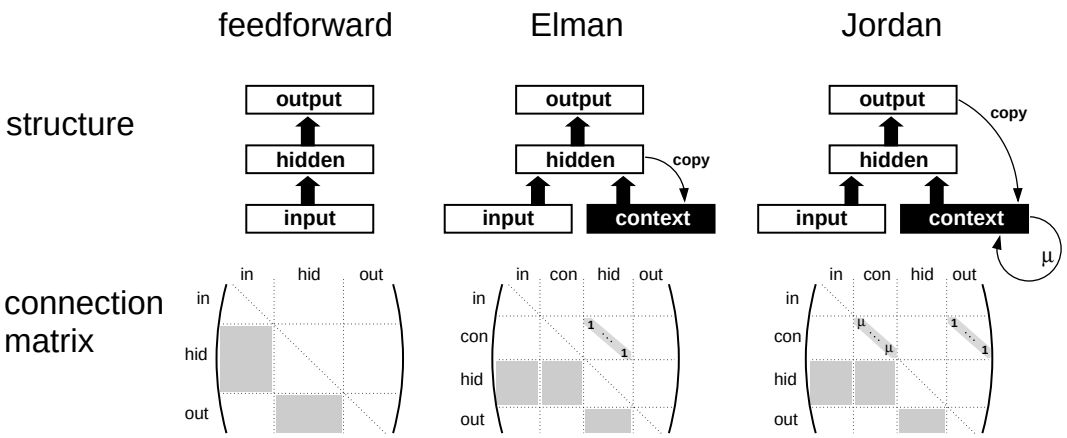

Figure 5.2: Left: Feedforward structure with the input layer connected to the hidden layer and the hidden layer connected to the output layer. The connection matrix has non-vanishing entries (marked as tinged areas) only in the lower left below the diagonal. **Middle:** The Elman network has an additional *context* layer, which is an exact copy of the previous output of the hidden layer. The connection matrix has entries above the diagonal, marking it as a recurrent connection matrix, where the hidden neurons are connected to the context neurons with weights equal one. **Right:** The Jordan network is similar to the Elman network. However, the context layer is a copy of the previous contents of the output layer. The connection matrix indicates connections with weights equal one from the output neurons to the context neurons. Additionally, the context neurons have a self reference with weights equal $\mu \in [0, 1)$.

layer of Jordan networks have additionally a feedback to themselves, rendering them effectively smoothing filters for the network output. Both network types can be easily described in our notation from Section 5.1.2 as they are only rigorous restrictions of the generalized dynamical networks. Assumed the network has M_{in} input neurons, M_{con} context neurons, M_{hid} hidden neurons, and M_{out} output neurons, with $M = M_{\text{in}} + M_{\text{con}} + M_{\text{hid}} + M_{\text{out}}$, the neurons are arranged according to their function. Then the external connection vector $\boldsymbol{b}$ and the internal connection matrix $\boldsymbol{C}$ in Eq. (5.14) are restricted as follows. The external connection vector has non-vanishing entries only for the input elements

$$\boldsymbol{b} = (b_1, \ldots, b_{M_{\text{in}}}, 0, \ldots, 0)^T \ . \tag{5.17}$$

The connection matrix $\boldsymbol{C}$ of both types of networks has the typical structure as schematically depicted in Fig. 5.2. It has roughly the lower left structure of a feedforward network with additional blocks above the diagonal, which indicate connections from hidden (output) neurons to context neurons. Since the context layer is merely a copy of the hidden (output) layer, the additional blocks are diagonal with entries equal one.

Because of the rigid rules that restrict the internal recurrent connections to a specified set, Elman and Jordan networks belong to the class of *partially recurrent networks*. Typically, the weights of the recurrent connections are

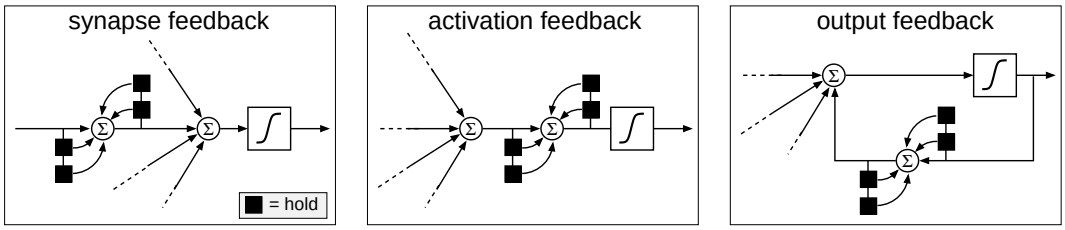

Figure 5.3: Three basic methods of local feedback. Depending on the position of the linear filter they are called *local activation feedback*, *local synapse feedback*, or *local output feedback*. (Figure freely drawn after [49].)

set to a constant value and are not changed during the learning phase. The relatively rigid internal structure of both network types allows for a simpler training procedure. However, the advantage of an easier training is often paid for by a lack of variety in the internal dynamics. *Fully recurrent networks* are much harder to train. However, their internal dynamics is in most cases much richer.

5.2.2 Locally recurrent globally forward

Another approach of bringing recurrent elements into former feedforward structures of neural networks were the so called *Locally Recurrent Globally Forward Networks* (LRGFN), proposed by Tsoi and Back in [68]. The main feature of these networks is that the recurrent loops are hidden in the dynamics of the individual elements. On a global level all elements are connected in a feedforward scheme, meaning that the output of an element does not influence other elements that lie in the same or in a preceding layer. However, on the local level feedback is present. This is achieved by introducing an internal memory and dynamics to the elements, making them more than passive nonlinear transformation functions.

The local feedback is implemented as linearly filtered (FIR or IIR) versions of the internal signals [68]. Depending on where the linear filter is placed one can distinguish three different methods (see Fig. 5.3). If the internal signals $\{x_t^{(i)}\}_{t\in\mathbb{I}}$ are filtered before they are summed up to the elemental input $\{I_t^{(i)}\}_{t\in\mathbb{I}}$, the method is called *local synapse feedback*. If the signals $\{I_t^{(i)}\}_{t\in\mathbb{I}}$ are filtered before they are nonlinearly transformed, the method is called *local activation feedback*. The last method is called *local output feedback*. Here a filtered version of its own output signal $\{x_t^{(i)}\}_{t\in\mathbb{I}}$ is fed back to the element.

The advantage of LRGFNs is that the stability of the network can be easily ensured. If all linear filters are stable, then the same is valid for the whole network [49]. In contrast to pure feedforward architectures the internal dynamics of LRGFNs allows them to store information for long periods of time, making the latter ideal for time processing tasks. However, similar to

partially connected networks, the emphasis on stability limits the range of possible responses of the network to externally applied driving signals.

Note that in a sense the coupled Lorenz oscillators that were used for modeling in Section 4.4.2 are continuous LRGFNs. There we exploited the stability property of individual driven Lorenz oscillators for building a stable network. In analogy to LRGFNs the global feedforward coupling scheme ensured us the stability of the whole system.

5.2.3 Echo State Networks

In contrast to the classical Elman and Jordan networks, *fully recurrent networks* are not restricted in their choice of recurrent connections. This freedom of choice comes at a high cost, though. Since the structure does not follow specific rules, the training of a fully recurrent network is much more complicated. Williams and Zipser presented in [69] one of the first learning algorithms that was capable of adjusting a fully recurrent architecture to a specific task. Later more algorithms were added with the Backpropagation Through Time and Realtime Recurrent Learning the most popular among them [31]. Every one of these approaches suffers from being computationally intensive and from frequently producing suboptimal solutions. Another drawback of fully recurrent networks is that stability is not automatically ensured by the training procedure. Therefore, although these networks have very good approximation abilities, they are usually avoided in applications due to their complexity.

Recently, a new modeling approach was presented by Jaeger in [38] (see also [36], [37]) that tries to overcome the weaknesses of the RNNs concerning training[5]. The modeling technique is referred to as *Echo State Networks* (ESNs). The naming is a little misleading in implying a new sort of network even though the internal dynamics of the neurons in ESNs are still the same as in simple sigmoid RNNs. The difference to former approaches lies rather in the training method, which will be described below.

The state equations of an ESN, consisting of M neurons and driven by a one-dimensional signal $\{u_t\}_{t\in\mathbb{I}}$, can be written in our notation as

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \tanh(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t)\,, \tag{5.18}$$

with the state vector $\boldsymbol{x}_t \in \mathbb{R}^M$. The relaxation matrix $\boldsymbol{A}$ is a diagonal matrix with the diagonal elements $a_{ii} \in [0,1)$, $i = \{1,\ldots,M\}$. Every element has its own bias value, subsumed in the vector $\boldsymbol{c}_0 \in \mathbb{R}^M$. If the input signal is k-dimensional, i.e. $\boldsymbol{u}_t \in \mathbb{R}^k$ with $k > 1$, the input coefficients are expressed as a matrix $\boldsymbol{B} \in \mathbb{R}^{M\times k}$ instead of a vector $\boldsymbol{b} \in \mathbb{R}^M$.

[5]Parallel to and independent from Jaeger's work, Maass et al. proposed in [48] a similar approach called *Liquid State Machines*. The difference between the two methods lies in the type of networks they are applied to, which are simple amplitude RNNs for Jaeger and spiking RNNs for Maass.

The Echo State approach involves randomly creating sparsely connected RNNs and using their state variables in Eq. (5.18) as reservoirs of different dynamics that can be superimposed to model any desired functional relationship between the input and the output signal. The network output is represented by the dynamics of the output neuron

$$y_t = g_{\text{out}}\left(\sum_{i=1}^{M} w_i x_t^{(i)} + w_0\right) = g_{\text{out}}\left(\boldsymbol{w}^T \boldsymbol{x}_t + w_0\right) , \tag{5.19}$$

with $x_t^{(i)}$ the i-th component of state vector $\boldsymbol{x}_t$ in Eq. (5.18) and the transformation function $g_{\text{out}}(\cdot)$ being either linear or sigmoid. A constant offset in the output is formally added as an extra weight $w_0 \in \mathbb{R}$. If the output of the network is d-dimensional, i.e. $\boldsymbol{y}_t \in \mathbb{R}^d$ with $d > 1$, the output weights form a matrix $\boldsymbol{W} \in \mathbb{R}^{M \times d}$ instead of a vector $\boldsymbol{w} \in \mathbb{R}^M$.

The main idea of ESNs is to leave all internal parameters, e.g. the connection matrix $\boldsymbol{C}$ or relaxation matrix $\boldsymbol{A}$, untouched during the training phase. In contrast to conventional approaches these parameters are randomly generated and held fixed at their initial values. Only the output weights w_i, $i = 0, 1, \ldots, M$, from Eq. (5.19) are adapted to the training data. In this way the modeling procedure is simplified to a quadratic optimization problem, making the optimal solution computable by some simple matrix manipulations (see Section A.2.1).

The price that has to be paid for the minimalistic training efforts of ESNs lies in the large sizes of the networks. In order to create a diversity of elemental dynamics in Eq. (5.18) the number of neurons has to be chosen sufficiently high. It is characteristic for ESNs to have more than 100 neurons, reaching even 1000 neurons [38]. These are numbers, which would prove more than problematic for usual training methods like Backpropagation Through Time.

Since the internal parameters are not adapted, the stability and reliability of ESNs has to be ensured in some other way. Otherwise, the dynamics could become unreliable as in any other complex dynamical system, which would rend the network useless for modeling purposes. The stability of ESNs is enforced by scaling the internal connection matrix $\boldsymbol{C}$ and the relaxation matrix $\boldsymbol{A}$ in Eq. (5.18) in an appropriate way [36] (see also Section 5.3.1). Jaeger refers to RNNs whose matrices comply to the scaling conditions as ESNs [36].

In a special mode of ESNs the output of recurrent networks is coupled back by an external feedback loop. In this case the state equations read

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \tanh(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t + \boldsymbol{c}^{\text{back}} y_{t-1}) , \tag{5.20}$$

with the feedback coefficients $\boldsymbol{c}^{\text{back}} \in \mathbb{R}^M$ for one-dimensional output and $\boldsymbol{C}^{\text{back}} \in \mathbb{R}^{M \times d}$ for d-dimensional output. We refer to this mode as the *external mode* of the ESNs to set it apart from the *internal mode*, where $\boldsymbol{c}^{\text{back}} = \boldsymbol{0}$.

It is important to differentiate between the two modes. While the internal mode guarantees stability , the external mode does not. We will discuss the implications in more detail in Section 5.3.2.

5.3 Practical aspects of RNNs

The idea that was presented with the Echo State Networks (ESNs) in [38] is very fascinating. A randomly and sparsely connected network is able to perform complex computations simply by combining the diverse response signals of the individual elements. Compared to the previous approaches concerning Recurrent Neural Networks (RNNs), which typically involve the optimization of every internal connection in computationally intensive procedures like RTRL or BPTT[6], the ESN approach seems to involve only minor computational efforts. Not only that, Jaeger showed in [38] that ESNs can well compete with the canonical RNNs and even outperform them.

In this section we want to explore the ESN approach since it fits seamlessly into the scope of using generalized synchronization for modeling. One of the major concerns in applications is the stability of models. Although stability is also the key concept of ESNs, it is treated rather shortly by Jaeger. For this reason we examine this aspect in greater detail and try to clarify some of its implications. Furthermore we show that the strategy of ESNs to use random connections, although intriguingly simple, is often not adequate to produce good models. Some techniques are presented that can considerably improve the performance of ESNs.

A mixture between ESNs and canonical RNNs is explored, which tries to combine the advantage of both approaches. The main idea of this combination is to use sparsely connected networks and to adapt them to specific tasks. However, instead of using a computationally expensive optimization routine like BPTT the diversity of the internal dynamics is exploited for the adaptation process. We argue that since the ESNs have a rich internal dynamics from the start, they do not depend on intensive optimization methods that are based on gradient descend. Rather some minor structure changes, like adding, cutting, or rewiring some of the connections, are oftentimes sufficient to adapt the RNNs to the modeling objectives.

The proposed RNNs are a mixture between canonical RNNs and ESNs because they are adapted to modeling tasks like the former but exploit at the same time the rich internal dynamics like the latter. Since they depend on a comprehensive reservoir of different internal signals the number of elements in the new RNNs is typically greater than in canonical RNNs. However, because of the adaptation process this number can be usually much smaller than

[6]Real Time Recurrent Learning and Backpropagation Through Time are the standard methods for training RNNs [31].

in ESNs in order to achieve comparable performances, making the combined approach better suited for time critical applications.

5.3.1 Stability revised

The Echo State approach is based on random connection matrices. Since ESNs are fully recurrent, special attention has to be paid to their stability properties. Jaeger provides in [36] stability conditions for the ENS. The stability of an ESN is guaranteed if its connection matrix $\boldsymbol{C}$ has a norm smaller one:

$$\sigma(\boldsymbol{C}) < 1\,, \tag{5.21}$$

where $\sigma(\boldsymbol{C})$ denotes the largest singular value or the 2-norm of matrix $\boldsymbol{C}$. This is valid in the global sense, i.e. a network with the state equations

$$\boldsymbol{x}_t = \tanh(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{b}u_t) \tag{5.22}$$

may start at arbitrary initial states $\boldsymbol{x}_0 \in \mathbb{R}^d$ and $\boldsymbol{x}'_0 \in \mathbb{R}^d$ and nevertheless arrive at the same trajectory

$$\lim_{t\to\infty} \|\boldsymbol{x}_t - \boldsymbol{x}'_t\| = 0\,, \tag{5.23}$$

given that it is driven by the same input signal $\{u_t\}_{t\in\mathbb{I}}$. This is what we call *reliable* because the model answers always in the same way to an external input after a finite transient phase (see Section 4.4.1). On the other hand if

$$\rho(\boldsymbol{C}) > 1\,, \tag{5.24}$$

is valid, with $\rho(\boldsymbol{C})$ denoting the spectral radius[7] of $\boldsymbol{C}$, it can be proved that the dynamics in Eq. (5.18) is unreliable.

According to Jaeger the interesting regime of scaling lies between these two cases. Numerical observations suggest that the condition in Eq. (5.21) is too strict. Jaeger mentions that he experimentally found

$$\rho(\boldsymbol{C}) < 1\,, \tag{5.25}$$

to be a sufficient condition for stability [36]. In practice the stability conditions can be ensured for all possible connection matrices $\boldsymbol{C}$. Since both values $\sigma(\boldsymbol{C})$ and $\rho(\boldsymbol{C})$ scale linearly with a multiplicative factor in the connection matrix, the stability conditions are fulfilled by using a modified connection matrix $\boldsymbol{C}^{\text{new}} = \alpha \cdot \boldsymbol{C}$ with an appropriate scaling factor $\alpha \in \mathbb{R}^+$ so that $\rho(\alpha \cdot \boldsymbol{C}) < 1$ is valid [36].

[7] The spectral radius $\rho(\boldsymbol{C})$ of an arbitrary matrix $\boldsymbol{C}$ is defined as the maximum of its greatest eigenvalue: $\rho(\boldsymbol{C}) = \max\{|\Lambda_i| \mid \Lambda_i \text{ is eigenvalue of } \boldsymbol{C}\}$. For symmetrical matrices the spectral radius is equal to the greatest singular value and thus to the matrix 2-norm.

Jaeger's arguments seem intuitive since it is known from linear systems

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} \tag{5.26}$$

that the zero state $\boldsymbol{x} = \boldsymbol{0}$ is a stable equilibrium for all initial conditions if the spectral radius of $\boldsymbol{A}$ is smaller one,

$$\rho(\boldsymbol{A}) < 1\,. \tag{5.27}$$

Such a passive system would always produce reliable response signals when driven by an external signal. However, the problem is that this property is partially destroyed for ESNs by the nonlinear transformation of the tanh-function. The condition in Eq. (5.25) also includes *unreliable* networks.

Consider as an example a network with the input coefficients $\boldsymbol{b} = (1, 1)^T$ and the following connection matrix

$$\boldsymbol{C} = \begin{pmatrix} 0.34 & 2.31 \\ -0.56 & -1.93 \end{pmatrix}\,. \tag{5.28}$$

The norm and the spectral radius of this matrix are

$$\sigma(\boldsymbol{C}) \approx 3.07 \;\text{ and }\; \rho(\boldsymbol{C}) \approx 0.79\,. \tag{5.29}$$

Following Jaeger's concept a network with such a connection matrix should have the "echo state property", i.e. it should be reliable for all input signals $\{u_t\}_{t\in\mathbb{I}}$. However, this is not the case. For the zero input signal with $u_t = 0, \forall t \in \mathbb{I}$, the network in Eq. (5.22) follows different trajectories depending on the initial state (see Fig. 5.4).

In the numerical experiment the recurrent network with the connection matrix $\boldsymbol{C}$ defined in Eq. (5.28) was iterated 2000 time steps according to Eq. (5.22) for different initial states $\boldsymbol{x}_0 \in \mathbb{R}^2$. Additionally to the stable equilibrium point $\boldsymbol{z}_1 = (0, 0)^T$ the system has also a stable 2-cycle, alternating between the points $\boldsymbol{z}_{2a} \approx (0.864, -0.694)^T$ and $\boldsymbol{z}_{2b} \approx (-0.864, 0.694)^T$, which are both stable fixed points of the 2-iterated system equations in Eq. (5.22). The basins of these attractors and the stationary behavior of the first component of the network are depicted in Fig. 5.4. The basin of the 2-cycle is split in region 2a and region 2b to mark the regular cycle and the anti-cycle, which result effectively in two different outputs from the network.

Since the network behaves in different ways, depending on the initial states, it does not fulfill the requirement for a reliable model, which is forgetting its history and approaching a specific trajectory that corresponds to the input signal. One may argue that the network could always be started from the same initial state, thus leading to the same behavior. However, this does not provide the desired reliability condition, as is shown next.

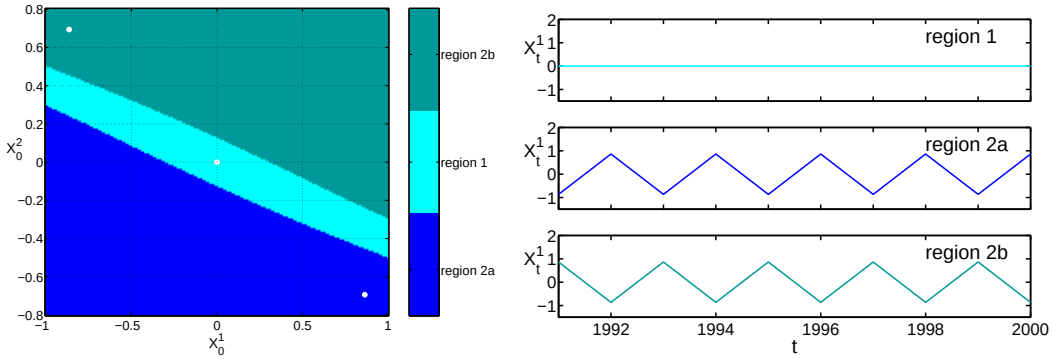

Figure 5.4: Left: Depending on the initial state $\boldsymbol{x}_0 = (x_0^1, x_0^2)^T$, the network dynamics in Eq. (5.22) settles either on the fixed point $\boldsymbol{z}_1 = (0,0)^T$ (region 1) or on the 2-cycle $\boldsymbol{z}_{2a/b} \approx (\pm 0.864, \mp 0.694)^T$ (region 2a/b). The three orbit points $\boldsymbol{z}_1, \boldsymbol{z}_{2a/b}$ are marked as white circles. **Right:** The final behavior of the first component of the state vector for time steps $t = 1991, \ldots, 2000$. Either constantly at zero (region 1) or alternation between -0.864 and 0.864 (region 2a/b). The behavior of the second component of the state is similar (not shown here).

In [36] Jaeger admits that "Recurrent networks [...] may exhibit disjoint regions $\mathcal{A}$, $\mathcal{B}$ of their state space in which network states stay contained". Additionally, he requires these regions to be independent of the input, and his discussion of the network behavior is based on the existence of one such region, in which the network dynamics takes place and which he calls the set of *admissible states* [36]. This requirement is very problematic. If multiple attractors coexist in a nonlinear system, their basins of attraction generally do depend on the input signal. This is demonstrated with the previous example in Fig. 5.5 and Fig. 5.6. Instead of driving the network with a pure zero signal, a simple sine signal with the amplitude $a = 0.15$ is prepended (see Fig. 5.5). The driving is started at three different points in time $t_0 = 1$, $t_0 = 101$, and $t_0 = 301$, which results in starting with a zero amplitude, the maximum amplitude and the minimum amplitude, respectively. For each of these runs the signal amplitude is set to zero at time step $t_1 = 1200$ and the network is given 800 time steps to settle into a stationary behavior. In analogy to Fig. 5.4 the initial states that result in different network dynamics at time step $t = 2000$ are marked as belonging to region 1, region 2a, or region 2b (see Fig. 5.6). As can be clearly seen, the basins of attraction are not fixed and depend strongly on the value of the input signal at the time step at which the driving was started.

This means that starting for example from the zero initial state $\boldsymbol{x}_0 = (0,0)^T$ can result in three different response signals. If the driving is started at time step $t_0 = 1$, the network settles in the fixed point $\boldsymbol{z}_0 = (0,0)^T$. If it is started at $t_0 = 101$, the network's stationary behavior is jumping between the points $\boldsymbol{z}_{2a} \approx (0.864, -0.694)^T$ and $\boldsymbol{z}_{2b} \approx (-0.864, 0.694)^T$. And for $t_0 = 301$ this

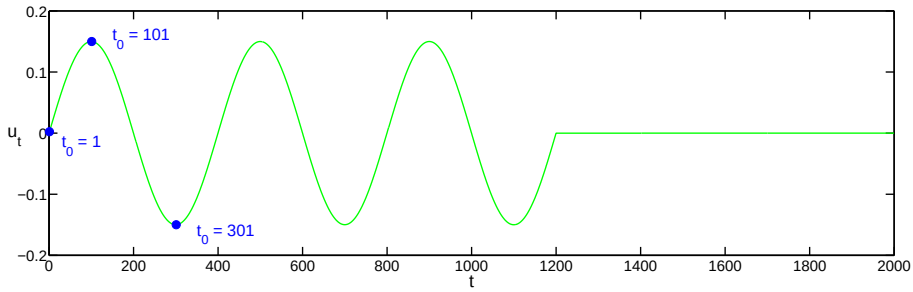

Figure 5.5: The input signal that was used for driving network in Eq. (5.22), beginning at three different points in time: $t_0 = 1$, $t_0 = 101$, and $t_0 = 301$.

jumping occurs in an anti-cycle.

The problem of different basins is not limited to the initial conditions. In many real world applications the input signal generally contains a certain amount of noise. Even if the network is started at an initial state which would normally produce a reliable response, a noise peak in the input signal may kick the internal state of the network from one basin into another and thus provoke an unreliable response.

It cannot be assumed that in high dimensional state spaces of networks with many elements the structure of the basins is simpler than for this 2-dimensional example. On the contrary, it can be expected that many different forms of stationary behavior can coexist, which are much more complicated and whose basins depend also strongly on the input signal. In practice this means that if there are multiple equilibrium states for a network it is impossible to find something like "admissible states" that can guarantee a reliable response signal to an input. Such networks are only reliable if driven by the same signal or at least one that deviates as less as possible from the original signal. However, models are seldom developed to be applied only on one specific input signal. On the contrary, typical applications involve the network being driven by different input signals. If the network behaves unreliably, it is certainly not suited as a model for tasks like control or prediction.

The conclusion from this section is that networks that comply to the condition in Eq. (5.25) are not automatically reliable and that they should be used with caution. While the strong condition in Eq. (5.21) ensures a reliable network, the weaker condition in Eq. (5.25) may produce unreliable networks. Therefore, the latter should be avoided as a guideline in the developing phase of recurrent networks if reliability is crucial for the application at hand. If this is not possible, a thorough testing phase with different input signals is necessary in order to minimize the risk of producing an unreliable model.

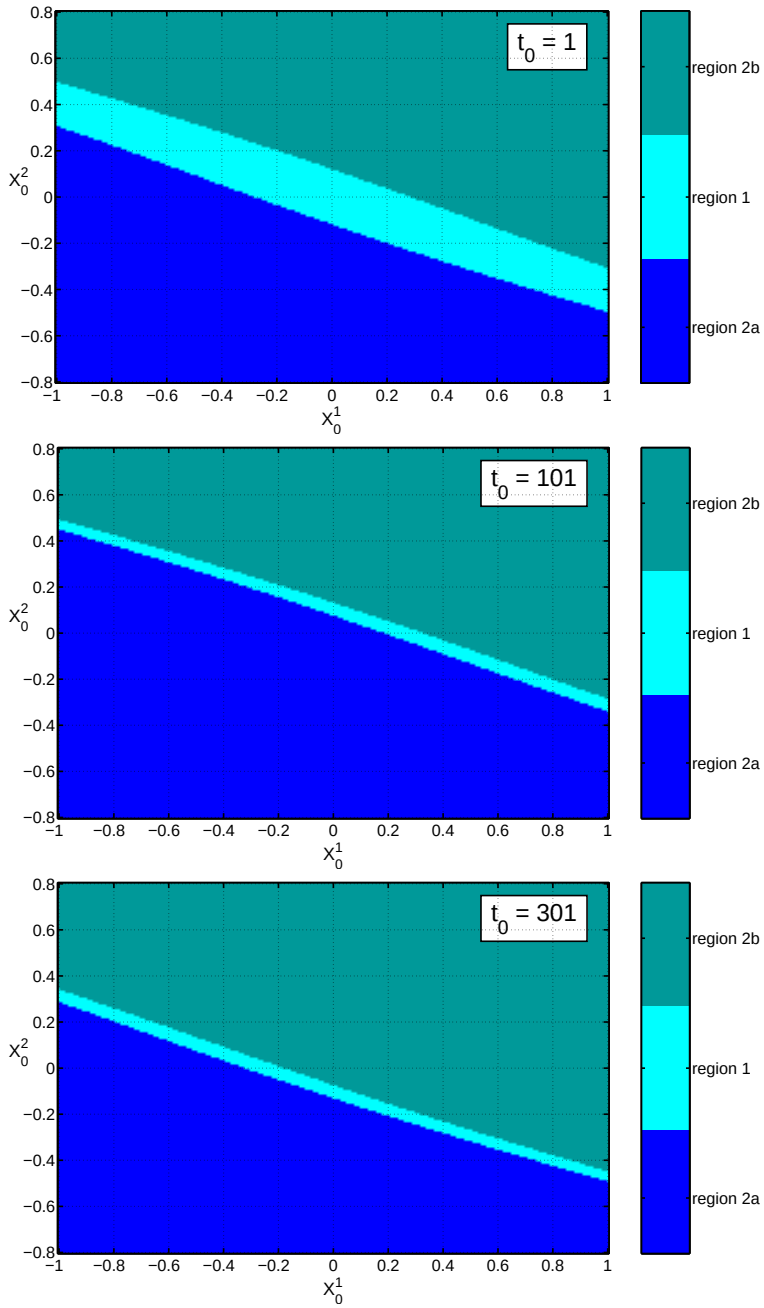

Figure 5.6: The basins of attraction for the network in Eq. (5.22) that was driven with the signal in Fig. 5.5, starting at three different times: $t_0 = 1$ (top), $t_0 = 101$ (middle), $t_0 = 301$ (bottom). The initial states are color coded depending on the stationary behavior at time step $t = 2000$ (as defined in Fig. 5.4).

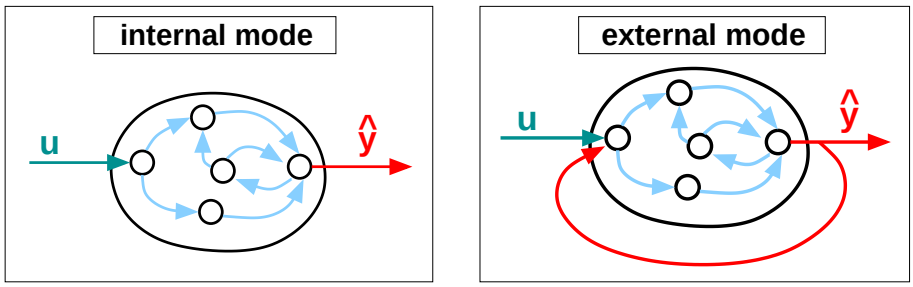

Figure 5.7: Left: In the internal mode the network is driven by an input signal $\{u_t\}_{t\in\mathbb{I}}$ and responds with an output signal $\{\hat{y}_t\}_{t\in\mathbb{I}}$. The elements are mutually coupled by the internal connection matrix. **Right:** In the external mode an additional external feedback loop is established by using the network output as an extra input signal. For some tasks, as for example free-running predictions, the network is decoupled from the external input, and the driving is taken over by the external feedback loop.

5.3.2 Internal and external mode

Recurrent networks can be used in two different ways, which are in this thesis referred to as *internal mode* and *external mode* (see Fig. 5.7). In the *internal mode* the network is driven by a signal $\{u_t\}_{t\in\mathbb{I}}$ from the outside and produces an output signal $\{\hat{y}_t\}_{t\in\mathbb{I}}$ (see lhs of Fig. 5.7). In this mode the elements of the network are coupled only by the internal connection matrix $\boldsymbol{C} \in \mathbb{R}^{M\times M}$, and the system equations in matrix notations are

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t\right) , \tag{5.30}$$

with $\boldsymbol{x}_t$ the state vector at time step $t \in \mathbb{I}$, $\boldsymbol{c}_0 \in \mathbb{R}^M$ the bias values of the elements, and $\boldsymbol{b} \in \mathbb{R}^M$ the connection coefficients to the external input signal. As discussed in the previous section (Section 5.3.1) connection matrix $\boldsymbol{C}$ is required to have a norm smaller unity, $\sigma(\boldsymbol{C}) < 1$, in which case the output of the network is reliable.

In the *external mode* the mutual coupling between the elements is extended by an external loop, which feeds the output of the network back as an additional input to the system (see rhs of Fig. 5.7). In the general notation the external mode can be expressed as an extension to the state equations in Eq. (5.30), namely as

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t + \boldsymbol{c}^{\text{back}}\hat{y}_{t-1}\right) , \tag{5.31}$$

with the additional feedback coefficients $\boldsymbol{c}^{\text{back}} \in \mathbb{R}^M$.

For both modes the output of the network is a function of its state variables as defined in Eq. (5.6). Typically this function is represented by the linear weighted sum

$$\hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 , \tag{5.32}$$

with $\boldsymbol{w} \in \mathbb{R}^M$ the output coefficients and $w_0 \in \mathbb{R}$ the bias value or the constant offset of the network. For this special choice of output function the notation of the external mode can be reduced to the one of the internal mode. The reduced form can be written as

$$\boldsymbol{x}_t = \tanh\left(\tilde{\boldsymbol{C}}\boldsymbol{x}_{t-1} + \tilde{\boldsymbol{c}}_0 + \boldsymbol{b}u_t\right) , \tag{5.33}$$

with the new connection matrix

$$\tilde{\boldsymbol{C}} \equiv \boldsymbol{C} + \boldsymbol{c}^{\text{back}} \otimes \boldsymbol{w}^T , \tag{5.34}$$

with $\otimes$ denoting the tensorial product, and the new bias values

$$\tilde{\boldsymbol{c}}_0 \equiv \boldsymbol{c}_0 + w_0\boldsymbol{c}^{\text{back}} . \tag{5.35}$$

Although this reformulation is formally possible, it is often unpractical. In most cases the integration of the feedback loop into the new connection matrix $\tilde{\boldsymbol{C}}$ leads to the property $\tilde{\boldsymbol{C}} > 1$. From Section 5.3.1 it is known that networks with such a connection matrix are unreliable, and this implies that networks in the external mode have to be treated differently than the ones in the internal mode[8]. It is important to differentiate between the two modes in applications and to know about their strengths and weaknesses. For this reason this topic is discussed in more detail for the modeling tasks of prediction and cross-prediction.

Prediction

The internal mode is typically applied for tasks like one-step predictions, where a time series $\{y_t\}_{t\in\mathbb{I}}$ has to be forecast a constant time step $T \in \mathbb{N}$ into the future (see Section 2.2.1 and Fig. 2.2). For example for $T = 1$ the network output $\hat{y}_t$ is an estimation of the true value y_t and the input signal is a delayed version of the original time series: $u_t = y_{t-1}$, $\forall t \in \mathbb{I}$. In this case Eq. (5.30) and Eq. (5.32) can be written as

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}y_{t-1}\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 . \tag{5.36}$$

If the norm of the connection matrix is smaller one, $\sigma(\boldsymbol{C}) < 1$ (see Section 5.3.1), the network in Eq. (5.36) follows for every input signal a specific trajectory, which is independent of the initial state $\boldsymbol{x}_0$ after a certain time span of transient behavior. Then the output weights $\boldsymbol{w}$ and w_0 can be adapted to the data in such a way that the network output approximates any desired signal that is deterministically related to the input signal.

[8] For the the case $\tilde{\boldsymbol{C}} < 1$ the external mode is indeed equivalent to the internal mode and does not need to be treated in an extra scope.

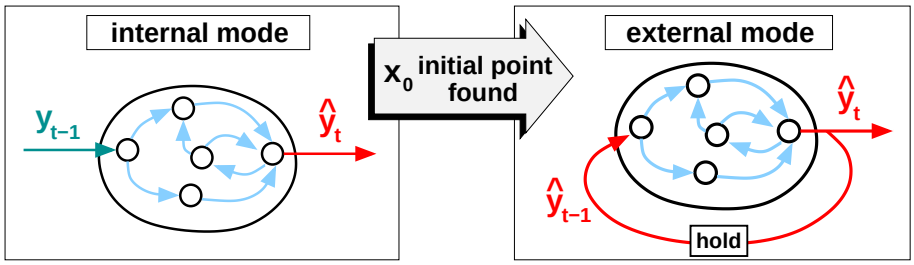

Figure 5.8: A recurrent network for free-running predictions has to be started in the internal mode. After the tuning-in phase, in which the transient behavior has receded and the right initial state has been found, it can be switched into the external mode.

When the network in Eq. (5.36) has been adapted to the data and produces one-step predictions of the time series $\{y_t\}_{t\in\mathbb{I}}$, it can also be used for free-running predictions. For this purpose the delayed version of the time series is replaced by the network's own output one time step from the past, so that we have

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}\hat{y}_{t-1}\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 \, . \tag{5.37}$$

Technically the step from one-step to free-running predictions is done by decoupling the network from the external input signal and by establishing an additional feedback loop from the output to the input. Therefore, it can be seen that Eq. (5.37) is a special case of the external mode where the external connections were set to zero $\boldsymbol{b} = \boldsymbol{0}$ and their former values were taken over by the feedback connections $\boldsymbol{c}^{\text{back}} = \boldsymbol{b}$.

As already mentioned, the external form can be formally reduced to the internal form. For Eq. (5.37) the result of such a reduction is

$$\boldsymbol{x}_t = \tanh\left(\tilde{\boldsymbol{C}}\boldsymbol{x}_{t-1} + \tilde{\boldsymbol{c}}_0\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 \, . \tag{5.38}$$

with

$$\tilde{\boldsymbol{C}} \equiv \boldsymbol{C} + \boldsymbol{b} \otimes \boldsymbol{w}^T \, , \tag{5.39}$$

and

$$\tilde{\boldsymbol{c}}_0 \equiv \boldsymbol{c}_0 + w_0 \boldsymbol{b} \, . \tag{5.40}$$

These are state equations of an autonomous network, which is not driven by an external signal. To be used for free-running predictions of a specific time series $\{y_t\}_{t\in\mathbb{I}}$, it has to be started at the correct initial state $\boldsymbol{x}_0$. The difficulty is to find the right one which corresponds to the time series. Another point is that stability for the dynamical system in Eq. (5.38) is no longer guaranteed but has to be numerically verified for every case.

While the former connection matrix $\boldsymbol{C}$ has to comply to the reliability criterion $\sigma(\boldsymbol{C}) < 1$, the new connection matrix $\tilde{\boldsymbol{C}}$ must not. Otherwise the state of the network in Eq. (5.38) would approach a fixed point and the output

of the network would become constant, which is hardly a desired behavior in free-running predictions. The implication is that for free-running predictions we are dealing with reliable networks that are deliberately made unreliable by an external feedback loop.

The unreliability of the network does not imply that it is unstable but only that it can produce unexpected responses if it is not started in the right basin (see Section 5.3.1). This is equivalent with the former notion that every time series $\{y_t\}_{t\in\mathbb{I}}$ that we want to predict has a corresponding initial state $\boldsymbol{x}_0$. A practical way to find such an initial state is to run the network in the internal mode as in Eq. (5.36), producing one-step predictions, until the transient behavior recedes. This is also called the *tuning-in phase* of the network where it atones to the time series $\{y_t\}_{t\in\mathbb{I}}$ that is to be predicted (see Fig. 5.8). The reliability criterion of connection matrix $\boldsymbol{C}$ guarantees that the transient behavior does eventually recede and the network starts to follow a specific trajectory that corresponds to the input signal. When this happens, the network can be switched into the external mode from Eq. (5.31) by replacing the external signal with its own output. If the right initial state $\boldsymbol{x}_0$ is known beforehand, the detour with the tuning-in phase can be avoided and the network can be directly started in the external mode. However, this is almost never the case since the basin structure of the network dynamics depends strongly on the input signal, as discussed in Section 5.3.1.

Cross-prediction

In the case of free-running prediction it was shown that the network has to run in a combination of the internal and the external mode. This is what we refer to as the *external approach*. The case of cross-prediction is more interesting because here the application of the external mode can be avoided and the network can be run in a pure internal mode. This is what we call *internal approach*. In this section the two approaches are compared to each other.

In a cross-prediction task we have a signal $\{u_t\}_{t\in\mathbb{I}}$ that we use as an input to the model and we want to predict another signal $\{y_t\}_{t\in\mathbb{I}}$ that is deterministically related to the former signal (see Section 2.2.2 and Fig. 2.4). The straight-forward way is the *internal approach* (see lhs of Fig. 5.9). Thereby the network is used in the internal mode, driving it with the input signal $\{u_t\}_{t\in\mathbb{I}}$ and reading out its output $\{\hat{y}_t\}_{t\in\mathbb{I}}$. The state equations are

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}u_t\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 , \tag{5.41}$$

with $\hat{y}_t$ the estimated value of the original one y_t. Since in this mode the network does not depend on its previous predictions, there is no difference between one-step and free-running cross-predictions for the internal approach.

In the *external approach* a delayed version of the time series $\{y_t\}_{t\in\mathbb{I}}$ is used as an additional input signal $\{u'_t\}_{t\in\mathbb{I}}$ (see rhs of Fig. 5.9). This is equivalent to

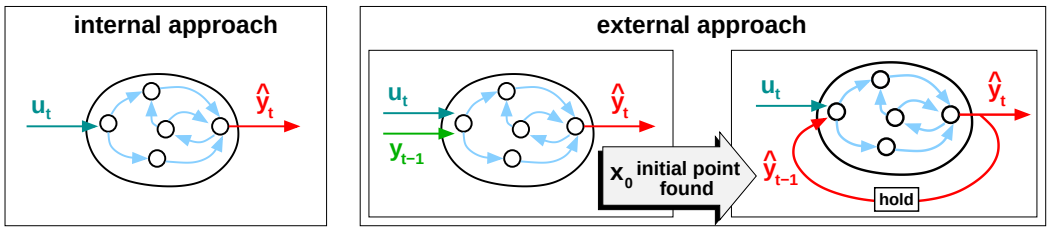

Figure 5.9: For cross-prediction task there are two possible approaches. **Left:** In the internal approach only the internal mode is applied. **Right:** The external approach is similar to the approach for free-running predictions in Fig. 5.8. The network has to be started in the internal mode. After a transient phase it can be switched into the external mode.

one-step predictions and can be written as

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}_1 u_t + \boldsymbol{b}_2 y_{t-1}\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 . \tag{5.42}$$

with $\boldsymbol{b}_1 \in \mathbb{R}^M$ the connection coefficients to the first input signal $\{u_t\}_{t\in\mathbb{I}}$ and $\boldsymbol{b}_1 \in \mathbb{R}^M$ the connection coefficients to the second input signal $\{u'_t\}_{t\in\mathbb{I}}$, with $u'_t = y_{t-1}\,, \forall t \in \mathbb{I}$. For free-running predictions the additional input signal is replaced by the network's own output, so that we have

$$\boldsymbol{x}_t = \tanh\left(\boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{c}_0 + \boldsymbol{b}_1 u_t + \boldsymbol{b}_2 \hat{y}_{t-1}\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 . \tag{5.43}$$

Again this can be reformulated in the reduced version as

$$\boldsymbol{x}_t = \tanh\left(\tilde{\boldsymbol{C}}\boldsymbol{x}_{t-1} + \tilde{\boldsymbol{c}}_0 + \boldsymbol{b}_1 u_t\right) , \quad \hat{y}_t = \boldsymbol{w}^T \cdot \boldsymbol{x}_t + w_0 . \tag{5.44}$$

with

$$\tilde{\boldsymbol{C}} \equiv \boldsymbol{C} + \boldsymbol{b}_2 \otimes \boldsymbol{w}^T , \tag{5.45}$$

and

$$\tilde{\boldsymbol{c}}_0 \equiv \boldsymbol{c}_0 + w_0 \boldsymbol{b}_2 . \tag{5.46}$$

Similar to the free-running predictions the external mode cannot be used directly from the start for free-running cross-predictions because in most cases $\sigma(\tilde{\boldsymbol{C}}) > 1$ and the network is thus unreliable[9]. Instead, the network is started in the internal mode and is only switched into the external mode after the transient behavior abated.

Compared to the internal approach the external approach seems to be unnecessarily complicated. The external mode cannot be applied directly but

[9] In contrast to free-running predictions there is no argument that the condition $\sigma(\tilde{\boldsymbol{C}}) > 1$ is really necessary for free-running cross-predictions. The network can produce variational responses even if $\sigma(\tilde{\boldsymbol{C}}) < 1$ because it is driven by an external input signal $\{u_t\}_{t\in\mathbb{I}}$. That means the reduced form in Eq. (5.44) can indeed yield connection matrices with $\sigma(\tilde{\boldsymbol{C}}) < 1$. In such a case the network can be treated in the scope of the internal approach.

has to be preceded by the internal mode to find the right initial state. This means that for every application of the network the user has to provide not only the input signal $\{u_t\}_{t\in\mathbb{I}}$ but also a short starting sequence of the time series $\{y_t\}_{t\in\mathbb{I}}$ for the tuning-in phase of the network.

Another disadvantage is that the guarantee for a stable system as in the internal approach is lost, because the feedback loop destabilizes the network dynamics. The network response can become unreliable for networks with elements that have a bounded activation function, like $\tanh(\cdot)$ or even grow unlimited for unbounded activation functions.

Nevertheless, the external approach has also advantages which can outweigh its negative sides. Since it is not limited by the requirement $\sigma(\boldsymbol{C}) < 1$, it can represent a broader class of input-output relationships. In a numerical experiment we have used the modeling task from Section 3.2.4 to compare the internal and the external approach to each other. The task consisted of reproducing the input-output behavior the system

$$y_t = a_0 y_{t-1} + \frac{b_0}{1+|y_{t-2}|} + c_0 u_t \,, \tag{5.47}$$

with the parameters set to $a_0 = 0.8$, $b_0 = -0.2$, and $c_0 = 1.0$.

As models we used recurrent networks with $M = 200$ elements, which were randomly connected with a connection probability[10] of $p = 0.01$, meaning that every element in the network was connected to two other elements on average. The probability for external connection was set to $p = 0.1$ and for elemental bias values to $p = 0.8$. The values of all nonzero random parameters were normally distributed, $c_{ij}, b^{(i)}, c_0^{(i)} \sim \mathrm{N}(0,\,1)$, $i = 1,\ldots,200$, and connection matrix $\boldsymbol{C}$ was scaled to accord to $\sigma(\boldsymbol{C}) = 0.95$. The output weights w_i, $i = 0, 1, \ldots, 200$, were trained on a training data set consisting of the input time series $\{u_t\}_{t\in\mathbb{I}}$ and the output time series $\{y_t\}_{t\in\mathbb{I}}$ each with $N = 10000$ data points. During training the first 1000 data points were left out to discard transients. For the internal approach as well as for the external approach the numerical experiment was repeated on 100 different networks, which were randomly created in the above described way.

The performance was measured on a distinct test set with $N = 10000$ data points. The models were to predict the original values in a free-running mode. The first 1000 data points were discarded as transients, while the rest was used for the NMSE quality measure

$$\mathrm{NMSE}_\infty = \frac{100\,\%}{\sigma_y^2} \sum_{t=1001}^{10000} (y_t - \hat{y}_t)^2 \,, \tag{5.48}$$

with σ_y^2 the variance of time series $\{y_t\}_{t\in\mathbb{I}}$, which was to be predicted. The results can be seen in a histogram in Fig. 5.10. While the performance of the

[10] Connection probability denotes the probability that an entry in the connection matrix $\boldsymbol{C}$ is non-zero.

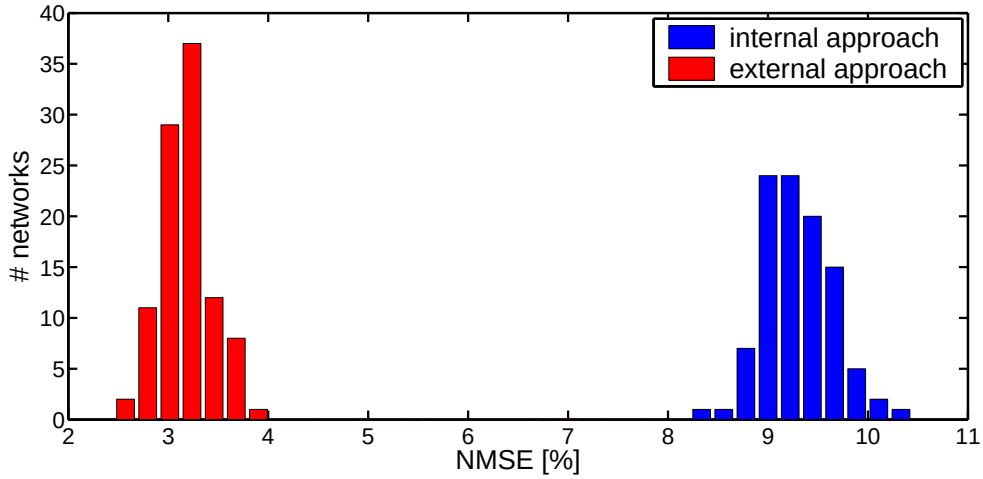

Figure 5.10: Performance of internal and external approach for a cross-prediction task of simulating the system in Eq. (5.47), measured with the NMSE_∞ value in Eq. (5.48). The external approach produces notworks that are much more precise for free-running cross-predictions than the ones produced by the internal approach.

internal approach is grouped around the mean value $\langle \mathrm{NMSE}_\infty \rangle \approx 9.4\,\%$, the performance of the external approach is much better with a mean value of $\langle \mathrm{NMSE}_\infty \rangle \approx 3.3\,\%$.

The superiority of the external approach concerning the prediction performances could be empirically confirmed on further numerical experiments, which cannot all be exemplified here. One possible reason for these results was already mentioned. By loosening the tight stability bound $\sigma(\boldsymbol{C}) < 1$ the networks can model a broader range of systems. Another reason is that the external approach effectively uses one-step predictors in the training phase. Free-running applications with these networks are only possible because of the external feedback loop. Since one-step predictions involve much simpler transformation functions from the input values to the output values, the probability that a randomly created network can successfully model such a relationship is much higher than for free-running predictions. Therefore, the internal approach should always be combined with an optimization procedure, as described in Section 5.3.4.

Aside from potentially being unstable, the external approach has another weakness. For noisy data it is susceptible to biased predictions. It was already described in detail that models based on one-step predictions produce outputs that systematically deviate from the true values, a phenomenon referred to as bias (see Section 3.2). The same happens for networks that are based on the external approach. For demonstration the above cross-prediction task concerning the simulation of the system in Eq. (5.47) was repeated. However, instead of using the original output time series $\{y_t\}_{t\in\mathbb{I}}$ in the training set it

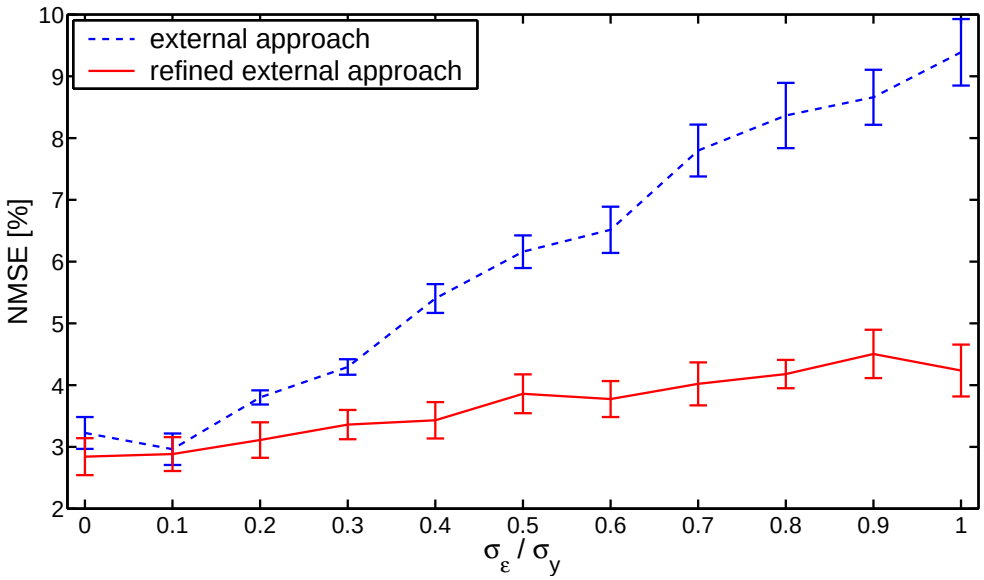

Figure 5.11: The NMSE_∞ values of free-running predictions on a noise-free test set, gained by networks that were created in accordance to the external approach on noisy training data. The noise level of the training set was stepwise increased as $\sigma_\varepsilon = k \cdot 0.1 \cdot \sigma_y$ with $k = 0, 1, \ldots, 10$. For each noise level 100 networks were tested. The lines connect the averages of the NMSE_∞ values while the errorbars denote the standard deviations of each performance distribution. The dashed blue line marks the results of the usual external approach while the red line represents the results of the refinement procedure.

was modified to

$$\tilde{y}_t = y_t + \varepsilon_t \,, \tag{5.49}$$

with normally distributed values $\varepsilon_t \sim \text{N}(0,\, \sigma_\varepsilon^2)$, representing additive noise as common for measurement errors. To see the effects of a noisy training set more clearly, the noise variance σ_ε^2 was increased stepwise according to $\sigma_\varepsilon = k \cdot 0.1 \cdot \sigma_y$, $k = 0, 1, \ldots, 10$. For every such noise level 100 randomly created networks were trained with the same specifications as mentioned above. Also the performance was measured in the same way, following the NMSE_∞ criterion in Eq. (5.48) on a noise-free test set. The results are displayed in Fig. 5.11. Clearly, networks based on the external approach are strongly influenced by noise in the training data. Their performance deteriorates more and more with increasing level of noise.

The problem that arises with noisy training data is based on limiting the training to one-step predictions (see Section 3.2). The output weights w_i of the network are adapted to one-step predictions instead of optimizing them for free-running predictions. However, the latter would be necessary, because in free-running application the values for the output weights are also used for the feedback loop $c^{\text{back}_i} = w_i$. Thus, optimal weights for free-running applications can be obtained by optimizing them on cost functions based on free-running

predictions like MSE_∞, which is in most cases a very time intensive nonlinear optimization procedure.

A linearized procedure, which approximately obtains the optimal solution is the following *refinement algorithm* for recurrent networks that are based on the external approach. The algorithm follows a predictor-corrector scheme, where the feedback loop is used to compute new output weights and then the output weights are used to update the feedback loop.

step 0: Set $n = 0$. Obtain output weights $\boldsymbol{w}^{(0)}$ by training network on one-step predictions (MSE_1). Set $\boldsymbol{c}^{\text{back}} = \boldsymbol{w}^{(0)}$. Compute free-running performance with $\text{NMSE}_\infty^{(0)}$ on validation set.

step 1: Set $n = n + 1$. Use the external loop $\boldsymbol{c}^{\text{back}}$ in the training as if part of internal connection matrix. Obtain new output weights $\tilde{\boldsymbol{w}}^{(n)}$, which are now based on MSE_∞.

step 2: Update real output weights according to $\boldsymbol{w}^{(n)} = (1 - \alpha) \cdot \boldsymbol{w}^{(n-1)} + \alpha \cdot \tilde{\boldsymbol{w}}^{(n)}$. Set $\boldsymbol{c}^{\text{back}} = \boldsymbol{w}^{(n)}$.

step 3: Compute free-running performance with $\text{NMSE}_\infty^{(n)}$ on validation set. If $\text{NMSE}_\infty^{(n)} < \text{NMSE}_\infty^{(n-1)}$ go to **step 4** else go to **step 5**.

step 4: Accept $\boldsymbol{w}^{(n)}$ for network and go to **step 1**.

step 5: Reject $\boldsymbol{w}^{(n)}$ for network and **stop**.

The update parameter $\alpha \in [0, 1] \subset \mathbb{R}$ should have a small value to enable smooth transitions of the values in the feedback loop. Another point is that a validation set has to be reserved in the training set. The validation set is used only to measure the performance of the network and not to determine the output weights. In this way overfitting is avoided (see Section A.3).

A demonstration of the refinement algorithm is seen in Fig. 5.11. We have used the update parameter $\alpha = 0.1$ and reserved the last 2000 data points in the noisy training set for validation. As can be seen, the performance of the networks can be considerably improved by the refinement algorithm. This is due to the optimization on free-running predictions, which tends to reduce the influence of noise on the modeling procedure (see Section 3.2). Note also that the performance can be slightly improved by the refinement algorithm even for noise-free training data. Although the refinement algorithm does not yield the optimal output weights it allows the development of recurrent networks which are far less susceptible to bias and which are better adapted to free-running predictions.

5.3.3 Using selection methods

To ensure a rich internal dynamics ESNs must have an appropriate number of elements. Depending on the application, typical network sizes are 100, 200, or 400 elements. On the one hand the random construction of the internal connections creates a broad range of response signals from the elements. On the other hand it cannot be ensured that each one of these signals differs from the others. On the contrary, in most cases some of the elements in the network will behave similarly, leading to redundant internal responses.

The redundancy of the internal dynamics can cause problems for the computation of the weights that are employed for the network output. They are computed by minimizing the cost function

$$\mathcal{L}(\boldsymbol{w}) = (\boldsymbol{y} - \hat{\boldsymbol{y}})^T \cdot (\boldsymbol{y} - \hat{\boldsymbol{y}}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T \cdot (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})\,, \tag{5.50}$$

with

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^{(1)} & \cdots & \boldsymbol{x}_1^{(M)} \\ \vdots & \cdots & \vdots \\ \boldsymbol{x}_N^{(1)} & \cdots & \boldsymbol{x}_N^{(M)} \end{pmatrix}\,, \tag{5.51}$$

the matrix of the internal states, representing the output of M elements in N time steps. The optimal weights can be expressed as

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \equiv \boldsymbol{X}^\dagger\boldsymbol{y}\,. \tag{5.52}$$

with the so called *pseudo inverse* $\boldsymbol{X}^\dagger$ (see Section A.2.1). If some of the columns of $\boldsymbol{X}$ are redundant, as in the case of elements with similar behavior, great care has to be taken concerning the computation of the pseudo inverse to avoid results corrupted by numerical instabilities (see Section A.2.1).

An even greater risk is that overfitting occurs because the number of adjustable parameters grows with the number of elements (see Section A.3). If the size of the training data set is comparably small or noise is present in the data, the generalization ability of the model cannot be ensured. This situation is similar to the static modeling approach with NARX models (see Section 3.1.2), in case the model consists of a great number of basis function. In the static approach this situation is remedied by using a forward selection technique called Fast Orthogonal Search (FOS, see Section A.2.2), where only a non-redundant subset from a pool of possible basis functions is selected. The advantage of the FOS procedure, aside from the anticipating redundancy, is that also chances of overfitting are considerably reduced. By choosing one basis function at a time the complexity of the model can always be adapted to the data. If only the relevant basis functions are selected, an adaptation of the model to artifacts, produced by noise or finiteness of data, can be averted or at least diminished.

Recurrent networks can also benefit from the FOS procedure. Instead of using all internal response signals of the network only the few best ones can be selected for the network output. In this way redundancy and overfitting is avoided. This is demonstrated in a numerical experiment. In this experiment the relationship of a Rössler system that drives a Lorenz system has to be reproduced by the model. The Rössler system is given by the following ODE

$$\begin{aligned} \dot{x}_1 &= 2 + x_1(x_2 - 4) \\ \dot{x}_2 &= -x_1 - x_3 \\ \dot{x}_3 &= x_2 + 0.45x_3 \end{aligned} \tag{5.53}$$

and the driven Lorenz system

$$\begin{aligned} \dot{z}_1 &= -10(z_1 - z_2) \\ \dot{z}_2 &= 28u(t) - z_2 - u(t)z_3 \\ \dot{z}_3 &= u(t)z_2 - 2.666z_3 \end{aligned} \tag{5.54}$$

with the driving signal $u(t) = x_1(t) + x_2(t) + x_3(t)$. The systems in Eq. (5.53) and Eq. (5.54) were numerically integrated by the Runge-Kutta algorithm with an integration time step $t_i = 0.025$. For modeling purposes the driving signal $u(t)$ was given as the input signal and the first component of the Lorenz system was given as the output signal $y(t) = z_1(t)$, where the latter had to be predicted by the model from the former. The training and test set contained both signals as sampled time series $\{u_t\}_{t\in\mathbb{I}}$ and $\{y_t\}_{t\in\mathbb{I}}$ with the sampling time $t_s = 0.1$.

A RNN with 200 elements was randomly created and was trained to reproduce the functional relationship between the driving and the driven oscillator. All networks were based on the internal approach (see previous section, Section 5.3.2). An artificial noise source was added to the training data

$$\tilde{y}_t = y_t + \varepsilon_t \,, \tag{5.55}$$

with the noise terms $\varepsilon_t \sim \mathrm{N}(0,\, \sigma_\varepsilon)$. Three different noise levels were applied: $\sigma_\varepsilon/\sigma_y = 0.0, 0.2, 0.4$, with σ_ε and σ_y the rooted variances of the noise and the output signal, respectively. The number of training points was varied from $N = 1000$ to $N = 10000$ data points. Training of the output weights of the network was done either with the **pinv** routine of Matlab or with the FOS algorithm. For both procedures the first 500 data points were discarded as transients. Following the training the performance of the models was measured on a noise-free test set by the NMSE criterion

$$\mathrm{NMSE} = \frac{100\,\%}{\sigma_y^2} \sum_{t=501}^{10000} (y_t - \hat{y}_t)^2 \,. \tag{5.56}$$

To account for the randomness in the creation of the RNNs every training procedure was repeated on 10 different RNNs. The results of the experiment

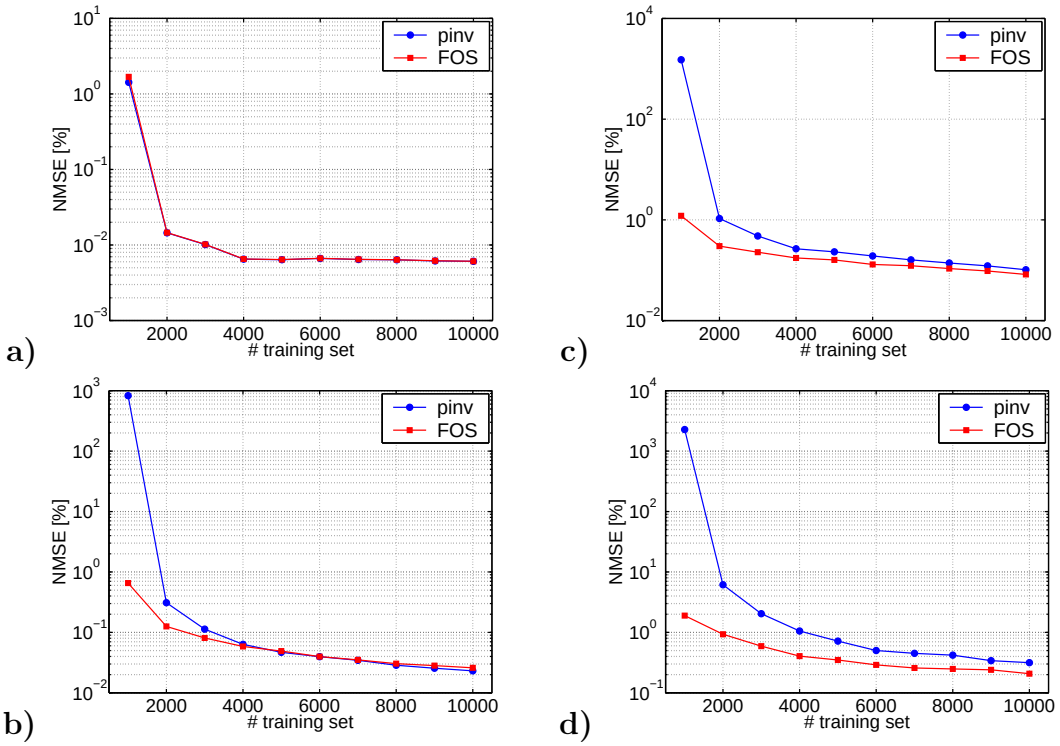

Figure 5.12: Comparison of the **pinv** and the FOS algorithm. Depicted are the prediction performances of RNNs with 200 elements against different sizes of the training set. While the performance values were measured on noise-free test data, the training data was artificially corrupted with different levels of noise. Every point in the figures represents the mean values of 10 different simulation runs. The relative noise levels of the training sets were: **a)** $\sigma_\varepsilon/\sigma_y = 0.0$, **b)** $\sigma_\varepsilon/\sigma_y = 0.1$, **c)** $\sigma_\varepsilon/\sigma_y = 0.2$, **d)** $\sigma_\varepsilon/\sigma_y = 0.4$ Note that the NMSE values are represented on a logarithmic scale.

are shown in Fig. 5.12. For noise-free training data both algorithms are effectively equivalent in producing good models. For both routines the quality of the models is improved by raising the number of data point in the training set. This improvement is drastic for the step from 1000 to 2000 data points and saturates around 4000 data points. With increasing noise level in the training data the differences of the two algorithms become more and more apparent. While the performance of the networks that were trained with the **pinv** algorithm deteriorates strongly, the networks trained with the FOS algorithm can keep up good results even for small training sets.

In the numerical example it was noticeable that only a fraction of the internal response signals was needed to create the output of the network. Many of the elements did not contribute to the output. This does not necessarily mean that these elements are useless. Some of them are needed to produce internal lags in the mutual information exchange, which also increases the diversity of the internal dynamics. However, many of the elements are indeed

redundant showing no or only small deviations in their response from other elements and having no positive effect on the internal dynamics. Herein lies the chance to reduce the network size. By adapting the model to the data many of the redundant elements can be put to use. This is primarily the reason why the size of optimized RNNs can be chosen smaller than that of ESNs without reducing the performance, as is shown in the next section (Section 5.3.4).

5.3.4 Optimization of internal connections

The strength of a network lies in the fact that a multitude of computing units process data sequences in a parallel manner. Even though the elements in the network may be simple functions, the network as a whole is able to perform very complex computations. To make this possible an exchange of information between the individual elements is needed. Through different connection paths signals are sent and received, creating a diversity in the responses of the otherwise equal elements. It comes at no surprise that the structure of these connections plays the most important role for the performance of recurrent networks.

The Echo State approach uses networks that are based on randomly connected elements [36]. These networks utilize the information in the training set only to adapt the output weights and not to adapt the internal connection weights. The argument behind this idea is that the amount of diversity in the internal dynamics of the network is rich enough to approximate any deterministic input-output relationship. Nevertheless, as can be seen in Fig. 5.10, the performance of randomly created networks can have a wide distribution. It seems that some of the random network architectures are better suited for the modeling task than others, and the difference between them cannot be totally compensated by adapting the output weights. It means that with luck we can have a very precise model, but also that in the other case a random network can perform very poorly.

In order to avoid picking the network with the worst performance the simplest strategy is to create many randomly connected networks and to choose the one that performs the best on a validation set. In this way we use the information in the training set also for choosing the best set of internal connections. Although this strategy is better than to use the next best network, it still uses the information of the data in an uncoordinated way. It does not use the fact that in general good networks can be made even better by slightly changing their parameters. Therefore, instead of jumping randomly in the parameter space, a better strategy is to make improvements by looking for better solutions in the vicinity of the current parameters.

Here we used an algorithm that is based on *Simulated Annealing* (SA). However, any algorithm that can deal with discrete and continuous variables, e.g. Genetic Algorithms, is possible. The SA procedure starts with a randomly

created network and the iteration parameter T, which is usually referred to as temperature, is initialized as $T = T_0 \in \mathbb{R}$. Step for step the temperature is decreased following an exponential cooling scheme, $T^{\text{new}} = \alpha \cdot T^{\text{new}}$ with $\alpha \in (0, 1) \subset \mathbb{R}$. And each step the network is modified by changing some of its parameters. In our case the changes could be

add connection: A zero entry in the connection matrix $\boldsymbol{C}$ is set to a random value taken from the normal distribution N(0, 1).

cut connection: A non-zero entry in the connection matrix is set to zero.

rewire connection: A zero and a non-zero entry in the connection matrix swap their locations.

change connection weight: A non-zero entry in the connection matrix is slightly changed by adding a random value from the normal distribution N(0, σ^2). The variance σ^2 may vary its value during a run of the SA procedure and is usually a function of the temperature. In our case: $\sigma^2 = (T/T_0)^2$.

change input weight: A nonzero entry in the input connection vector $\boldsymbol{b}$ is changed (see **change connection weight**).

change bias value: A nonzero entry in the bias vector $\boldsymbol{c}_0$ is changed (see **change connection weight**).

After having performed the modifications the stability criterion $\sigma(\boldsymbol{C}) < 1$ is checked. If the new connection matrix does not comply, it is rescaled appropriately. The modified network is evaluated against the former one on a validation data set, using the NMSE value as a measure of quality. If the new network performs better, i.e. $\text{NMSE}^{\text{new}} < \text{NMSE}^{\text{old}}$ it is automatically accepted. If its performance is worse, it is accepted with the probability

$$p = e^{\delta/T} \,, \quad , \delta = \frac{\text{NMSE}^{\text{old}} - \text{NMSE}^{\text{new}}}{\text{NMSE}^{\text{old}}} \,. \tag{5.57}$$

The acceptance of bad networks serves as a measure against getting stuck in a local minimum in the parameter space. It is mainly active in the beginning of an optimization run. With decreasing temperature the probability of accepting a bad network is effectively zero at the end.

For a demonstration of the algorithm we picked the same numerical experiment as in the subsection about cross-prediction in Section 5.3.2. The task consisted of simulating the input-output characteristics of the following system

$$y_t = a_0 y_{t-1} + \frac{b_0}{1 + |y_{t-2}|} + c_0 u_t \,, \tag{5.58}$$

with the parameters $a_0 = 0.8$, $b_0 = -0.2$, and $c_0 = 1.0$. The models were chosen to be RNNs with the dynamics described by Eq. (5.41). On the one side we used randomly created networks with different numbers of elements $M = 100$, $M = 200$, and $M = 400$. On the other side we used networks with $M = 60$ elements, that were optimized in the above described way. We could choose the network size drastically smaller because the optimization routine enabled us to reduce the number redundant elements, which are inevitably present in randomly created networks.

The random networks were created with a connection probability adjusted in such a way that every element had on the average 4 connection to other elements, i.e. $p = 0.04$, $p = 0.02$, and $p = 0.01$. These values were chosen according to the best results on the validation set. The optimized networks had a connection probability of $p = 0.06$, which translates almost to the same number of average connections. The probability for external connections was set to $p = 0.1$ for elemental bias values to $p = 0.8$. The values of all nonzero random parameters were normally distributed, $c_{ij}, b^{(i)}, c_0^{(i)} \sim \mathrm{N}(0,\, 1)$, $i = 1, \ldots, 200$, and connection matrix $\boldsymbol{C}$ was scaled to accord to $\sigma(\boldsymbol{C}) = 0.9$.

The output weights w_i of all networks were trained on a training data set consisting of the input time series $\{u_t\}_{t\in\mathbb{I}}$ and the output time series $\{y_t\}_{t\in\mathbb{I}}$ each with $N = 10000$ data points. For the optimized networks this data set was split in the first 8000 data points for training and the last 2000 data points for validating. The number of optimization steps in one SA run was set to 400. During training the first 1000 data points were left out to discard transients. All networks were trained in the sense of the external approach. Afterward the networks were evaluated with the NMSE_1 and the NMSE_∞ criterion on a distinct test set consisting of $N = 10000$ data points. The first 1000 data points were neglected as transients. For all randomly created networks we repeated the experiment 400 times for each network size. The optimization routine was repeated 100 times. In Fig. 5.13 and Fig. 5.14 the resulting performance distributions are depicted for the one-step cross-predictions (NMSE_1) and the free-running cross-predictions (NMSE_∞).

As can be seen the performance of the networks can be improved by the addition of elements. This is naturally true as the diversity of the internal dynamics is higher in a network with $M = 400$ elements than in a network with only $M = 100$ elements. However, even a network with such a large number of elements as $M = 400$ is inferior to the smaller networks with $M = 60$ if the latter are optimized. As can be seen in Fig. 5.13 and Fig. 5.14, the optimized networks on the average outperform the randomly created ones. Only the best random networks with $M = 400$ elements can compete with an average optimized network.

We can conclude from this numerical experiment that randomly created networks can have very good performance values, which can even be further improved by raising the number of elements. However, if performance is im-

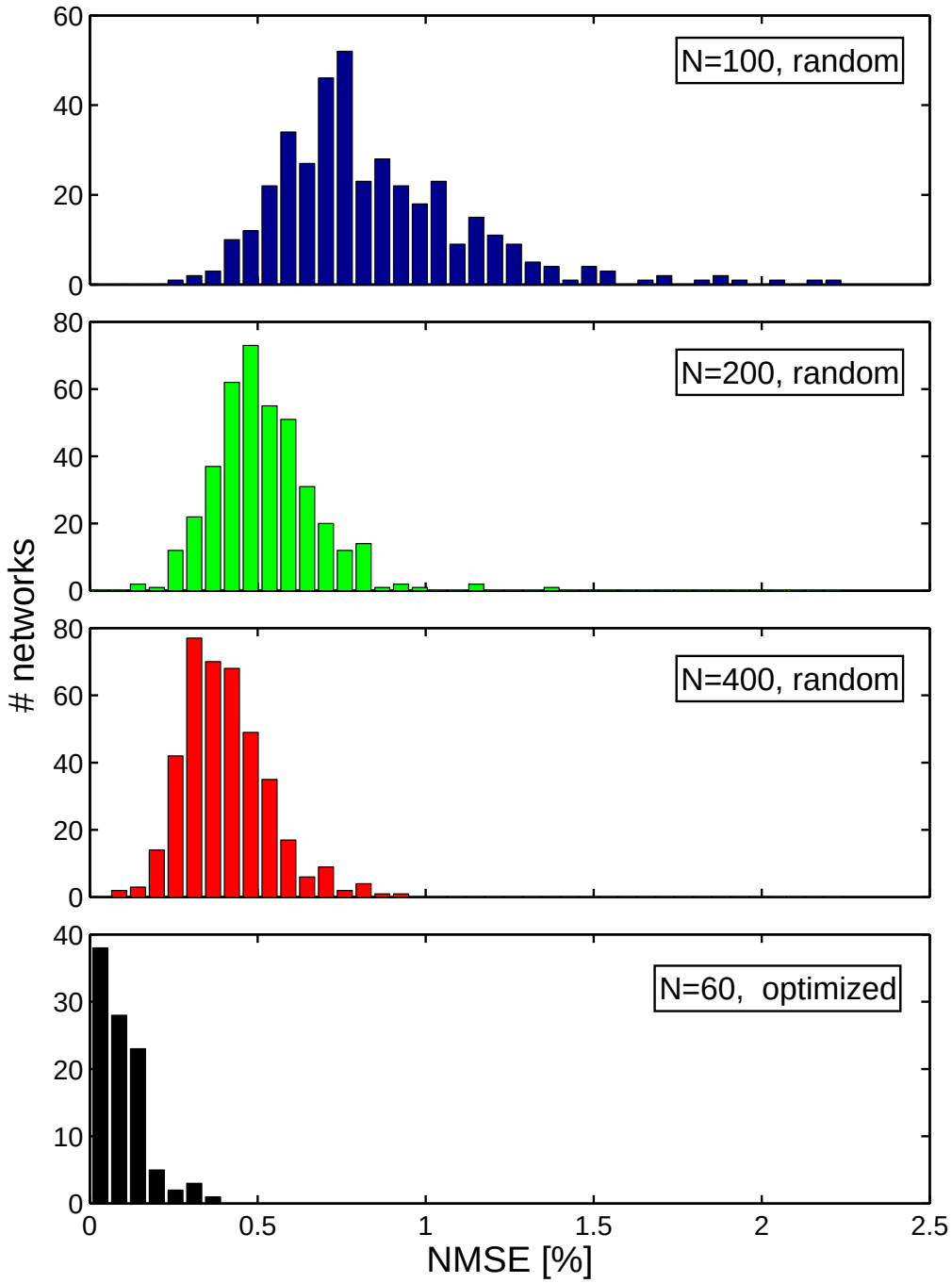

Figure 5.13: Performance of networks measured with NMSE_1 (one-step cross-predictions). The networks were either created with random internal connections or with optimized connections. The number of elements in the randomly connected networks varies between $N = 100$, $N = 200$, and $N = 400$. The number of elements in the optimized networks is $N = 60$. Note that only 100 optimized networks were produced, while the number of the random networks was 400 for each network size.

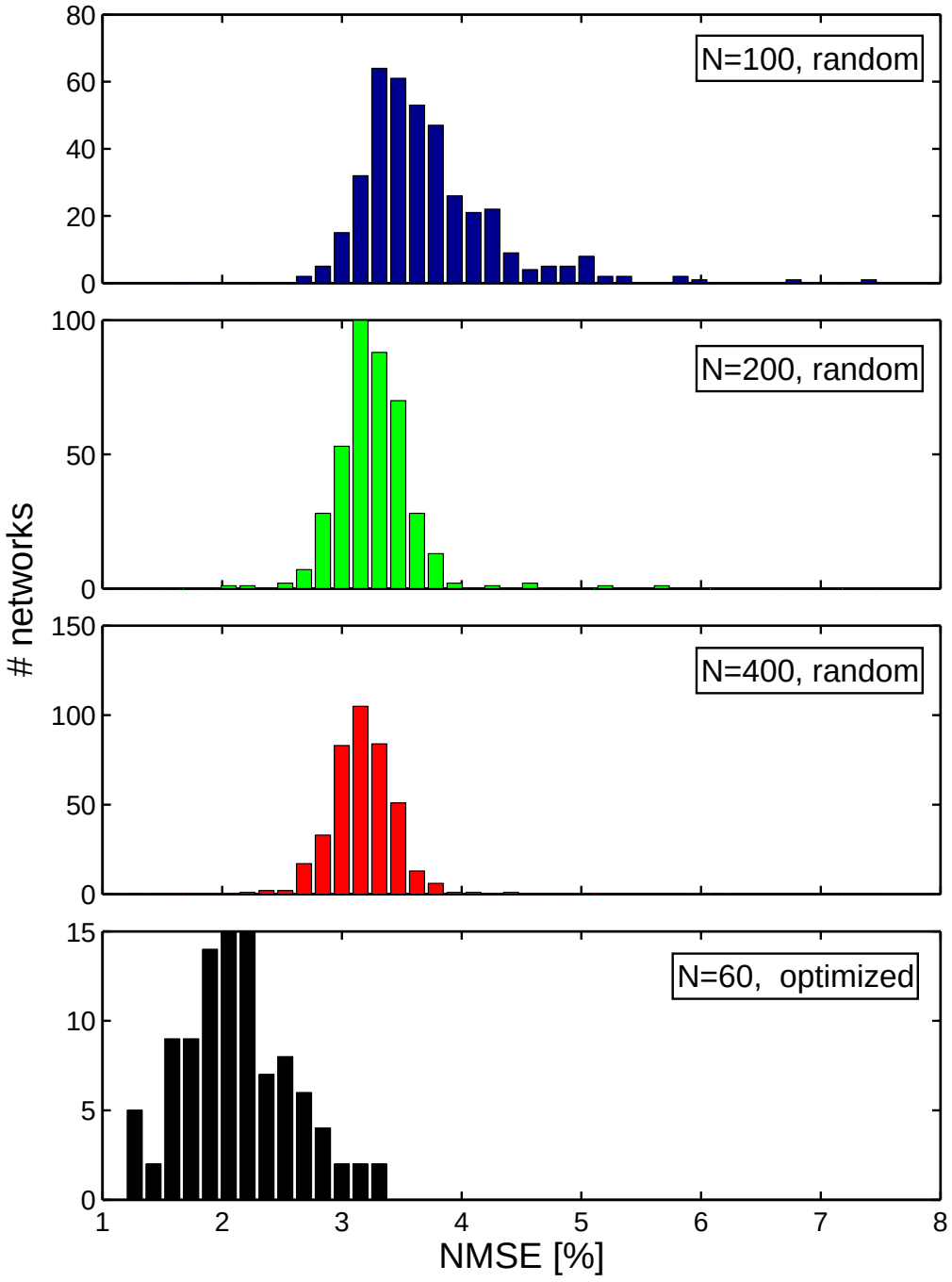

Figure 5.14: Performance of the same networks as in Fig. 5.13. However, this time measured with NMSE_∞ (free-running cross-predictions).

portant and the time spent for the optimization routine is not an issue, the connection structure of a network should always be optimized, because this leads to smaller and faster networks, which are in most cases more precise than random ones. Echo State Networks, which are huge and slow but need no optimization routine, and classical recurrent networks, which are small and fast but invest a great deal of computation effort in optimizing every single internal connection, are only two extreme sides of the modeling approach with recurrent networks. It seems that the middle way can integrate the advantages of both sides very well. A greater number of elements can enrich the internal dynamics of a recurrent network and thus make computationally intensive algorithms like the BPTT superfluous. A coarse optimization can help in keeping the network size in sensible bounds.

Chapter 6

Friction

Friction is a phenomenon that affects our everyday life. When the surface of an object slides across the surface of another, friction is the tangential force that opposes the motion. This force is sometimes beneficial. In a world without friction shoelaces would open permanently, screws would fall from the walls, pedestrians would slip on the pavement, and car breaks would fail miserably. The functionality of many important things depends on friction. Nevertheless, more often we deal with the downsides of friction. Mechanical devices like watches devices have to be supplied with energy otherwise they stop working. A great part of the fuel in cars is wasted in the effort to compensate for frictional forces. Drills annoy us with noise and have to be cooled in order to function properly in permanent operation. Most of the negative effects of friction can be subsumed under unwanted energy dissipation, noise emission, and wear of mechanical devices. The perhaps most frightening effect of friction are earthquakes. When tectonic plates slide over each other, friction forces let the earth tremble, often with devastating effects on a geographical scale.

One way or the other, the importance of friction in our world simply cannot be overestimated. The many aspects of this ubiquitous phenomenon have fueled the interest of researchers to investigate it and to develop theories and models to describe its nature. Aside from scientific curiosity, this interest has been furthered by problems and practical concerns stemming from the development and production departments of the industry. The car industry, for example, has always tried to increase friction effects to improve the efficiency of breaks or tires. On the other hand the manufactures of mechanical or optical storage devices, have tried to reduce friction effects in order to enhance the precision in the movements of read heads. Even though much research has been done, applicable friction models are still in great demand by engineers.

The first section of this chapter (Section 6.1) serves as an introduction to the field of friction. The reader is familiarized with the main phenomena. Furthermore, common theories and modeling approaches are presented, which try to describe frictional behavior. The purpose behind it is to demonstrate

the complexity of friction and to lay the conceptual basis for the sections to come.

Following the introduction, results in the context of black-box modeling with static and dynamical models are presented. These results were obtained in the scope of the VW-Stiftung project No. 1/76938. Cooperating were the Katholieke University of Leuven, Belgium, the University of Sheffield, UK, the University of Patras, Greece, and our University of Göttingen, Germany. Specifically, Section 6.2 deals with the modeling of experimentally measured pre-sliding friction. The results were published in [55]. As a pre-step, modeling of pre-sliding friction has also been done on numerically simulated data. The results of these simulations have led to publications in in [4] and [5]. Section 6.3 extends the modeling attempts from the pure pre-sliding to the sliding regime. Again, experimental data was used for modeling and the results were published in [70]. The last section (Section 6.4) deals with control of plants whose movement is affected by friction. Here, the experience and the results from modeling of friction were applied to a numerical simulation of a tracking problem.

6.1 Friction phenomena and models

The field of research that investigates friction is called *tribology*, from the Greek word $\tau\rho\iota\beta o\sigma$, meaning *rubbing*. Although the naming is not older than 40 years, investigations started much earlier. Leonardo da Vinci (1452–1519) was one of the first to study friction phenomena in a systematic way. He made two remarkable observations: If a body is dragged across a surface,

- the friction force does not depend on the size of the contact area.
- the friction force is proportional to the normal force.

Unfortunately da Vinci's ideas concerning friction were not made public and thus went unnoticed by the science world. Guillaume Amontons (1663–1705) rediscovered the two basic laws of da Vinci and added some of his own observations to them [8]. Since then the laws are referred to as *Amontons' laws*. John Theophilius Desaguliers (1683–1744), Leonard Euler (1707–1783), and Charles Augustin Coulomb (1736–1806) extended the knowledge about friction [24], [26], [21]. The concept of *adhesion* was introduced and another law added:

- The friction force is independent of the sliding velocity.

These individual laws tried to capture the essence of friction. However, their greatest weakness was that they were purely empirical and, unlike Newton's laws, nothing fundamental could be deduced from them. Nevertheless,

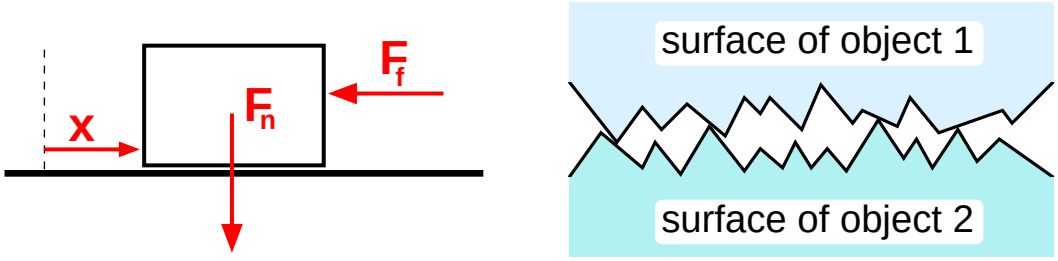

Figure 6.1: Left: The contact between an object and the surface beneath it is influenced by the normal force F_n (weight of the object or normal load). If the object is dragged a certain distance x across the surface, its movement is opposed by a force F_f, which is called friction. **Right:** Physical theories explain friction as a microscopic effect caused by asperities on the surface of objects. On a microscopic scale every surface, no matter how well polished, is uneven. If two objects are brought together, the effective area of contact is much smaller than the surface area of both objects because they touch only at the protrusions in the surface.

scientists persisted in their efforts to find generalized rules concerning friction. Their work resulted in some remarkable achievements but, unfortunately, was not crowned with success as there is still no satisfactory model for friction. None of the models developed so far has the ability to reproduce all aspects of friction that were found in experiments. The difficulty lies in the fact that friction is a highly nonlinear phenomenon and includes processes on many scales, demanding a theory that can integrate all the different levels. Up to now, models of friction usually describe only some of its aspects in good agreement with experiments while being inaccurate in others. As long as there is no profound theoretical model for friction, researchers have to resort to heuristic models, which are based on experimental observations. In the following we present some of the most important friction phenomena accompanied by corresponding modeling approaches. For a deeper review of friction effects and the development of models see for example [9] and [51].

In 1950 Bowden and Tabor investigated friction in systematic experiments and proposed a physical explanation for the friction laws [14], [15]. They surmised that the effective contact zone of two objects touching each other is not identical to the contact area but much smaller. The reason for this can be seen schematically on the right hand side of Fig. 6.1. On a microscopic scale even very smooth surfaces reveal a certain roughness. As a consequence, when two surfaces touch, contact is established only at discrete protruding points, which are called *asperities*. That means that the real contact area is necessarily smaller than the apparent one. If the normal load is increased the asperities are deformed and the effective area of contact grows. This explains the dependence of friction on the normal load rather than on the apparent contact surface.

Consider the simple example of an object lying on an arbitrary surface (see lhs of Fig. 6.1). If the object is moved across the surface with the velocity $v = \dot{x}$, this movement is obstructed by the friction force F_f. Classical models describe the F_f as a function of the relative displacement x and the relative velocity $v = \dot{x}$ between the two surfaces in contact. Typical for these kinds of models is a discrimination between *static friction*, or short *stiction*, for the case $v = 0\,\mathrm{m/s}$ and *dynamic friction* for the case $v \neq 0\,\mathrm{m/s}$.

Stiction is trivial to model if the external force that is applied to the object is known. The friction force simply counteracts the external force and precludes the object from moving. Static friction can usually reach higher magnitudes than dynamic friction and, naturally, it cannot be described as a function of velocity v. The classical way of modeling stiction is

$$F_{\mathrm{static}} = F_e\,, \tag{6.1}$$

with the precondition $v = 0\,\mathrm{m/s}$ and F_e being the external force applied to the moving object (see lhs of Fig. 6.2). At some critical force value F_s, which is called *breakaway force*, the object starts to glide across the surface. If this happens, the friction force drops rapidly to a lower value and from then on has to be described in the context of dynamic friction.

Things get more complicated when the object's dynamics transits between the static and the dynamic regime. Then the breakaway force F_s is not a constant but changes its value depending on the *dwell time*, i.e. the time span in which the relative velocity of the two surfaces is zero [9]. During the dwell time the adhesion forces between the asperities of both surfaces is established. If it is short, the contacts are weak and the breakaway force is small. If it is long, all contacts have enough time to build up strong adhesion forces. Then the tangential force needed to break these bonds is much higher.

Even though no sliding occurs in the static regime, it is found experimentally that the external force effects a displacement of the object [64]. It is explained by an elastic deformation or shearing of the asperities. For external force values smaller than the breakaway force the asperities behave similar to nonlinear springs, resulting in a displacement of the object. Since the contacts do not break and thus no real sliding occurs, these tiny displacements are referred to as *pre-sliding* or *micro-sliding*.

However, limiting the deformations of the asperities to be only elastic is a gross oversimplification. Additionally to the elastic deformations pre-sliding involves also plastic deformations. An increase with a following decrease of the external force results in an non-vanishing effective displacement of the object (see rhs of Fig. 6.2). Some of the asperities retain their deformations without returning to their previous state. Thus, a memory effect is introduced to pre-sliding [22]. The result for an alternating external force application is a hysteresis curve, which is typical for the pre-sliding regime. The special feature of the hysteresis curve is its non-local aspect. The future values of the

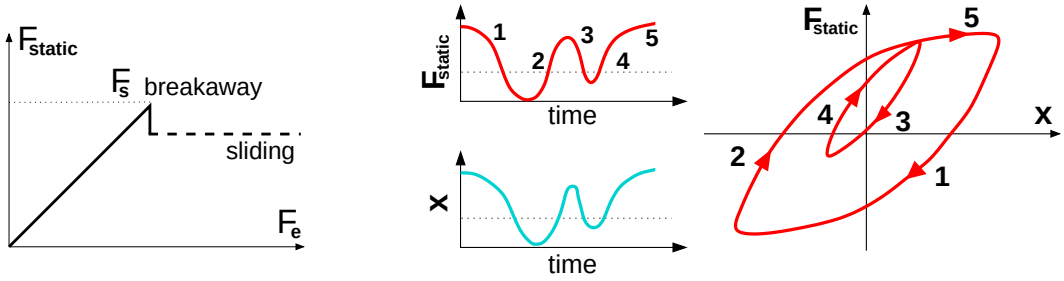

Figure 6.2: Left: In the static regime the friction force F_s simply compensates the externally applied force F_e. If a critical value F_s is reached, the object starts to slide and the magnitude of the friction force usually drops to a lower value. **Right:** Although the regime is called static, an externally applied force F_e effects a displacement x of the object. It is caused by elastic and plastic deformations of contacting asperities on both contacting surfaces. The plastic deformations prevent the object from returning to its previous position. It retains an effective displacement. If an alternating force signal is applied, this behavior leads to a hysteretic curve. The latter is attributed with a non-local memory: if the inner loop (3,4) is closed the hysteresis follows again precisely the track of the outer loop (5,1,2). This shows that the system 'remembers' its previous extremal displacement from the past.

friction force are not uniquely defined by the actual friction value and the future displacements of the object. Instead, the history of the past displacements is important for the future behavior (see for example [11]).

In contrast to static friction, dynamic friction is a function of velocity and not of displacement. The so called *Coulomb friction*

$$F_f(v) = F_c \cdot \mathrm{sign}(v), \tag{6.2}$$

is a discontinuous function that connects velocity with dynamic friction. It states that for steady velocities ($v = \mathrm{const}$) the friction force does not depend on the magnitude of the velocity but has a constant value F_c and is opposed to the direction of movement. Usually, *viscous friction* is added to Eq. (6.2) (see Fig. 6.3), resulting in

$$F_f(v) = F_c \cdot \mathrm{sign}(v) + F_v \cdot v\,. \tag{6.3}$$

The viscous part is proportional to the velocity and is inspired by the viscous friction force from hydrodynamics. It is applicable if the surfaces are not dry but coated with lubricants.

Combining static and dynamic friction together in one equation leads to a multi-valued function

$$F_f = \begin{cases} F_{\mathrm{static}} = F_e & \text{if } v = 0\,\mathrm{m/s} \quad (|F_e| < F_s) \\ F_{\mathrm{dynamic}}(v) = F_c \cdot \mathrm{sign}(v) + F_v \cdot v & \text{else} \end{cases}, \tag{6.4}$$

which is shown in Fig. 6.3. In the static regime the friction force is independent of the velocity. As soon as the breakaway force is exceeded and the object starts

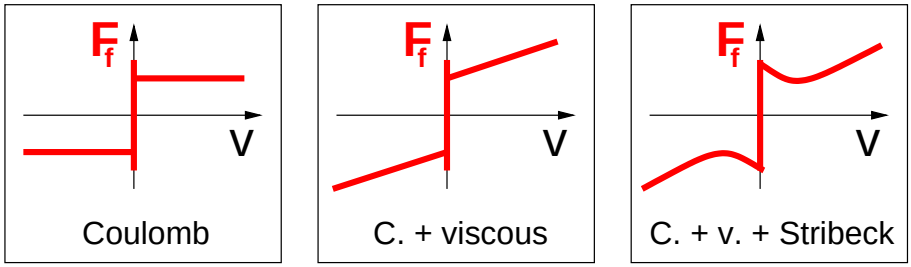

Figure 6.3: The dependence of the friction force F_f on the velocity v in the stationary limit ($v = \text{const}$). All three functions are multi-valued in the region of static friction ($v = 0\,\text{m/s}$). Here the friction force is a function of the position rather than of velocity and cannot be properly displayed in the diagrams. **Left:** The Coulomb friction models the dry friction case. The friction force is constant and only depends on the direction of the velocity. **Middle:** If a lubricant is present, the Coulomb friction is augmented by the viscous friction, which is proportional to the velocity. **Right:** The Stribeck curve takes into account that the transition from static friction to dynamic friction is continuous.

to glide, the friction force drops instantaneously to a lower value. This sharp drop in magnitude for an object that is just starting to move is not confirmed by experimental observations. Rather, it was found by Stribeck that the velocity dependence is continuous [66]. A common form for the dependence is

$$F_{\text{dynamic}}(v) = F_c \cdot \text{sign}(v) + F_v \cdot v + (F_s - F_c)e^{-|v/v_s|^{\delta_s}}\,, \tag{6.5}$$

where the Stribeck velocity v_s and the parameter δ_s are adjusted to experimental data. The typical minimum (maximum) in the Stribeck curve for positive (negative) velocities in Fig. 6.3 is called *Stribeck effect*.

The model in Eq. (6.4) is referred to as *classical static model*. Although it describes many of the aspects of friction, it is not well suited for simulations or control tasks in practice. One drawback is that the external force F_e is used as an input in the static regime. However, in many applications F_e is not always explicitly given. Instead, one often wishes to have a functional relationship between the friction force and the displacement of the object. This relationship can be deduced in the dynamic regime from the velocity without problems. However, in the static regime the relationship is more complicated as it involves memory effects. Another drawback is the strict distinction between the pre-sliding and the sliding regime. During a numerical simulation it is notoriously difficult to decide at which point a system has zero velocity. Although Karnopp proposed a simple cure for this problem by defining a region $|v| < v_{\text{bound}}$ for stiction in [40], his solution is only roughly approximating the real situation.

Perhaps the greatest weakness of Eq. (6.4) is its inability to describe friction for non-constant velocities. It was shown by Hess and Soom in [32] that a hysteretic relationship between velocity and friction can be observed if the

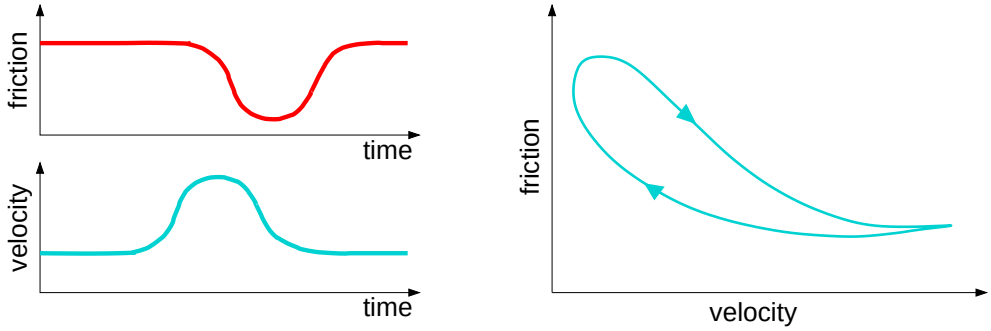

Figure 6.4: Left: If the velocity of the object is changed, the friction force does not follow instantaneously but with a delay. This is referred to as *friction lag*. **Right:** The friction lag leads to a hysteretic behavior for an alternating velocity.

magnitude of the velocity is changed alternately (see Fig. 6.4). The reason for the hysteresis curve is a temporal lag in the reaction of the friction system. If the velocity of a sliding object is changed, the friction force changes its value not simultaneously but with a delay, which is referred to as *friction lag*. The classical static model in Eq. (6.4) is not able to explain these findings.

To overcome the shortcomings of the static models, attempts were made to describe friction by dynamic models. One of the basic models is that of Dahl [23]. In its common form it describes friction as a function of the displacement

$$\frac{\mathrm{d}F_f}{\mathrm{d}x} = \sigma \left(1 - \frac{F_f}{F_c}\mathrm{sign}(\dot{x})\right) , \tag{6.6}$$

with σ being the *stiffness coefficient* of the asperities and F_c the Coulomb friction. Using $\dot{F} = (\mathrm{d}F/\mathrm{d}x)\dot{x}$ Eq. (6.6) can be reformulated as a time dependent function

$$\dot{F}_f = \sigma \left(1 - \frac{F_f}{F_c}\mathrm{sign}(\dot{x})\right)\dot{x} = \sigma \left(\dot{x} - \frac{F_f}{F_c}|\dot{x}|\right) . \tag{6.7}$$

Another reformulation results in the known state space model

$$\begin{aligned} \dot{z} &= \dot{x} - \frac{\sigma|\dot{x}|}{F_c} z \\ F_f &= \sigma z , \end{aligned} \tag{6.8}$$

with one internal state z. For a steady state $\dot{z} = 0$ the friction is given by

$$F_f = F_c \,\mathrm{sign}(\dot{x}) , \tag{6.9}$$

which is exactly the Coulomb force in the dynamic regime. The advantages of Dahl's model are its simplicity and its ability to incorporate both the static and the dynamic friction regime without having to resort to switching. Its disadvantages are a moderate accuracy in the description of the static regime

and the lack of features like the Stribeck effect. The Dahl model was used in many applications and served as a basis for some more elaborate models. The most popular among these models is the *LuGre model*[1] [16]. As an extension to the Dahl model in Eq. (6.8) the LuGre model is similar in its state equations

$$\begin{aligned} \dot{z} &= \dot{x} - \frac{\sigma_0|\dot{x}|}{g(\dot{x})}z \\ F_f &= \sigma_0 z + \sigma_1 \dot{z} + \sigma_2 \dot{x}\,, \end{aligned} \tag{6.10}$$

with the extra micro-viscous part $\sigma_1\dot{z}$ and the viscous part $\sigma_2\dot{x}$. The function $g(\dot{x})$ in Eq. (6.10) models the steady-state behavior for constant velocities. It is usually set to

$$g(\dot{x}) = F_c + (F_s - F_c)e^{-(\dot{x}/v_s)^2} \tag{6.11}$$

to reproduce the Stribeck effect with a Gaussian function. Choosing damping parameter σ_1 to be constant as in Eq. (6.10) is called *standard parameterization* of the LuGre model. Choosing this parameter as dependent on the velocity introduces dissipation

$$\sigma_1(\dot{x}) = \sigma_1 e^{-(\dot{x}/v_d)^2}\,, \tag{6.12}$$

as was shown by Olsson in [50]. The LuGre model is able to model many of the experimentally observed friction phenomena, like static friction behavior, friction lag, Stribeck effect, and stick-slip behavior. However, prediction of experimental data, especially in the pre-sliding regime, could not be achieved by this model with a satisfying accuracy.

Swevers *et al.* suggested in [67] another extension, known by the name of *Leuven model*, that deals with some shortcomings of the LuGre model. In particular, the Leuven model incorporates additional free parameters, which enable the fitting of arbitrarily shaped transition curves, i.e. the functional relationships between displacement and friction force. For the description of the non-local hysteresis in the pre-sliding regime the Leuven model uses memory stacks that store the minima and maxima of previous force values.

To avoid the rather inconvenient computations with stacks, Laempert *et al.* proposed in [47] a modification that is based on the Maxwell-slip model. In a refined version in [45] and [6] this model is referred to as *Generalized Maxwell-slip (GMS) model*. The main idea of the Maxwell-slip model is to simulate a fixed number M of plasto-elastic springs in parallel, each with its own characteristic hysteretic behavior (see Fig. 6.5). The GMS model extends the Maxwell-slip model by substituting the springs with massless block-springs. These elemental blocks behave in the static regime similar to the springs but show additionally sliding characteristics of friction systems, e.g. the Stribeck

[1] The naming 'LuGre' is derived from the first letters of the two cooperating universities: **Lu**nd in Sweden and **Gre**noble in France.

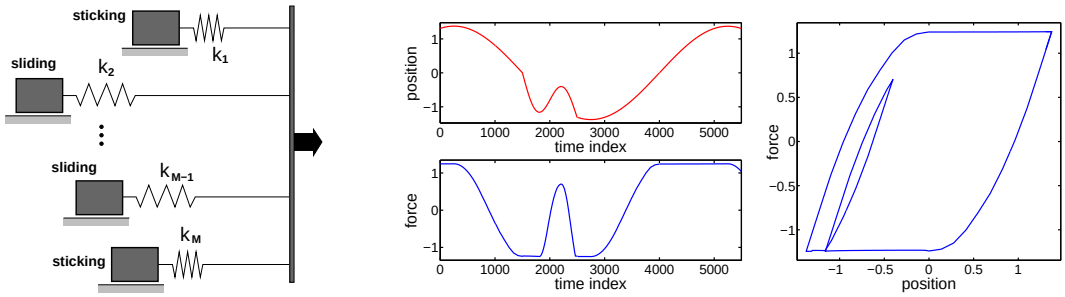

Figure 6.5: Left: Generalized Maxwell-slip model consists of M elemental, massless block-springs acting in parallel and following their own dynamics. The total friction force of the whole system is a superposition of the individual forces. **Right:** Position and force signal from a simulation of a GMS model consisting of $M = 10$ elements. When plotted against each other, the result is a hysteresis with non-local memory, which is typical for the static regime.

effect. For the i-th element the following state equation is given for sliding

$$\dot{F}_i = \mathrm{sign}(\dot{x}) C \left(1 - \frac{F_i}{\nu_i g(\dot{x})} \right) , \tag{6.13}$$

with C a constant parameter and $g(\dot{x})$ defined in analogy to Eq. (6.11), representing the steady state behavior for constant velocities. For compatibility reasons the individual parameters ν_i have to fulfill the condition

$$\sum_{i=1}^{M} \nu_i = 1 \, . \tag{6.14}$$

The element remains slipping until its velocity goes through zero. Then the element sticks and the state equation is given by

$$\dot{F}_i = k_i \dot{x} \, , \tag{6.15}$$

with a constant individual parameter k_i (stiffness). The element remains sticking until $F_i = \nu_i g(\dot{x})$. The total friction force results by superimposing the outputs of the individual elements and adding an extra term that accounts for the viscous friction:

$$F_f(t) = \sum_{i=1}^{M} F_i(t) + \sigma_2 \dot{x} \, . \tag{6.16}$$

The GMS model is able to describe the pre-sliding regime as well as the sliding regime with good accuracy. All major phenomena of friction are captured, e.g. breakaway force, stick-slip behavior, or friction lag. Note that by using a population of individual elements for the simulation of friction, the modified Leuven model is only a few steps away from the physics based models, which are described in the remaining paragraphs of this section.

The presented friction models so far have been all empirically motivated. Their structure was developed in a heuristic approach to describe the experimentally observed phenomena. Thereby, the empirical models have two somewhat contradicting goals. On the one hand they try to describe frictional behavior with preferably good agreement to experiments. On the other hand they try to retain a simplicity that makes them applicable in online control tasks. This compromise between accuracy and simplicity is sometimes a disadvantage if one is interested specifically in the dynamics of friction. Therefore, an alternative approach exists to modeling friction. It starts with modeling the physical processes that happen at the microscopic level. The experimentally observed friction effects, like Stribeck effect or hysteresis curves, are then an emergent feature of such physically motivated models. Since these models involve the simulation of populations of microscopic processes, they are computationally much more demanding than the empirical models and therefore too unwieldy to be used for applications like control. However, their advantage is that they enable a deeper insight into the mechanisms behind friction.

It is generally accepted that the deformation of asperities on the rubbing surfaces and the mutual adhesion between them is responsible for friction, as proposed by Boden and Tabor [14], [15]. Haessig and Friedland introduced in 1991 the so called *bristle model*, which tries to model the behavior of the mutual contact points at the level of the asperities [29]. The touching surfaces are modeled as randomly littered with flexible bristles (see lhs of Fig. 6.6). If the surfaces are moved against each other, the bristles are deformed. The contact between each individual bristle breaks if a certain threshold deformation is reached. Since the bristles act like springs, each of them contributes to the friction force resisting the motion. The dependency on the velocity is captured by reducing and increasing the number of random bristles according to the speed of movement. The bristle model is found to yield good results and can also capture the randomness present in the friction process.

After the introduction of the bristle model many similar approaches were developed that tried to include even more of the friction phenomena. Recently, Al-Bender *et al.* from the KU Leuven proposed a theoretical model, which captures most of the friction effects in one unifying formulation [7], [6]. The basis of their model are idealized asperities with certain geometrical and elastic attributes. During an application a population of thousands and more of such asperities is used for simulating the frictional behavior of surfaces. Thereby all elastic properties are attributed to the moving surface on top, while the lower surface is simulated as a rigid profile. Before the simulation is started, a randomly uneven profile is chosen for the lower surface and the asperities are also placed arbitrarily across the moving surface. All attributes are randomly assigned to each of the asperities. On the right hand side of Fig. 6.6 the schematics and the average life-cycle of an asperity is depicted. Every asperity has an individual mass m, height h, tangential stiffness k_t, and normal stiffness

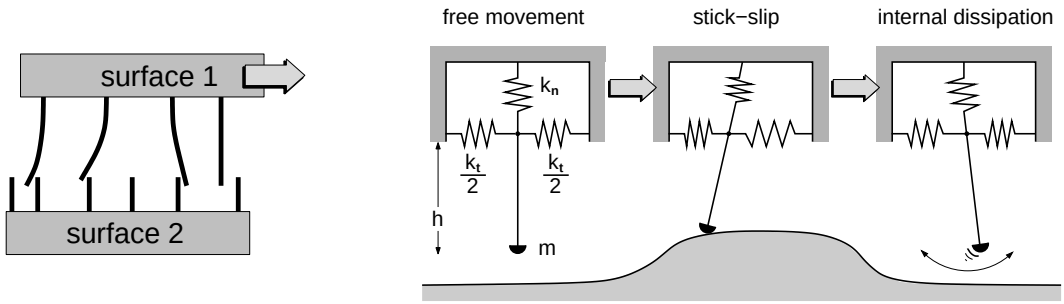

Figure 6.6: Left: Bristle model proposed by Haessig and Friedland. Flexible bristles, which are randomly placed on the rubbing surfaces, represent the asperities on a micro-scale level. If the surfaces are moved against each other, the bending of the bristles evokes a force (friction) that opposes the motion. **Right:** Generic friction model proposed by Al-Bender *et al.*. The asperities are modeled with individual masses m and heights h, displaying elastic behavior with a normal stiffness k_n and a tangential stiffness k_t. In a typical life-cycle the asperity transverses from free movement to an active state, where it establishes contact to the lower surface, and back again to free movement. During contact the asperity exhibits a stick-slip behavior. Directly after breaking loose from the surface all stored elastic energy is dissipated by means of internal processes.

k_n. During the movement of the surface the asperities transition from states in which they have contact and states in which they move freely. If an asperity establishes contact, it sticks to the rigid profile. The external force applied to the asperity increases until the breakaway threshold is reached. Then it is dragged across the surface (sliding). The breakaway force is modeled as being dependent on the dwell-time of the asperity. After gliding across the profile the asperity looses the contact at some point. If this happens, the asperity vibrates in tangential and normal direction while dissipating the stored elastic and inertial energy. During the simulation the state of every asperity is updated every time step. By calculating the work that was performed between two time steps by an asperity and dividing it by its relative displacement, the contribution to the friction force can be acquired. Summing up all contributions yields then the total friction force. As shown in [7] and [6], the so called *generic friction model* (GF model) is in excellent agreement with most of the friction phenomena.

6.2 Modeling of pre-sliding friction

As already mentioned, friction in the pre-sliding regime is dominated by the position and not the velocity of an object that is rubbed against an arbitrary surface. The micro-sliding occurring in this regime is a result of the applied force. Theoretically, the dependence between position and force should be easy to model. However, the nonlocal memory inherent in this system poses some problems. If a force is applied to the friction object, it changes its

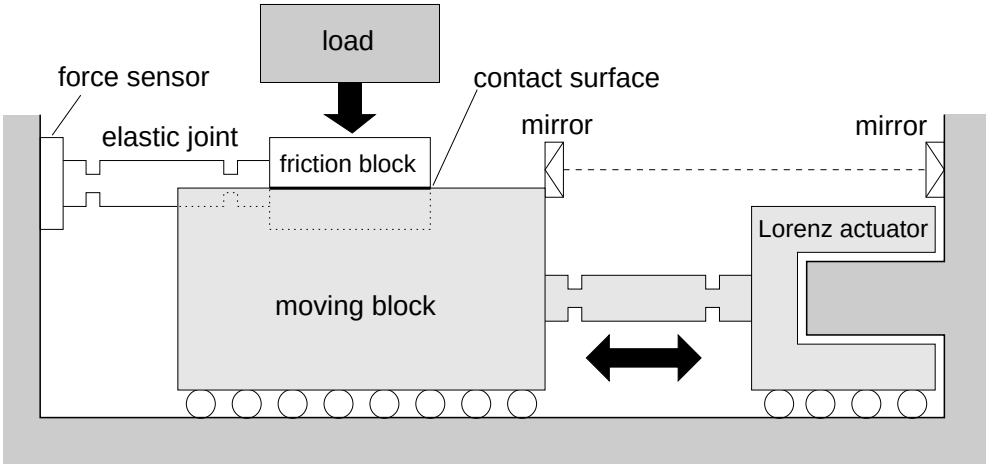

Figure 6.7: Schematic diagram of the tribometer setup with three main components: **actuator part:** Lorenz actuator, stinger, and moving block **friction part:** friction block, elastic joint, and force sensor **loading part:** load, exerting normal force on friction block. The friction force was measured by a force sensor (piezo crystal), while the position of the moving block was determined optically with a laser interferometer (fixed and moving mirror).

position depending on previous force applications. Formally, the system has an inner state that retains the history of the driving signal to some extent. This behavior must be captured by the model if it is to describe pre-sliding friction correctly.

6.2.1 Experimental setup

The experimental friction data was measured with a tribometer developed by the Department of Mechanical Engineering, Division P.M.A., KU Leuven, Belgium. The description of the setup follows [55] (see also [46]). In Fig. 6.7 a schematic of the tribometer is shown, which consists mainly of three parts: the actuator part, the friction part, and the loading part. The actuator part and the friction part are coupled by the friction interface, which is the object of interest. All other mutual influences between the three parts are minimized as much as possible by the chosen setup in order to guarantee a precise measurement of the friction force. Especially a decoupling between the loading part and the frictional part is established by the usage of an air-bearing. In this way all tangential forces arrive at the force sensor.

A Lorenz actuator in the actuator part drives the moving block with the help of a stinger, while the exact displacement of the moving block is measured with a Renishaw laser interferometer. The latter measures the distance between the mirror fixed on the moving block and the one fixed on the frame. By feeding back the actual position of the moving block through a controller

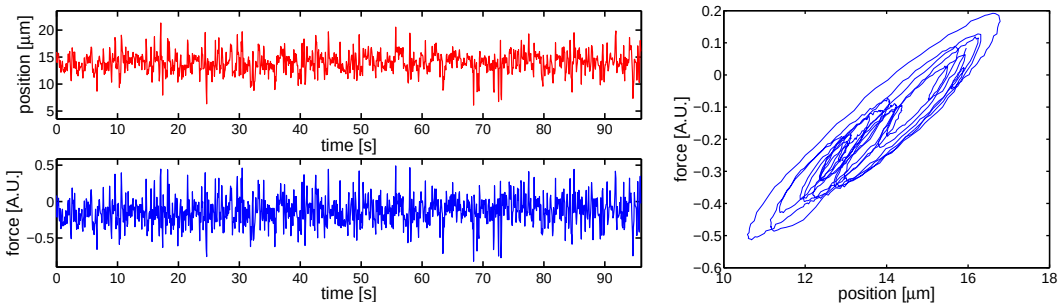

Figure 6.8: Experimentally obtained position and friction force, measured for a time period $T = 96\,\mathrm{s}$ and with a sampling frequency $f_s = 250\,\mathrm{Hz}$. **Left:** Position and friction force plotted against time. **Right:** Friction force plotted against position. The hysteresis effect is clearly visible.

the Lorenz actuator can be used to impose either a specific force signal or a desired displacement trajectory.

The friction force between the contacting surfaces of the moving block and the friction block is measured by the friction part. It consists of the friction block, an elastic joint, and a force sensor. The joint comprises two pairs of elastic hinges with the purpose of setting off small vertical, lateral, and rotational alignment errors of the friction block. Thus, the friction part in the longitudinal direction follows the principle of minimal compliance. Only if this stiffness property is valid, the relative displacement between the moving block and the friction block can be considered equal to the absolute displacement of the moving block. Also in this case the force measured by the force sensor in the friction part equals the friction force without influence from the inertial force. As verified experimentally, a peak-to-peak force value of 10 N in the setup in Fig. 6.7 results in a displacement equal to $0.12\,\mu\mathrm{m}$.

6.2.2 Training and testing

The position signal $P(t)$ and force signal $F(t)$ were measured for a time period $T = 96\,\mathrm{s}$ and with a sampling frequency $f_s = 250\,\mathrm{Hz}$. In Fig. 6.8 the two resulting time series, position $\{P_t\}_{t\in\mathbb{I}}$ and force $\{F_t\}_{t\in\mathbb{I}}$, $\mathbb{I} = \{1, \ldots, N\}$, are shown with $N = 24000$ data points each. The position P_t was measured in μm while the friction force F_t was deduced from the voltage output of the force sensor. Since the scaling of the force signal is not important for our modeling process, we will leave the units of the force time series as arbitrary units (A.U.).

Looking at the force signal, one might notice a slow trend in the data. This trend can be made visible by looking at the mean part of the time series. In Fig. 6.9 the force signal is shown after it was filtered by a symmetrical moving average filter with an effective length of $N_{\mathrm{eff}} = 2001$ data points. As can be clearly seen, the mean of the force signal is not constant but slowly

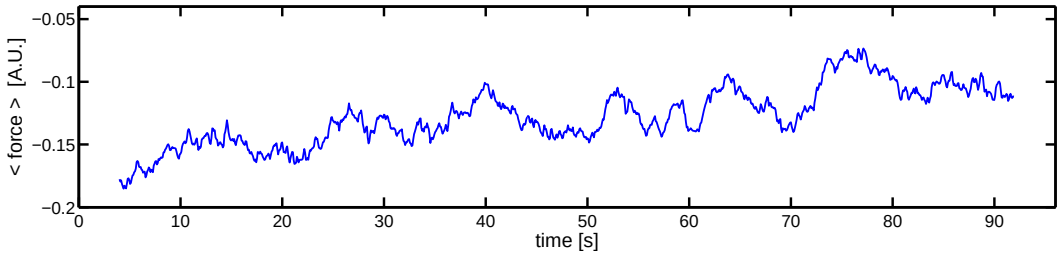

Figure 6.9: Filtered force signal indicates a trend in the data. The filtering was done with a symmetrical moving average filter having a window length of 2001 data points. Instead of zero-padding at the beginning and the end of the time series, the first and last 1000 data points were left out. Note, the scaling of the y-axis is different from the one in Fig. 6.8.

rises with time. Obviously, the force signal is non-stationary and has a slight trend towards higher values. The trend is explained by leakage of charge from the force sensor (piezo crystal)[2]. To make things worse, the trend does not continue linear in time but saturates slowly. Therefore, it is not straightforward to adjust the measured data to compensate for the artificial trend.

We followed two different approaches to solve the 'trend problem'. In the first approach we tried to detrend the data by subtracting the part which linearly grows in time, knowing that this could improve the modeling results for black-box models but also that we might introduce new inconsistencies. In the second approach we left the measured data untouched and saw the trend as part of the modeling problem. Although this second approach is more inconvenient for black-box models, it might be the more realistic one. Non-stationary data with biased measurement errors are not the kind of data modelers hope for but they reflect real-life conditions, which have to be regularly coped with.

A cross-prediction from the position signal to the force signal was chosen as model objective. It translates to determining the actual force value F_t at time step $t \in 1, \ldots, N$ that led to the actual position P_t. The prediction of the force value was constrained to be based solely on the position signal, meaning that free-running models had to be employed. This model objective was chosen because it leads to models that can be used for control (see Section 6.4).

The data points of both time series, position and force, were split into the three classical sets for modeling: the training set, the validation set, and the test set. Each set was chosen to have the same size, resulting in all sets containing 8000 data points. The training and the validation set were both applied for adjustment of the model parameters. The test set was not used during the training phase of the model but was only presented afterwards for the evaluation of the generalization abilities of the model. The quality measures were the usual NMSE and MAX criteria based on free-running predictions. To assure

[2]Unfortunately, this measurement error was unavoidable for measurements in the static regime.

the stability of the models, the free-running predictions were started at the very first force value F_1 and followed through until the last value F_{24000}. The NMSE was then computed on the last 8000 data points, representing the test set,

$$\text{NMSE} = \frac{100\%}{8000 \cdot \sigma_F^2} \sum_{t=16001}^{24000} (F_t - \hat{F}_t)^2 \,, \tag{6.17}$$

with F_t being the measured force signal at time step t and $\hat{F}_t$ the free-running prediction produced by the model. Similarly the MAX criterion was computed as

$$\text{MAX} = \frac{1}{\sigma_F} \max \left\{ |F_t - \hat{F}_t| \;\mid\; t = 16001, \ldots, 24000 \right\} . \tag{6.18}$$

6.2.3 Results

The results gained with NARX models and RNN models are shown in Fig. 6.10. The first presented model is a polynomial NARX model, which was developed with the FOR technique and optimized on free-running predictions. It has 32 monomials with a maximum degree $p_{\max} = 3$ as basis functions and is using the following targets and regressor vectors in the embedding

$$y_t = F_t \,, \quad \boldsymbol{x}_t = (P_t, P_{t-2}, \ldots, P_{t-18}, F_{t-2})^T \,, \tag{6.19}$$

with ten values of the position time series and one value of the force signal as components in the regressor. The evaluation of the model on the test set yielded the quality criteria NMSE = 2.49% and MAX = 0.41, indicating that the model describes the system fairly well but fails in recognizing all of the characteristics. A look at the errors for the whole test set reveals the reason for this failure. The model cannot adapt to the slight trend in the force signal. The free-running predictions tend to be smaller than the original values. In the course of time the errors are bound to grow even more because the trend is not accounted for by the model.

The results for the polynomial NARX model could be improved by removing the linear trend from the friction force. As said before, this pre-processing step was not ideal, as the artificial measurement error was not really growing linearly in time. However, it was the simplest possible procedure and it provided data that were more stationary and thus improved the adaptation process for the NARX model. In this case the FOR procedure yielded a model consisting of 54 monomials with maximum degree $p_{\max} = 3$, which was based on the following embedding

$$y_t = F_t \,, \quad \boldsymbol{x}_t = (P_t, P_{t-2}, \ldots, P_{t-36}, F_{t-2})^T \,. \tag{6.20}$$

In contrast to the previous model, this embedding includes much more information from the past, since the regressor vector contains 18 previous values of

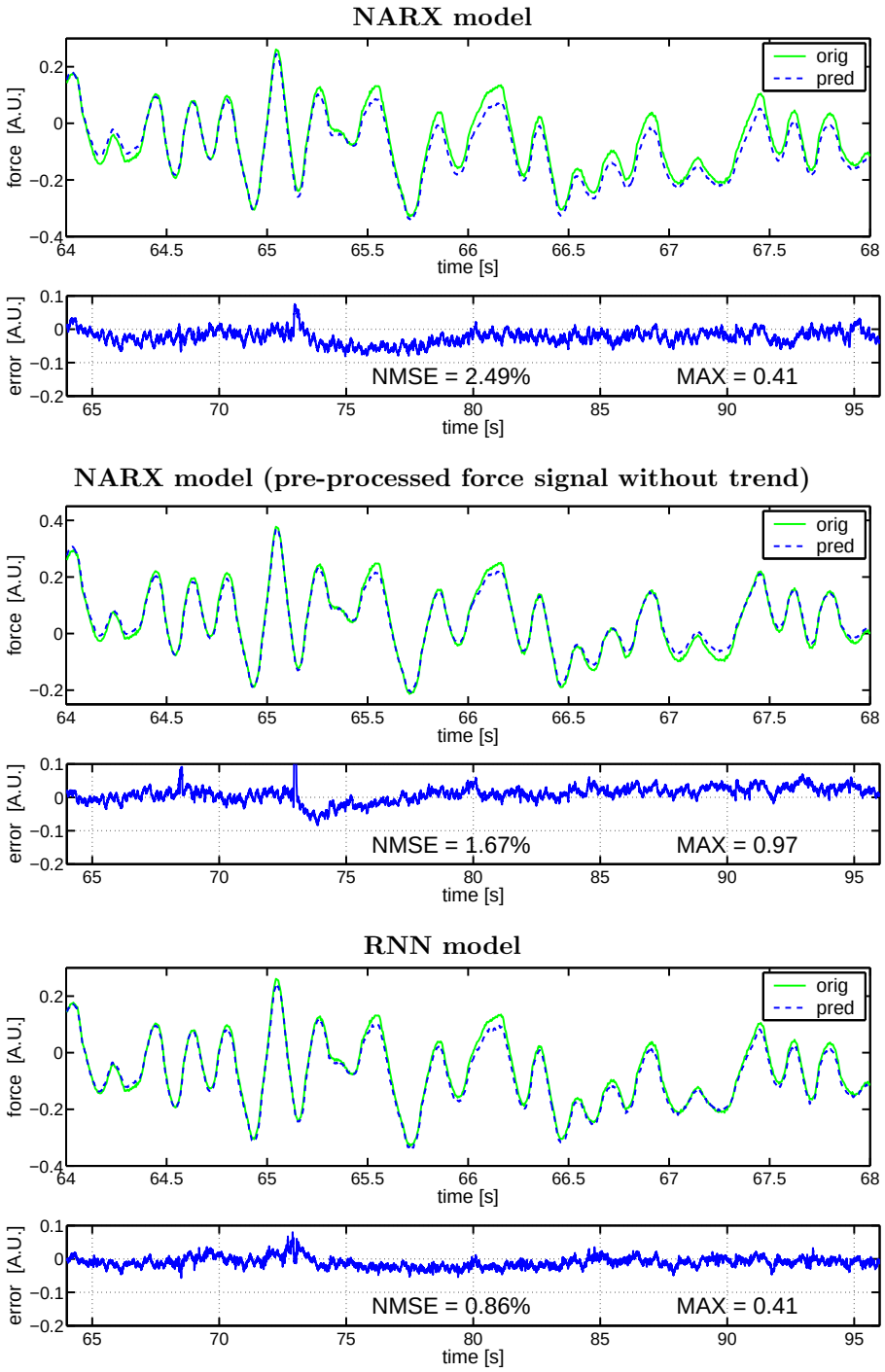

Figure 6.10: Results of the cross-prediction problem for the polynomial NARX and the RNN model. For each model a sample of the first 1000 predicted force values $\hat{F}_t$ and original values F_t of the test set are presented along with the errors $e_t = \hat{F}_t - F_t$ for the whole test set (note the different time spans). **Top:** The NARX model trained on the measured data shows the worst results. As can be seen, the errors are biased, because the model predictions are generally smaller than the original values. This is a result of the trend in the force signal. **Middle:** The predictions of the NARX model can be improved by removing the trend from the measured force signal. The bias of the NARX model that was trained and tested on such pre-processed data is considerably smaller. **Bottom:** The predictions of the RNN model are very good even though it was trained and tested on the original data with trend.

the position time series. Indeed, although the preprocessing was not optimal, the performance of the model could be improved leading to NMSE = 1.67% and MAX = 0.97. Again, this is an acceptable result although still not overly precise.

The third model we used was an RNN consisting of ten elements with the following internal dynamics

$$x_t^{(i)} = \alpha^{(i)} x_{t-1}^{(i)} + (1 - |\alpha^{(i)}|) \tanh(c_{i0} + \sum_{j=1}^{10} c_{ij} x_{t-1}^{(j)} + b^{(i)} P_t) \,, \quad i = 1, \ldots, 10 \,, \tag{6.21}$$

with the free parameters $\alpha^{(i)} \in [0, 1)$, $c_{ij} \in \mathbb{R}$, and $b^{(i)} \in \mathbb{R}$. This network was optimized on the training and validation set according to learning methods described in Section 5.3.4. As can be seen in Fig. 6.10 the RNN yields the best predictions even though it was trained on the original force signal with trend. With its performance of NMSE = 0.86% and MAX = 0.41 it seems to capture the characteristics of the friction system very well.

In [55] these three models were compared to other black-box models and to physics based models that were specifically designed to describe friction system. Although the NARX models showed acceptable results among the black-box models, they were no match for the more sophisticated physics based models. However, the RNN could well compete with the best of the physics based models, which had quality criteria of NMSE = 0.75% and MAX = 0.51 (see [55]).

As an aside, the best prediction results could be achieved by combining the two best physics based models and the RNN black-box model in a so called *ensemble model*. The improved criteria in this case were NMSE = 0.28% and MAX = 0.24. These remarkable values indicate that physics based models and black-box models are not necessarily mutually exclusive but on the contrary may benefit from each other.

6.3 Modeling of pre-sliding and sliding friction

The modeling of pre-sliding friction combined with sliding friction is a great challenge for most of the modeling approaches. Both regimes are fundamentally different from each other concerning the behavior of the system. In contrast to pre-sliding, the friction in the sliding regime is governed by velocity. Although, for itself, the sliding regime is not more difficult to describe than the pre-sliding regime, the difficulty lies in the combination of both. Whenever the system switches from pre-sliding to sliding, genuinely new characteristics have to be modeled.

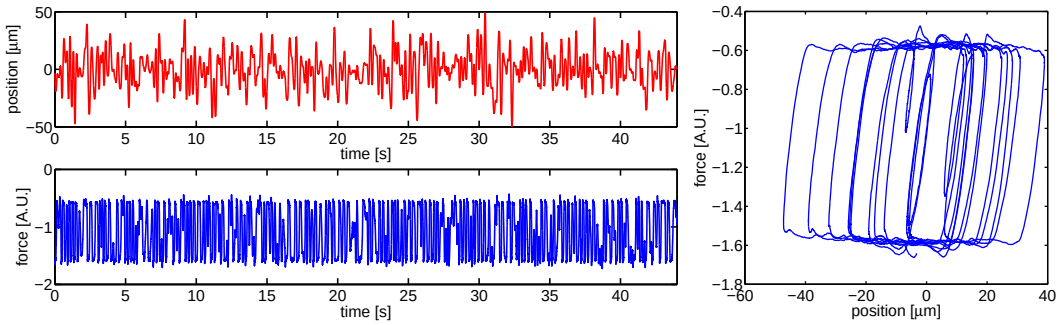

Figure 6.11: Experimentally obtained friction data. Position and friction force measured for a time period $T = 44\,\mathrm{s}$ with a sampling rate $f_s = 2500\,\mathrm{Hz}$. **Left:** Position and friction force plotted against time. **Right:** Friction plotted against position. Compared to the pre-sliding case the hysteresis curves are much broader.

6.3.1 Training and testing

The friction data from the sliding regime were measured by the group of KU Leuven with the same setup as was used before for the pre-sliding friction (see Section 6.2.1 and Fig. 6.7). In order to initiate sliding the peak-to-peak value of the reference trajectory was chosen to be higher than in the pre-sliding case. As a result the friction block would slide for greater displacements in the same direction and stick at the reversal points of the trajectory, yielding a position and a force signal that included sliding as well as pre-sliding characteristics (see Fig. 6.11).

Comparing the right hand side of Fig. 6.11 with that of Fig. 6.8, one can see the broadening of the hysteresis curves, which is characteristic for the gross-sliding regime. It is evoked by the occurrence of sliding behavior. Here, where the friction block starts to glide on the surface beneath it, friction is no longer dominated by displacement. Instead, the velocity becomes the critical variable. Plotted against time, the sliding regions are discernible as 'plateaus', where the force seems almost constant over long periods of time (see Fig. 6.12). However, zooming into such a region (see rhs of Fig. 6.12) reveals that the force is by no means constant. It oscillates rather on a much finer scale. These oscillation are explained by 'stick-slip behavior'[3] of the friction-block.

The fast oscillations stemming from stick-slip motion are the reason why the sampling frequency had to be increased to $f_s = 2500\,\mathrm{Hz}$ in comparison to the pre-sliding case, where it was $f_s = 250\,\mathrm{Hz}$. This is explained in Fig. 6.12. Similar to the pre-sliding case the force signal follows coarsely the oscillations of the reference position signal. Since the reference position signal is chosen

[3] Here, the term stick-slip behavior does not refer to the classical usage, which describes a movement, where an object alternates between standing still and moving. Instead, stick-slip behavior is used for a movement with an alternating slow and fast velocity. Note that the parentheses will be omitted for the rest of the section.

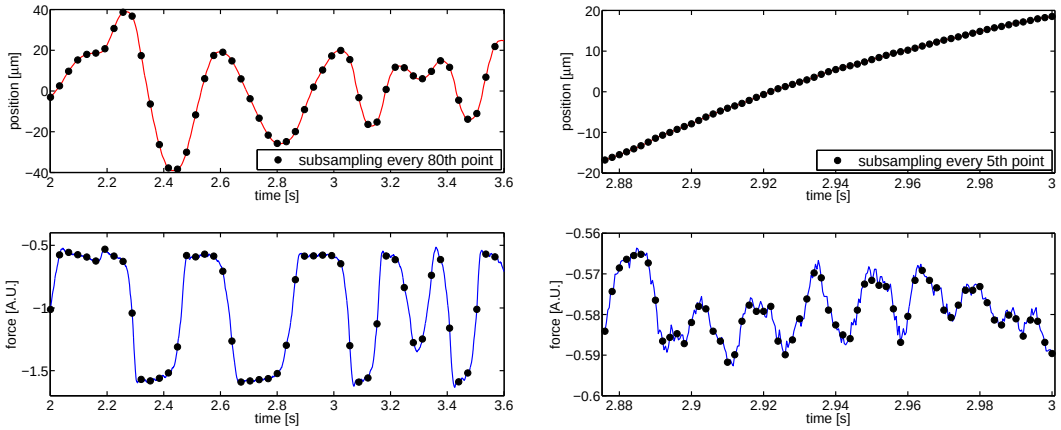

Figure 6.12: The measured force signal poses considerable difficulties for modeling because of its oscillations on very distinct scales. **Left:** The low frequency component in the force signal follows the oscillations in the reference position signal. It is well captured by a heavily subsampled version. For demonstration every 80th point is marked by a filled circle. **Right:** Zooming into a sliding regime reveals a finer scale in the force signal. These faster components do not result from fluctuations in the reference position signal and thus are not predefined by the user. Rather they correspond to stick-slip behavior of the friction system. Information in the higher frequencies is only preserved for higher sampling frequencies. For demonstration every 5th data point is marked by a filled circle. Note that the fast oscillations are also present in the measured position signal. However, they are not visible in this figure due to their very small amplitude.

to have only low frequencies ($f \sim 4\,\mathrm{Hz}$), the response signal from the friction system (namely the friction force) alternates also very slowly. Information about these oscillations is preserved even if a lower sampling frequency (see lhs of Fig. 6.12) is used. However, in contrast to pre-sliding, stick-slip behavior occurs in the sliding case. During such periods the system does not follow exactly the reference position. Due to sticking the position of the system sometimes lags behind the reference trajectory until a slipping phase occurs. This alternating behavior is characterized by low amplitude oscillations in the measured position and the force signal (see rhs of Fig. 6.12). The stick-slip oscillations are much faster than the oscillations of the reference position signal ($f \sim 60\,\mathrm{Hz}$). Therefore, information about the sliding phase is only preserved if the sampling frequency is chosen higher than in the pre-sliding case.

With a sampling frequency $f_s = 2500\,\mathrm{Hz}$ the number of data points was much higher than in the pre-sliding case, although a shorter measurement period $T = 44\,\mathrm{s}$ was used. Each time series (position and force) consisted of $N = 110000$ samples. From these time series the first 90000 data points were used for training and validation, while the last 20000 data points were taken for testing. As before for the pre-sliding case, the modeling task was a cross-prediction. The force value F_t at a time step $t \in \mathbb{N}$ had to be predicted from past and present values of the position time series $P_t, P_{t-1}, \ldots$

For quality evaluation the NMSE and MAX criteria were based on free-running predictions. Again, the free-running predictions were started at the very first force value F_1 and followed through until the last value F_{110000}. The NMSE was computed as

$$\text{NMSE} = \frac{100\%}{20000 \cdot \sigma_F^2} \sum_{t=90001}^{110000} (F_t - \hat{F}_t)^2 \,, \tag{6.22}$$

and MAX as

$$\text{MAX} = \frac{1}{\sigma_F} \max \left\{ |F_t - \hat{F}_t| \mid t = 90001, \ldots, 110000 \right\} . \tag{6.23}$$

6.3.2 Results

The multi-scaled force signal, as mentioned in the previous section, had a great impact on the strategies and the results of the static and dynamic modeling approach. Both approaches had to be adapted in specific ways in order to achieve good prediction results, and it is worthwhile to address the different strategies of the two approaches separately.

NARX models

Since the force signal included information on different scales, it was not trivial to determine an appropriate embedding for the static modeling approach. The usual rule of thumb is to choose the delay about one forth of the dominant period of oscillation in the signal. However, here we had two very distinct frequencies: fast oscillations in the sliding regime and slow oscillations following the course of the reference trajectory. With a try-and-error approach the following embedding was found to yield acceptable results

$$\begin{aligned} y_t = F_t \,, \quad \boldsymbol{x}_t \;=\; & (F_{t-1}, P_t, P_{t-2}, P_{t-4}, P_{t-6}, \\ & P_{t-80}, P_{t-82}, P_{t-84}, P_{t-86} \\ & P_{t-160}, P_{t-162}, P_{t-164}, P_{t-166})^T \,. \end{aligned} \tag{6.24}$$

The regressor had 13 components and two different delays, which reflected the individual scalings in the signal. The short delay $\tau_1 = 2$ stored the information about the fast oscillation, while the long delay $\tau_2 = 80$ contained information about the slow oscillations.

The FOR procedure yielded a polynomial NARX consisting of 57 monomials with maximum degree $p_{\max} = 3$. The one-step prediction error of this model was excellent. However, the real testing was supposed to be performed for free-running predictions. Unfortunately, in this mode the NARX model turned out to be unstable. Starting with the first point in the training set, the prediction performance was acceptable until time step $T \approx 31.1\,\text{s}$ (see

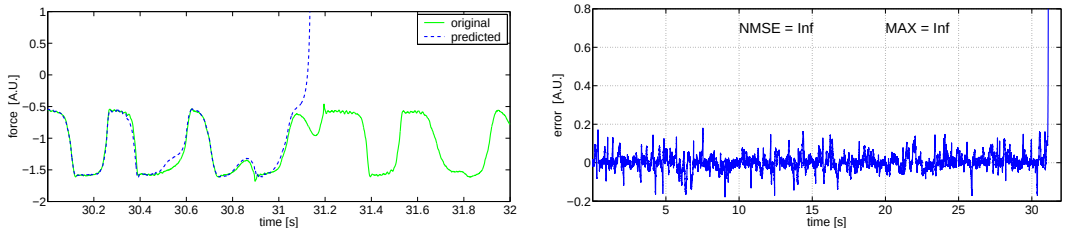

Figure 6.13: After training the NARX model, it was tested in the free-running mode. Unfortunately, it revealed itself as being unstable. From the time $T \approx 31.1\,\mathrm{s}$ the prediction errors grew unboundedly. Because of the instability the NMSE and the MAX criterion could not be evaluated on the test set, which starts not until $T = 36\,\mathrm{s}$.

Fig. 6.13). From here on the prediction errors grew unboundedly rendering the evaluation on the test set (starting from $T = 36\,\mathrm{s}$) impossible. Clearly, this NARX model could not be used without a stabilization strategy.

There are many ways to stabilize polynomial NARX models, and we tested three of them. The simplest way to stabilize a model is by bounding its output. Instead of the genuine model output $\hat{F}_t$ a transformed version $\hat{\hat{F}}_t = h(\hat{F}_t)$ with a bounding function $h : \mathbb{R} \to \mathbb{R}$ is used for further processing. For example, a simple bounding function is

$$h(x) = \begin{cases} U & \text{if } x > U \\ x & \text{if } L \leq x \leq U\,, \\ L & \text{if } x < L \end{cases} \tag{6.25}$$

with an upper bound $U \in \mathbb{R}$ and a lower bound $L \in \mathbb{R}$. This strategy is fast to apply because it does not involve any retraining of the model. However, the lack of retraining is at the same time a weakness of the strategy as it might introduce artificial behavior of the model that is not sanctioned by the information in the training data.

In our case, the bounding strategy was effective in stabilizing the model. Fig. 6.14 shows the results of for two different sets of bounds. In both cases the model output was left unbounded from below with $L_1 = L_2 = -\infty$. In the first case the upper bound was set to be marginally over the maximum force value in the training set: $U_1 = -0.42$. In the second case we were more generous with $U_1 = 1.0$. As can be seen, stabilization is successfully established in both cases. After the instability region at $T \approx 31.1\,\mathrm{s}$ is overcome, the predictions regain their former precision. Another result is that the prediction quality on the test set is not influenced by the choice of the upper bound. For both cases the criteria are NMSE $= 1.28\%$ and MAX $= 0.53$.

Another strategy for stabilizing polynomial NARX models is ridge regression (see for example [30]). It leads to a reduction of the model complexity

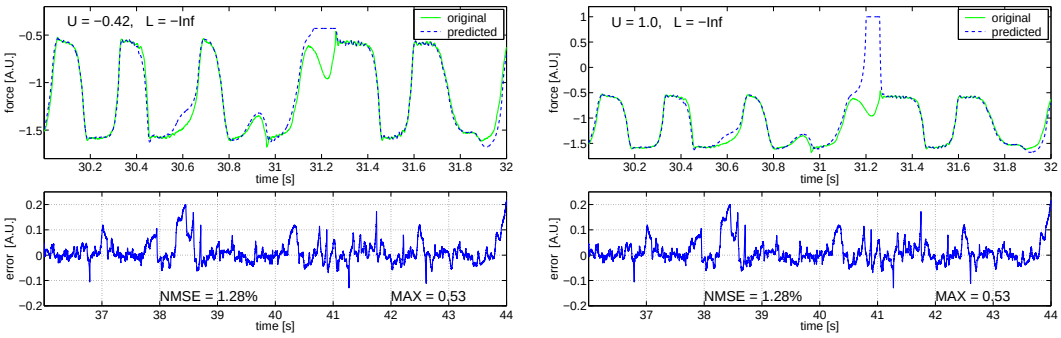

Figure 6.14: Results of the bounding strategy for the free-running mode of the NARX models with two different upper bounds. Note, in both cases the model outputs are not bounded from below ($L = -\infty$). **Left:** The upper bound is chosen as $U = -0.42$, which is a value slightly above the maximum value of the force signal in the training set. Bounding the model output overcomes the instability at $T \approx 31.1\,\mathrm{s}$. After that the model finds its way back to its previous accuracy. **Right:** The upper bound is chosen very generously as $U = 1.0$. However, even with this high value stability can be established. After some time the free-running equal the ones in the previous case. Thus, the NMSE and the MAX criterion on the test set are not influenced by the choice of the upper bound.

by shrinking the individual weights of the basis functions. The shrinkage is thereby controlled through a balancing factor $k \in \mathbb{R}$. For higher values of k the procedure leads ideally to simpler models, which also tend to be more stable. The tuning of the balancing parameter can be done with the help of the validation set. Ridge regression is slower than the bounding strategy because it requires a retraining of the model.

The results for two different values of the balancing factor k are shown in Fig. 6.15. The first value $k = 0.05$ was optimized on the validation set. It strikes a good balance between model simplicity and model accuracy. One can see that the resulting model outputs have considerable deviations from the true values at the former instability region $T \approx 31.1\,\mathrm{s}$. However, the model stays stable, and the precision criteria on the test set are NMSE = 1.38% and MAX = 0.62.

For demonstration purposes, ridge regression was repeated with the higher balancing factor $k = 0.2$. Higher factors favor simpler models, which are generally also more stable. However, the constraint in the complexity usually makes the models more rigid, leading to lower model accuracy. These features are nicely visible on the right hand side of Fig. 6.15. The new model copes better with the instability at $T \approx 31.1\,\mathrm{s}$ but the price paid are higher errors on the test set. The quality criteria NMSE = 2.28% and MAX = 0.86 are considerably higher than before.

A third possibility to stabilize polynomial NARX models is the optimization of the parameters based on free-running predictions. This is maybe the best but also the most time-consuming procedure. Similar to ridge regression

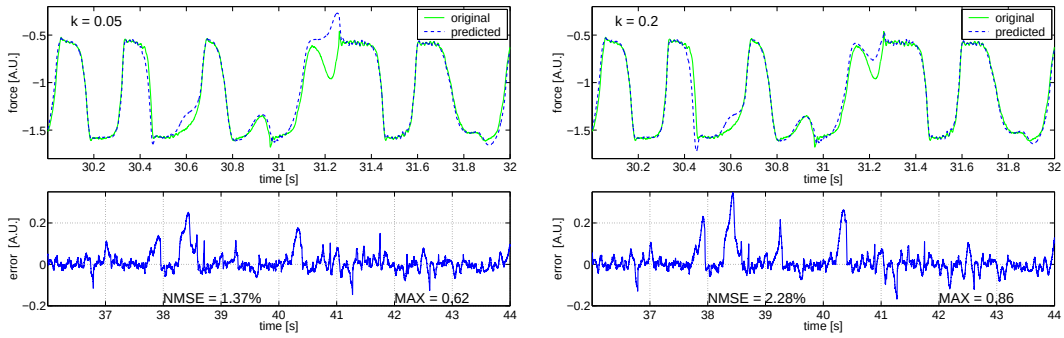

Figure 6.15: Results of the ridge regression strategy for two different values of the balancing factor k. **Left:** The first factor $k = 0.05$ was optimized on the validation set and strikes a good balance between model complexity and model accuracy. As expected, the model errors on the test set are low. **Right:** By raising the factor to $k = 0.2$ the complexity of the model is further constrained. This leads to a model that shows an even more stable behavior in the instability region $T \approx 31.1\,\mathrm{s}$. However, the accuracy suffers under these constraints as can be seen by the higher errors on the test set.

the model has to be retrained. However, unlike ridge regression the minimization problem in the free-running mode of the model is in general not convex. Nonlinear optimization has to be applied, which is computationally intensive.

In our case, the optimization of the free-running model could not be done directly because of the instability at $T \approx 31.1\,\mathrm{s}$. Instead, the optimization was performed in combination with error-propagation. In this method the output of the model is fed back to the model in a pre-processed version. Instead of using the pure model output it is mixed with the original time series and the model is trained with this modified feedback. The mixture is a weighted average of model output and original time series. By putting more weight to latter, instabilities of the model can be remedied. After one training phase is finished a new weighting is applied in the mixture and a new training phase is started. Iteratively the main weight of the mixture is shifted from the original time series to the model output. In the last iteration only the model output is used.

The results of this procedure are shown in Fig. 6.16. As can be seen, the instability at $T \approx 31.1\,\mathrm{s}$ is overcome easily by the retrained NARX model. In fact, the errors in this region for the free-running approach are the lowest of all our stabilization efforts. Good results were also achieved concerning the model accuracy on the test set where the NARX model performs with NMSE $= 1.11\%$ and MAX $= 0.49$.

As a last option we tried to optimize the embedding step of the static modeling approach. Instead of simple try-and-error a new procedure based on a genetic algorithm was tested. The new embedding was optimized for local models, where it showed very good results. It had the following targets and

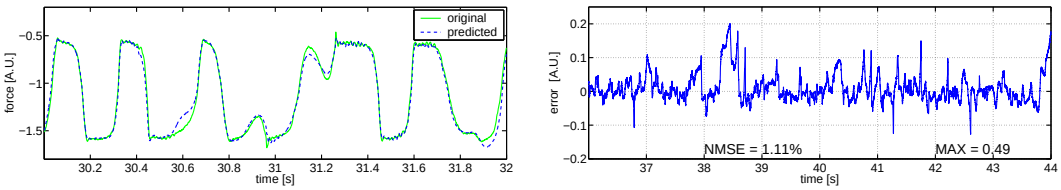

Figure 6.16: Results gained by optimizing the model performance in the free-running mode. The former instability of the model at $T \approx 31.1\,\mathrm{s}$ has almost vanished. The performance on the test set with very low error amplitudes is remarkable.

regressors

$$y_t = F_t\,, \quad \boldsymbol{x}_t = (F_{t-19}, P_t, P_{t-16}, P_{t-66}, P_{t-67})\,. \tag{6.26}$$

This embedding was used as a basis for a new polynomial NARX model. The FOR procedure produced a model consisting of 84 monomials with maximum degree $p_{\max} = 5$. Following, its weights were optimized in the free-running mode because this procedure showed already good results for the previous NARX model. In Fig. 6.17 the results of the optimization procedure are shown. With the new embedding the performance on the test set could be significantly improved leading to NMSE = 0.44% and MAX = 0.37. This demonstrates how important the choice of the right embedding is for a static modeling approach.

RNN models

The multi-scaled force signal complicates things in the dynamical modeling approach as well. In order to reproduce the fast oscillations a finely sampled version of the position signal has to be provided as an input to the RNN. However, for the slow oscillations information from a long time in the past is needed. In order to have such information present at the actual time step, the RNN has to store informations for a long period of time. This leads to a dilemma. Since the memory of RNNs is not lossless and the memory capacity is constrained by the number of elements, a very large network is required. On the other hand, such a large network hampers optimization attempts by being too slow and unwieldy.

In order to avoid problems concerning overly large networks and to save computation time another solution was found. An external memory in form of delay lines was used, which relieved the internal memory of the RNN. On the left hand side of Fig. 6.18 the concept of these delay lines is schematically depicted. The position signal P_t which serves as an input to the RNN is split into three different signals. The first signal is the original position signal. The second signal is a delayed version of the position signal with a delay equal to $\tau = 80$ samples. And similarly the third signal has a delay $\tau = 160$ samples. These three signals are presented to the RNN as a new multi-variate input

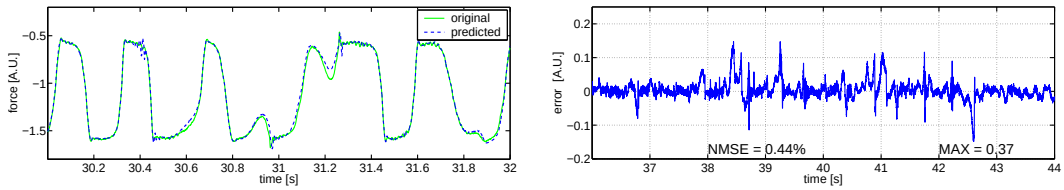

Figure 6.17: Prediction performance of polynomial NARX model, which is based on an optimized embedding. The former instability region at $T \approx 31.1\,\mathrm{s}$ has effectively vanished, and the prediction performance on the test set could be improved, as can be seen by the remarkably good model accuracy.

signal. Thus, the RNN can resort to past values without having to store them internally.

This strategy seemed to be very effective in our case. On the right hand side of Fig. 6.18 are shown the results of the dynamical modeling approach. The optimized network consisted of 45 elements with the internal dynamics

$$x_t^{(i)} = \alpha^{(i)} x_{t-1}^{(i)} + \tanh(c_{i0} + \sum_{j=1}^{45} c_{ij} x_{t-1}^{(j)} + b^{(i)} P_t)\,, \quad i = 1, \ldots, 45\,. \tag{6.27}$$

The connectivity of this network was about 11% translating to 225 internal connections. The optimization procedure was based on Simulated Annealing as described in Section 5.3.4. In Fig. 6.18 one can see that the instability region of the polynomial NARX models at $T \approx 31.1\,\mathrm{s}$ was no problem for the dynamical modeling approach. This is obviously resulting from the optimization procedure of the RNN, which is automatically based on free-running predictions and cannot produce unstable results. Furthermore, the performance concerning the precision of predictions on the test set is by far better than that of the NARX models. Both criteria, NMSE = 0.13% and MAX = 0.2, have very low values indicating a very precise model.

In [70] the NARX model and the RNN were compared to other black-box models and to gray-box models, with the latter including knowledge about some aspects of the physics behind friction. Similiar as for pre-sliding friction the dynamic modeling approach could show its strength in producing reliable and stable models also for the sliding regime of friction. The RNN was the best model from all black-box and gray-box models. The next best model was a gray-box model with the quality criteria of NMSE = 0.29% and MAX = 0.90 (see [70]). By combining the RNN with other models in an ensemble model the performance could be slightly improved.

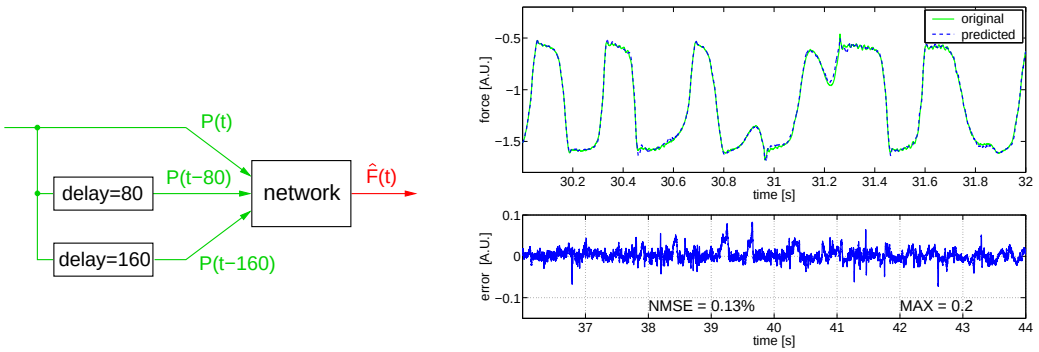

Figure 6.18: Left: For capturing slow oscillations the RNN has to have access to information in the input signal P_t that lies long time in the past. This can be achieved by creating an external memory in form of delay lines. The signal is stored in these lines for a specific period of time (80 samples or 160 samples) and then presented to the network as an extra input. **Right:** The instability region of the NARX models at $T \approx 31.1\,\mathrm{s}$ is not present for the multi-variate RNN. Also the performance on the test set is better than that of the NARX models.

6.4 Control

Models of friction are usually meant to be applied in control tasks. Even though common sense suggests that accurate models probably show better control results than inaccurate ones, this implication does not follow automatically. It is not clear whether models with good prediction performances will indeed have a positive impact on the controllability of a system. Accuracy comes often at the cost of complexity, which could prove a hinderance in control applications. In this section we take a closer look at the usage of dynamical networks for the tracking problem.

6.4.1 Tracking problem

The tracking problem is one of the most important control tasks. The objective is to develop a controller that forces the behavior of a system to follow a desired course. Formally, the dynamics of a discrete system can be described by the state equation

$$\boldsymbol{x}(t) = f(\boldsymbol{x}(t-1), u(t))\,, \tag{6.28}$$

with an inner state variable $\boldsymbol{x}(t) \in \mathbb{R}^{d_x}$, a transition function $f : \mathbb{R}^{d_x} \times \mathbb{R} \to \mathbb{R}^{d_x}$, and an input signal $u(t)$. The transition function defines deterministically how the system changes its states for certain inputs. If the inner state is accessible, the tracking problem reduces to the identification of the transition function $f(\cdot)$. With the knowledge of the transition function the future states $\boldsymbol{x}(t)$ can be forced to follow a desired, reference trajectory $\boldsymbol{x}_r(t)$ by generating an appropriate input signal $u(t)$.

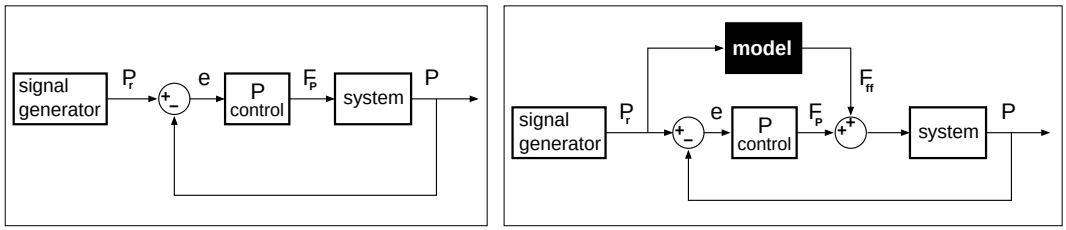

Figure 6.19: Left: Closed loop control by a P-controller. The P-controller forces the system to follow a reference trajectory $P_r(t)$ created by a signal generator. For this purpose it applies a control signal $F_P(t)$, which is proportional to the deviations $e(t) = P_r(t) - P(t)$ between the actual position $P(t)$ and the reference position. **Right:** The closed loop control is extended by an additional feed-forward controller. The model uses the information from the desired trajectory to produce an additional control signal $F_{ff}(t)$, which assists the closed loop controller.

Unfortunately, the inner state variable is seldom available to the user in real world applications. Typically the access from the outside is limited to an observable variable $\boldsymbol{y}(t) \in \mathbb{R}^{d_y}$, which is a transformation of the original system state

$$\boldsymbol{y}(t) = h(\boldsymbol{x}(t))\,, \tag{6.29}$$

with the transformation function $h : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$. We call the observable variable also the output or response of the system. Without the explicit knowledge of the inner states, the system response is the only means by which the system's behavior can be determined and controlled. The tracking problem is to find an appropriate controller that forces the output signal of the system to follow (or track) a reference signal $\boldsymbol{y}_r(t)$, which is specified by the user. The control is regarded successful if the output follows exactly the reference. The distinction between the inner state and the observable variable is not necessary if the functional relationship between them is one-to-one. However, this case is rare because most of the time only partial information about the system state can be gained by measurements on real physical systems. The incomplete knowledge of the system state is often what makes tracking problems complicated.

In our numerical simulation we simulated the friction setup from Fig. 6.7. The input to the system was the force signal, $u(t) = F(t)$, its output the position variable, $y(t) = P(t)$, and the reference signal the desired position, $y_r(t) = P_r(t)$. This setup is very general and one can find many examples for it, e.g. a robot-arm following a desired trajectory or the positioning of read head in an optical storage device under the influence of friction.

As a basis for the simulation a closed-loop control scheme as depicted on the left hand side of Fig. 6.19 was used. In this scheme the actual position of the system, $P(t)$, is measured and compared to the reference trajectory $P_r(t)$. The

deviations or *tracking errors* $e(t) = P_r(t) - P(t)$ are used by a P-controller[4] as a feedback to produce a control signal $F_P(t)$. The control signal is proportional to the deviations and affects the system in such a way as to minimize the tracking errors. For linear systems this control scheme is nearly perfect, rendering an error signal $e(t)$ that is almost vanishing. However, a system influenced by friction is nonlinear in its nature and for such systems the P-controller displays weaknesses. The deviations of the system trajectory from the reference signal become significant in their amplitude, leaving room for improvement. In this case one should consider applying controllers, specifically designed to handle nonlinearities.

As a simple solution we used a nonlinear model (RNN) as an additional feed-forward controller (see rhs Fig. 6.19). The model made use of the information from the reference position to produce an appropriate force signal $F_{ff}(t)$. Ideally, the additional force signal was to minimize the tracking errors, leading eventually to a control region that could be considered linear and where the P-controller could be employed in a more effective way. The improvement was measured by the reduction in the amplitude of the error signal $e(t)$ (see next section).

Since the setup was numerically simulated, we had to use a model for the friction system in Fig. 6.19. This led to a somewhat redundant situation of developing a model of a model. Selecting both models from the same model class, e.g. RNNs, in this situation would normally yield perfect control results. These results, however, would only be of limited value. Instead, we tried to minimize the redundancy by choosing very distinct model classes. Specifically, the applied friction model was required to have only little in common with RNNs and to include as many aspects of real friction systems as possible. These requirements were nicely fulfilled by the generic friction model (GF model) of the Leuven group (see Section 6.1), which is among the most sophisticated among the physics based models. With its statistic approach to modeling friction on the microscopic level with a huge population of elements it is sufficiently different from RNNs and poses with its random nature a great challenge to control attempts. As a pre-step to modeling and controlling the complex generic model we also used the Generalized Maxwell-slip model (GMS model) of the Leuven group (see Section 6.1), the latter model having the advantage of being more simplistic and thus better suited for exhaustive numerical simulations.

Summarizing we can state that in the light of the mentioned redundancy it is problematic to transfer results from numerical simulations of control experiments to real world applications. However, with the precaution steps we followed, the numerical simulations should at least yield some qualitative facts

[4] A P-controller is the simplest possible feedback controller. It produces a control signal which is proportional to the deviations between system position and reference trajectory. Usually, it is combined with an integral and a derivative part in PID control.

about the control ability of dynamical black-box models. Especially the usage of a RNN as control model and the generic friction model as friction system preserves realistic aspects and is a legitimate pre-step for real world control experiments.

One valid question still remains: Why not use the generic friction model from Leuven for the control itself if it is so sophisticated? The short answer is that such an approach usually has immense disadvantages. Even though a model may be solidly grounded in physics and show good agreements to real physical systems, it does not automatically mean that it is easy to apply to a concrete modeling task. As already mentioned in Section 6.1, there is a difference between physics based models, which should show as much resemblance to real systems as possible, and simple black-box models, which are trained to show good approximations only for specific situations. Often, complex models have dependencies on parameters that are notoriously difficult to determine. For example, in the case of friction many parameters depend on material properties or environmental conditions that are unknown or eventually change in time. In such a case it is often advantageous to adjust a black-box model to the current situation (or even use self-adjusting ones) rather than to reestimate all parameters of a sophisticated model. In the end it condenses to the question: What is the final goal of the modeling task? If it is a deeper understanding of friction, physics based models should be used. If it is a quick solution to a tracking problem, black-box models may be a better choice.

6.4.2 Simulation setup

The simulation was done in Simulink (Version 5.0 (R13)) with the setup following the schematics in Fig. 6.20. The basis for this schematics was already explained in Fig. 6.19. In its core it consists of a closed-loop control (P-controller) complemented by feed-forward control through a nonlinear model. For the simulation a fixed-step ODE solver (ode5, Dormand-Prince) was chosen with a time step $t = 0.001$. The subsystems, simulating the friction and the signal generator, were developed by the group of KU Leuven, who were also responsible for the experimental part of the project. The friction systems are based on their GMS and GF model (see Section 6.1).

Following from the experiences made in context with the modeling of sliding friction data (see Section 6.3) we employed an external memory for the dynamical model. The input to the model subsystem, namely the reference trajectory $P_r(t)$, was split into three individual versions, $P_r(t-2\tau)$, $P_r(t-\tau)$, and $P_r(t)$ with a fixed delay τ (see Fig. 6.20). This multivariate input signal was then provided to the dynamical model (RNN), which produced a force signal $F_{ff}(t)$. The force signal from the nonlinear model was used as an extra feed-forward control signal in addition to the signal from the P-controller. The internal parameter of the P-controller was held fixed for all simulations

at $K_p = 4000$. The feed-forward control could be switched on and off with the help of a time trigger during a simulation run.

In separate simulation runs two different friction models were used for the friction subsystem. In both cases the inertial forces were computed for a friction block with a mass $m = 5$.

Friction system 1 included the GMS model for simulating the dry friction force. The GMS model consisted of 10 elements with individual parameters. Additionally a viscous part depending on the velocity was simulated with the parameter $\sigma_2 = 0.2$. To include random aspects of friction, which cannot be captured by models, an internal noise source with uniform random numbers $r_i \sim \mathrm{U}\,[-0.05,\, 0.05]$ (see Fig. 6.20).

Friction system 2 included only the physics based GF model in addition to the inertial part (see Fig. 6.20). Since the GF model is inherently random, the noise source could be omitted here. The GF model consisted of 1000 elements with individual parameters. In this case the viscous part was left out.

6.4.3 Training

Before dealing directly with the control problem, the nonlinear model was adapted to the system behavior in the training phase. For this purpose we had to acquire sample data from the system, which could be used for offline learning. Although it may seem simple, the acquisition of data can be done in many ways and the choice between them is often critical for the success of the modeling task. Probably the most important requirement is that the training data should capture the system behavior in all main operation regimes that could also occur during its later application in the control setup. This is especially important for black-box modeling because here the training data represent the only source of information about the modeled system.

First, we tried a straightforward approach to creating training data as shown on the left hand side of Fig. 6.21. With a signal generator a random reference trajectory $P_r(t)$ was produced. From this trajectory the inertial force $F(t)$ was computed that is necessary to move a mass block with $m = 5$. The inertial force was applied to the friction system as input. In this situation a system without friction would follow exactly the reference trajectory. However, due to internal friction forces the real system position $P(t)$ deviates from the trajectory. In our case the deviations rendered the computed input and output data useless for training because of the instationarity of the position signal (see rhs of Fig. 6.21). The amplitude of the position drifted in regions that were not typical for the later application in the control environment. Note that in this example we used friction system 1 (GMS model) for simulating the friction block. However, the same drifting happens also for friction system 2 (GF model).

In order to gain usable training data, an alternative approach was chosen.

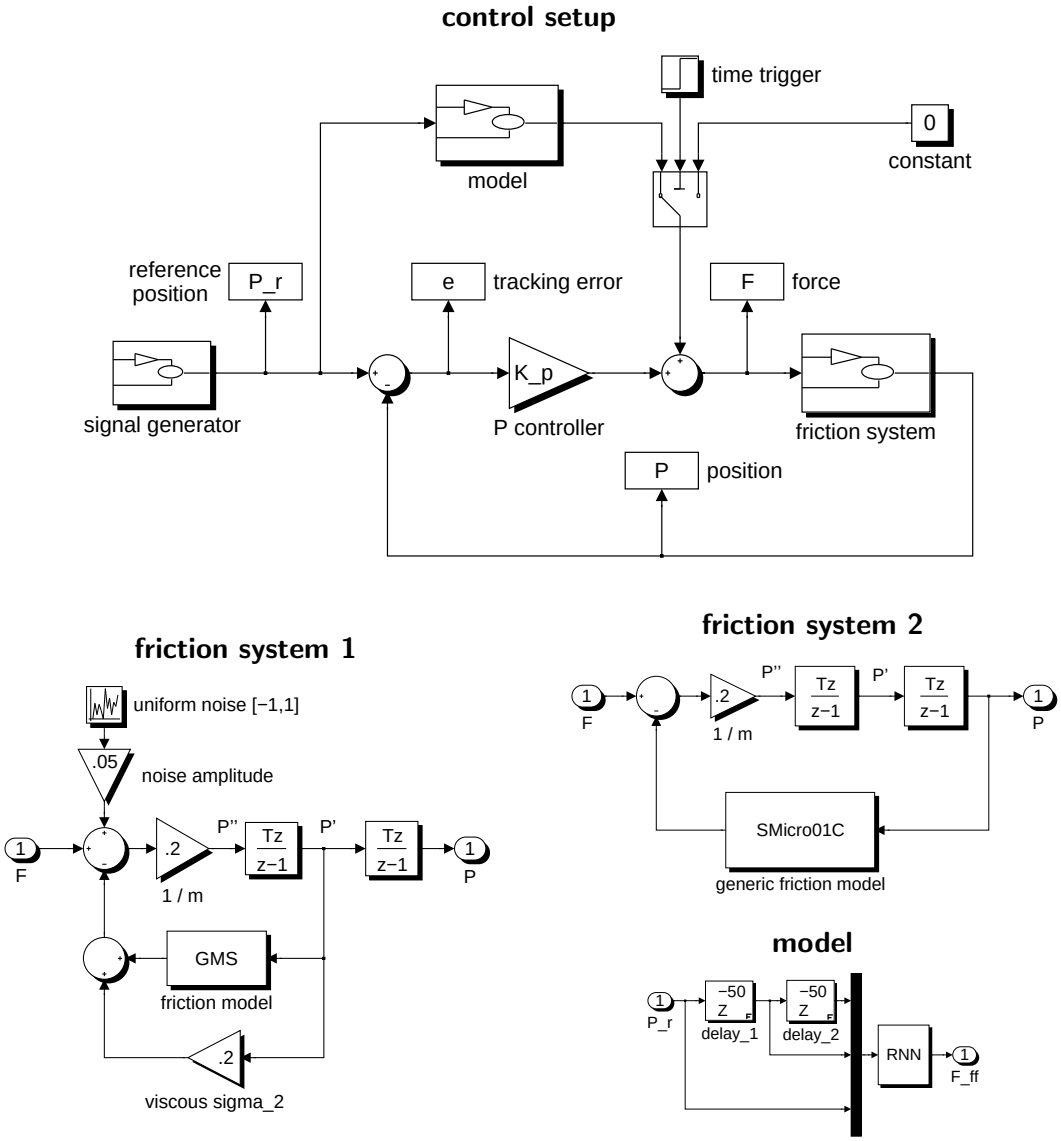

Figure 6.20: The **control setup** in Simulink follows the scheme presented in Fig. 6.19. Additionally a time trigger is included, which activates the feed-forward control during a test run. During the numerical simulations two different friction models are employed for the friction system. Both models are provided with the control signal $F(t)$ (force) as input and export their actual position $P(t)$ as output. **Friction system 1** consists of a frictional part, which is simulated by the GMS model. Additionally, a viscous friction is implemented, which is proportional to the velocity of the system. An internal noise source simulates the unpredictable aspects of friction. **Friction system 2** consists only of the frictional part that is simulated by the GF model. The GF model is much more complex than the GMS model. Since it is inherently random, the random noise can be omitted here. In contrast to friction system 1 viscous friction is not simulated. **Subsystem model:** Before the reference position $P_r(t)$ is provided as an input to the model, it is split into three delayed versions. The RNN is then driven with the new multivariate signal $\hat{\boldsymbol{P}}(t) = (P_r(t-2\tau), P_r(t-\tau), P_r(t))$ and produces the control signal $F_{ff}(t)$ as an output (here: $\tau = 50$ samples).

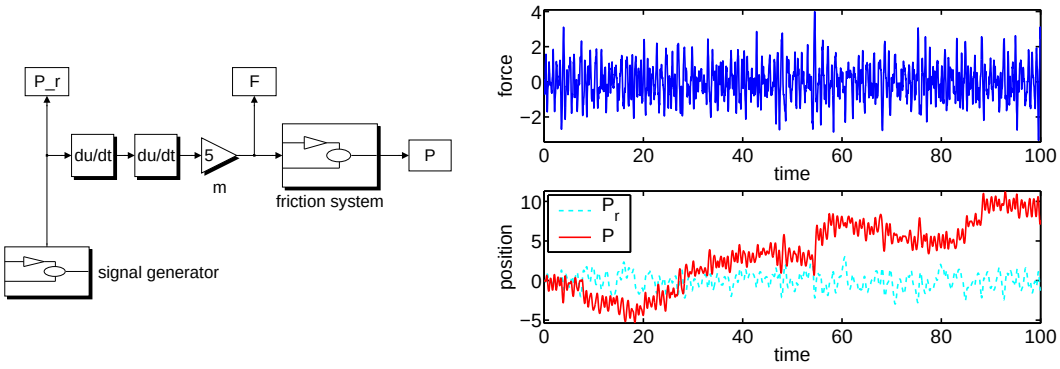

Figure 6.21: Direct approach to creating training data leads to instationary data. Simulation was run with friction system 1 (GMS model). **Left:** A random reference trajectory $P_r(t)$ is produced by the signal generator. The inertial force $F(t)$ is computed from this trajectory and provided as input to the system. Because the system is under the influence of friction, the real position $P(t)$ deviates from the reference position. **Right:** The computed force $F(t)$ and position signal $P(t)$ are not suited for training because the position signal is highly instationary. The deviations to the reference trajectory $P_r(t)$ are not typical for the following control application.

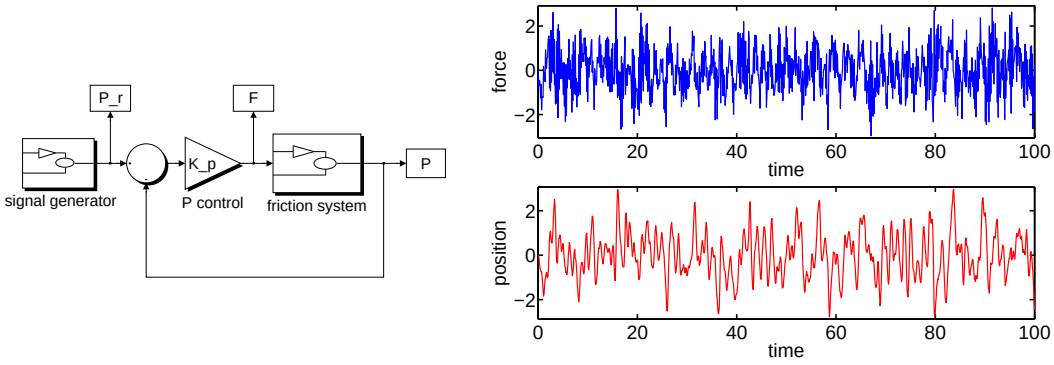

Figure 6.22: Indirect approach to creating training in the scope of closed-loop control Simulation was run with friction system 1 (GMS model). **Left:** A random reference trajectory $P_r(t)$ is produced by the signal generator. A P-controller ($K_p = 4000$) is used to force the friction system to follow this trajectory. The control signal $F(t)$ (force) and the actual position $P(t)$ are recorded and used later for training. **Right:** The computed force $F(t)$ and position signal $P(t)$ are well suited for training. In contrast to the direct approach, the position signal stays in the operation region that is also relevant later for control. Note, the reference trajectory was not plotted because in chosen resolution it coincides with $P(t)$.

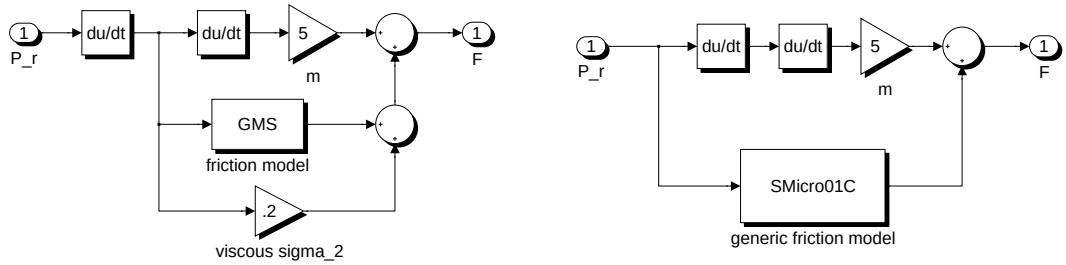

Figure 6.23: Left: Perfect model $M^{(1)}_{perf}$ for friction system 1 (see Fig. 6.20). Additionally to the inertial force the friction forces (GMS dry friction and viscous friction) are accounted for. **Right:** Perfect model $M^{(2)}_{perf}$ for friction system 2 (see Fig. 6.20). The dry friction force is accounted for by the GF model.

Instead of directly applying a force to the friction system, the training data was generated in the scope of a closed-loop control scheme (see Fig. 6.22). The same signal generator that was used for the former direct approach produced a reference trajectory $P_r(t)$. However, this time a P-controller with $K_p = 4000$ forced the friction system to follow this trajectory. Thus, the recorded position $P(t)$ remained in an operating region that was relevant for the following control application. With this approach a training data set, consisting of the force and the position time series each with 100000 data points, was generated.

6.4.4 Testing

After training the dynamical models (RNNs) for the control application, their quality was evaluated with two different criteria in order to discriminate between their static and their dynamic properties. Additionally, we had to bear in mind that random generators were used in the friction systems to simulate internal noise sources. Since these noise terms could not be modeled, they introduced a limit to the model precision. This limit had to be considered for the evaluation of our models.

In our case the precision limit could be very easily computed by applying perfect models for testing. These perfect models are a convenient byproduct of our artificial, numerical environment. Usually, in real world applications we do not have full knowledge about the systems that we want to model. However, in our numerical case we knew everything about friction system 1 and friction system 2 and we could use this knowledge to create models ($M^{(1)}_{perf}$, $M^{(2)}_{perf}$) for them (see Fig. 6.23). Without any internal noise source, $M^{(1)}_{perf}$ could predict and control friction system 1 flawlessly (same goes for $M^{(2)}_{perf}$ and friction system 2). However, the inherent randomness of the friction systems lead to modeling errors also in this case. Evaluating the performance of $M^{(1)}_{perf}$ and $M^{(2)}_{perf}$, we could derive a quality limit for our dynamical models.

Static evaluation

Ideally, after the training process, a model has captured the main characteristics of the system and is capable of predicting the system's behavior. Thus, before using the model for control, it is interesting to see its prediction qualities on a static data set.

For this purpose we applied the usual NMSE and MAX criteria on a test data set, which was created with the same setup as the training data (see Fig. 6.22). The only difference between creating training and test data was a different seed parameter in the random signal generator. Thus, a test signal with similar characteristics to that of the training data was created but with a different time evolution. In analogy to the training data the test data contained also $N = 100000$ data points. The test data was employed to compute the static NMSE and MAX,

$$\mathrm{NMSE}_s = \frac{100\%}{N \cdot \sigma_f^2} \sum_{i=1001}^{N} \left(F_i - \hat{F}_i\right)^2 , \tag{6.30}$$

$$\mathrm{MAX}_s = \frac{1}{\sigma_F} \max\left\{ |F_i - \hat{F}_i| \mid i = 1001, \ldots, N \right\} , \tag{6.31}$$

which were based on free-running predictions of the force values as required by the intended application in the control environment. The first 1000 data points were omitted to reduce the influence of transient behavior. The lowest possible values $\mathrm{NMSE}_s^{\mathrm{perf}}$ and $\mathrm{MAX}_s^{\mathrm{perf}}$ were evaluated with the perfect models $\mathrm{M}_{\mathrm{perf}}^{(1)}$ and $\mathrm{M}_{\mathrm{perf}}^{(2)}$.

To account for the internal noise sources of the friction systems, we repeated the computation of all evaluation criteria for 20 different realizations of the test set and averaged the values.

Dynamic evaluation

In tracking problems the success of control is determined by the magnitude of the control errors $e(t) = P(t) - P_r(t)$. The smaller these deviations are, the better the control. Therefore, we used dynamic NMSE and MAX criteria, which were defined similarly to the static NMSE and MAX and applied to the tracking errors $e(t)$. According to the setup in Fig. 6.20 a simulated control task was performed for a time duration $T = 100$ and with a time step $t_s = 0.001$. During the simulation the tracking error was recorded, yielding the time series e_i, $i = 1, \ldots, N$, with $N = 100000$ data points. This error signal was used for the computation of the dynamic NMSE and MAX values.

The absolute values of NMSE and MAX had little significance in the dynamic scenario and had to be extended by relative values. The reason for this approach is that the P-control already reduced the deviations e_i to a great extend. Therefore, low absolute values for NMSE or MAX did not automatically

indicate a good model. It was necessary to compare the control performances with and without the feed-forward control. To capture the relative improvements two extra criteria were added: α and β.

We included a time trigger in our setup that switched between pure P-control and P-control in combination with feed-forward control (see Fig. 6.20). The switching took place at a time step $T_{\text{switch}} = 50$ in the middle of the test run. During the first half only the P-controller was operating, yielding an absolute NMSE value

$$\text{NMSE}_{\text{P}} = \frac{100\%}{N \cdot \sigma_{P_r}^2} \sum_{i=20001}^{50000} e_i^2 \,, \tag{6.32}$$

with σ_{P_r} the standard deviation of the reference trajectory. The first 20000 data points were generously left out to exclude any transient behavior. After the switching at time step $T_{\text{switch}} = 50$ the feed-forward control from the RNN was added. The new NMSE was computed as

$$\text{NMSE}_{\text{ff}} = \frac{100\%}{N \cdot \sigma_{P_r}^2} \sum_{i=70001}^{100000} e_i^2 \,, \tag{6.33}$$

again leaving out the first 20000 data points.

Comparing both NMSE values the relative improvement was subsumed in the α-criterion

$$\alpha = \frac{\text{NMSE}_{\text{P}} - \text{NMSE}_{\text{ff}}}{\text{NMSE}_{\text{P}}} \cdot 100\% \,. \tag{6.34}$$

An α-value near 100% indicated a nearly flawless control, while a value near 0% indicated that the application of the dynamic model could not improve the P-control.

In analogy to NMSE the MAX criterion was defined as

$$\text{MAX}_{\text{P}} = \frac{1}{\sigma_F} \max \{|e_i| \mid i = 20001, \ldots, 50000\} \,, \tag{6.35}$$

$$\text{MAX}_{\text{ff}} = \frac{1}{\sigma_F} \max \{|e_i| \mid i = 70001, \ldots, 100000\} \,. \tag{6.36}$$

The relative improvement was described by the β-criterion

$$\beta = \frac{\text{MAX}_{\text{P}} - \text{MAX}_{\text{ff}}}{\text{MAX}_{\text{P}}} \cdot 100\% \,. \tag{6.37}$$

The lowest possible values $\text{NMSE}_{\text{ff}}^{\text{perf}}$ and $\text{MAX}_{\text{ff}}^{\text{perf}}$ were again evaluated with the perfect models $\text{M}_{\text{perf}}^{(1)}$ and $\text{M}_{\text{perf}}^{(2)}$, leading to the corresponding values α^{perf} and β^{perf}, respectively.

To account for the internal noise sources of the friction systems, we repeated the control simulation with 20 different realizations of the internal noise signals. All evaluation criteria were accordingly averaged.

6.4.5 Results

Friction system 1 (GMS)

Before starting with the modeling procedure we determined the theoretical precision limits of the modeling task. The static evaluation with the perfect model $\mathrm{M}^{(1)}_{\mathrm{perf}}$ yielded the results

$$\mathrm{NMSE}_{\mathrm{s}}^{\mathrm{perf}} = (1.5 \pm 1.7) \cdot 10^{-4}\,\%\,, \qquad \mathrm{MAX}_{\mathrm{s}}^{\mathrm{perf}} = (1.0 \pm 1.1) \cdot 10^{-2}\,.$$

These values represent the best possible performance that can be achieved for static prediction. They reveal that the internal randomness of friction system 1 plays only a minor role in its dynamics. Most of its behavior is deterministic and can be predicted from the input time series (position).

The dynamic evaluation yielded similar results

$$\begin{aligned} \mathrm{NMSE}_{\mathrm{P}} &= (8.9651 \pm 0.0005) \cdot 10^{-1}\,\%\,, & \mathrm{MAX}_{\mathrm{P}} &= (2.7981 \pm 0.004) \cdot 10^{-1}\,, \\ \mathrm{NMSE}_{\mathrm{ff}}^{\mathrm{perf}} &= (7.143 \pm 0.009) \cdot 10^{-5}\,\%\,, & \mathrm{MAX}_{\mathrm{ff}}^{\mathrm{perf}} &= (9.7 \pm 0.2) \cdot 10^{-4} \\ \alpha^{\mathrm{perf}} &= (99.992 \pm 0.008)\,\%\,, & \beta^{\mathrm{perf}} &= (99.65 \pm 0.02)\,\%\,. \end{aligned}$$

Summarizing, we can say that the internal noise should have no significant effect on the modeling precision because the behavior of friction system 1 is dominated by deterministic processes.

In the training phase the parameters of the RNN model were adapted to the training data in an offline learning procedure according to the optimization methods described in Section 5.3.4. The optimization procedure was combined with a constructive approach. Every time the optimization procedere would stop improving the results on the validation set, new elements were added to the network and the optimization was restarted. If this also did no lead to better performances, the training phase of the RNN was stopped. In our case the final structure of the RNN consisted of 27 elements with 58 internal connections. The elemental dynamics used was

$$x_i(t) = \alpha_i x_i(t-1) + \tanh\left(\sum_{j=1}^{27} c_{ij} x_j(t-1) + \sum_{k=1}^{3} b_{ik} u_k(t)\right)\,, \quad i = 1, \ldots, 27\,, \tag{6.38}$$

with the multivariate input $u_1(t) = P_r(t-100)$, $u_2(t) = P_r(t-50)$, and $u_3(t) = P_r(t)$ (see Fig. 6.20).

In the static testing phase the prediction quality of the model was evaluated (see Fig. 6.24). Here the prediction criteria were

$$\mathrm{NMSE}_{\mathrm{s}} = (3.020 \pm 0.006) \cdot 10^{-1}\,\%\,, \qquad \mathrm{MAX}_{\mathrm{s}} = (1.587 \pm 0.004) \cdot 10^{-1}\,.$$

Although these values are not as good as the ones of the perfect model, they are very good considering the kind of model that was used. As a black-box

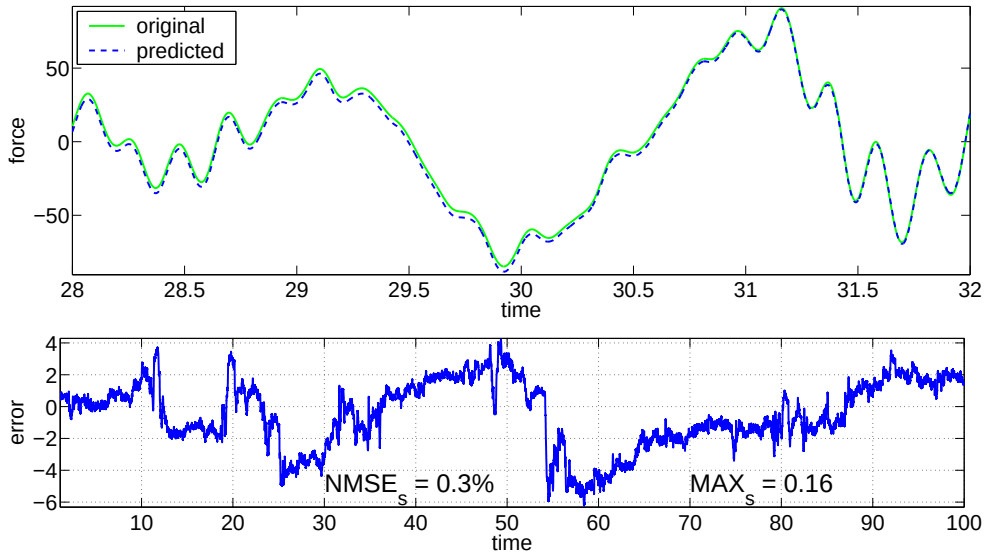

Figure 6.24: Prediction performance of RNN model for friction system 1. **Top:** The original force time series and the prediction of the model for a sample period between $t_1 = 28$ and $t_2 = 32$. **Bottom:** Prediction error for the whole test data set.

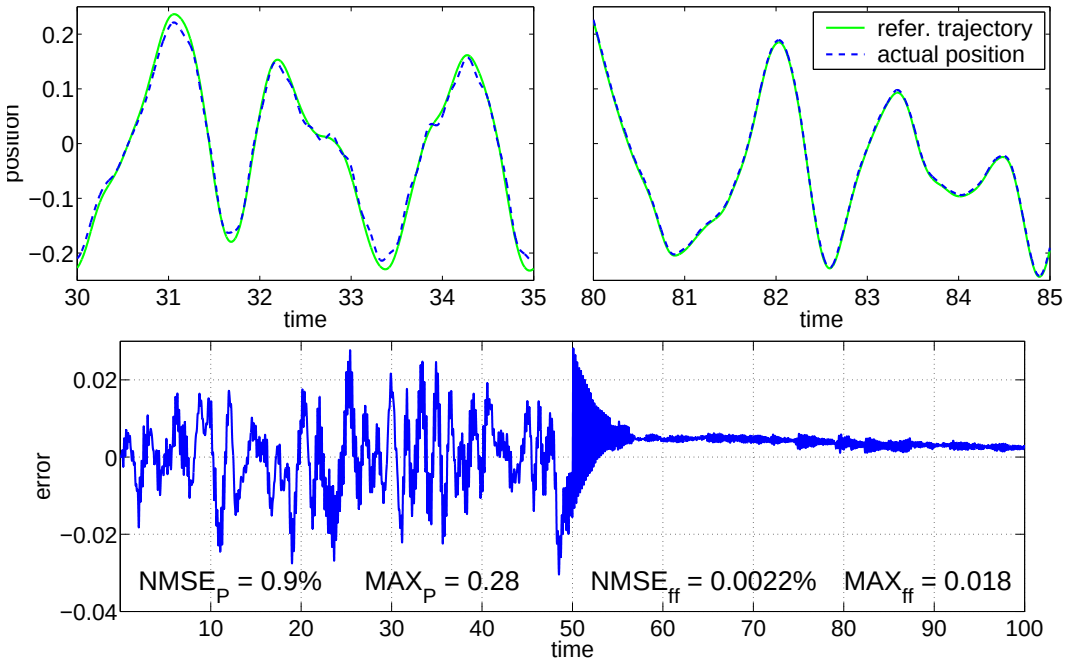

Figure 6.25: Control performance of RNN for friction system 1. **Top Left:** Reference trajectory and actual position of system for pure P-control in the time interval $[30, 35]$. **Top Right:** Reference trajectory and actual position of system for P-control with additional feed-forward control by the RNN in the time interval $[80, 85]$. **Bottom:** The tracking error for the whole simulation time $T = 100$. At switch time $t_s = 50$ feed-forward control is added to the P-control.

model the RNN was based on no other knowledge than the information in the training data. The NMSE as well as the MAX value indicate that the behavior of friction system 1 was indeed captured by the RNN.

With the good results concerning the prediction abilities of our nonlinear model, it was interesting to see how the latter would do in the control setup. The quality values for this dynamical task were

$$\begin{aligned} \mathrm{NMSE_P} &= (8.9651 \pm 0.0005) \cdot 10^{-1}\,\%\,, & \mathrm{MAX_P} &= (2.7981 \pm 0.004) \cdot 10^{-1}\,, \\ \mathrm{NMSE_{ff}} &= (2.150 \pm 0.009) \cdot 10^{-3}\,\%\,, & \mathrm{MAX_{ff}} &= (1.768 \pm 0.005) \cdot 10^{-2} \\ \alpha &= (99.760 \pm 0.008)\,\%\,, & \beta &= (93.68 \pm 0.03)\,\%\,. \end{aligned}$$

The good results from the static evaluation could be well reproduced in the dynamic environment. It is noteworthy that the averaged precision of the RNN, which is described by the NMSE and α criteria, is almost as good as that of the perfect model. Only for the maximum deviations, described by MAX and β, the RNN is slightly inferior to the perfect model. Nevertheless, the dynamical evaluation revealed a successful modeling attempt. The tracking error was considerably reduced, as can be seen in Fig. 6.25.

Friction system 2 (GF)

Again we started with determining the precision limits of the modeling task. With the perfect model $\mathrm{M}^{(2)}_{\mathrm{perf}}$ the results of the static evaluation were

$$\mathrm{NMSE}^{\mathrm{perf}}_{\mathrm{s}} = (8.2 \pm 0.6) \cdot 10^{-3}\,\%\,, \qquad \mathrm{MAX}^{\mathrm{perf}}_{\mathrm{s}} = (3.8 \pm 0.3) \cdot 10^{-2}\,.$$

Better results than these cannot be achieved by any model. The NMSE and the MAX criteria of the perfect model reveal that randomness in friction system 2 plays a greater role than in friction system 1. However, its influence is still very small in comparison to the deterministic part.

The results of the dynamic evaluation were

$$\begin{aligned} \mathrm{NMSE_P} &= (6.047 \pm 0.008) \cdot 10^{-1}\,\%\,, & \mathrm{MAX_P} &= (2.340 \pm 0.009) \cdot 10^{-1}\,, \\ \mathrm{NMSE}^{\mathrm{perf}}_{\mathrm{ff}} &= (3.5 \pm 0.8) \cdot 10^{-4}\,\%\,, & \mathrm{MAX}^{\mathrm{perf}}_{\mathrm{ff}} &= (7.0 \pm 1.1) \cdot 10^{-3} \\ \alpha^{\mathrm{perf}} &= (99.94 \pm 0.19)\,\%\,, & \beta^{\mathrm{perf}} &= (97.0 \pm 0.8)\,\%\,. \end{aligned}$$

Randomness in friction system 2 has greater effects than in friction system 1. It lowers the precision of even perfect models. However, the lowered limits are still at such a high level that they should not affect the modeling results.

The RNN was trained in exactly the same way as in the previous modeling task concerning friction system 1. Also the elemental dynamics was the same as well as the external delay lines. The final structure of the RNN consisted of 70 elements with 383 internal connections.

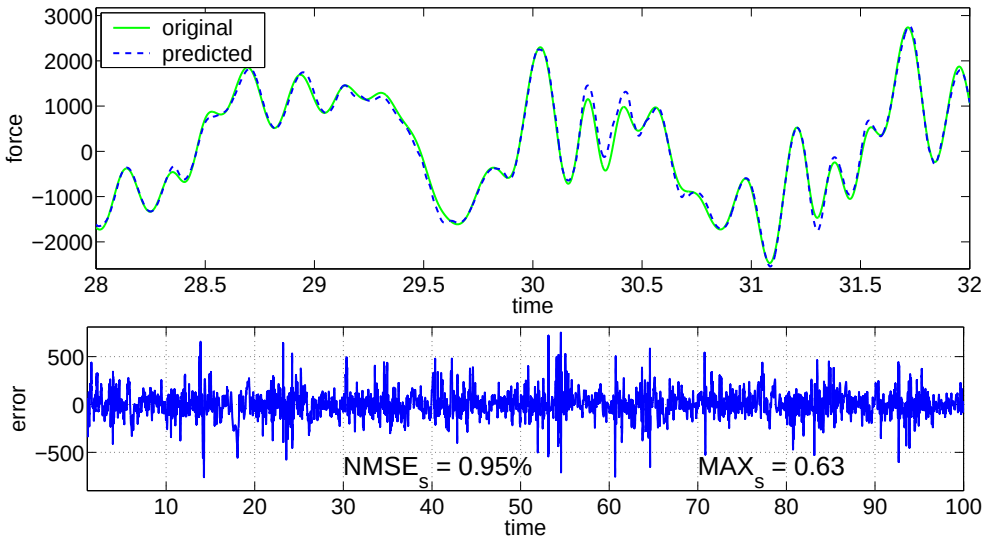

Figure 6.26: Prediction performance of RNN model for friction system 2. **Top:** The original force time series and the prediction of the model for a sample period between $t_1 = 28$ and $t_2 = 32$. **Bottom:** Prediction error over the whole test data set.

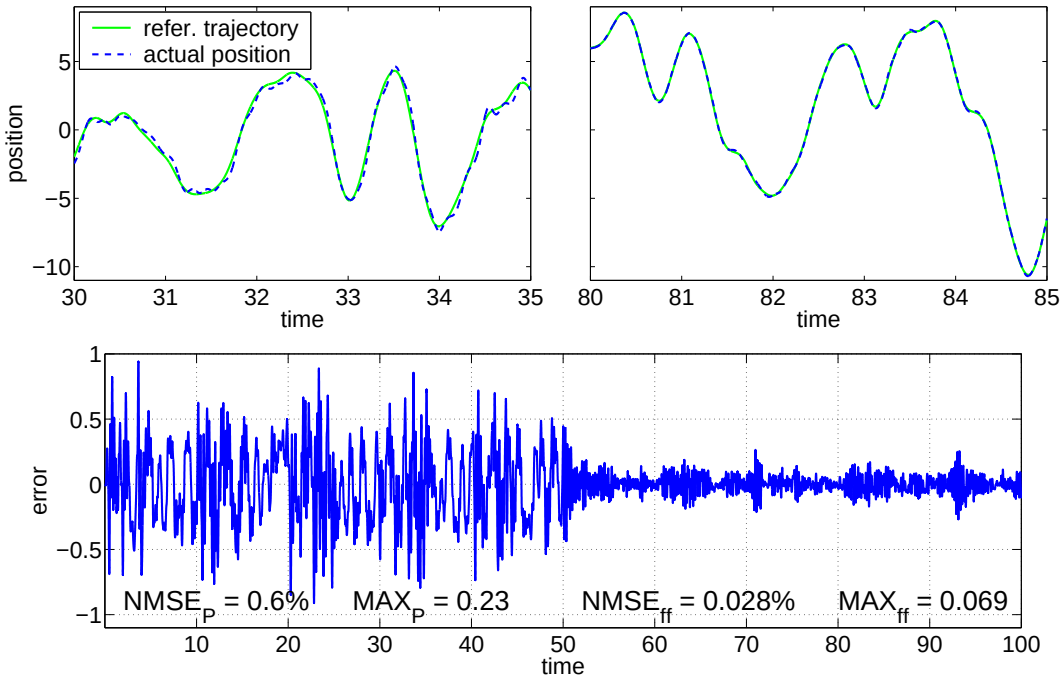

Figure 6.27: Control performance of RNN for friction system 2. **Top Left:** Reference trajectory and actual position of system for pure P-control in the time interval $[30, 35]$. **Top Right:** Reference trajectory and actual position of system for P-control with additional feed-forward control by the RNN in the time interval $[80, 85]$. **Bottom:** The tracking error for the whole simulation time $T = 100$. At switch time $t_s = 50$ feed-forward control is added to the P-control.

In the static testing phase the RNN had to predict the force value that is needed to push the plant to a specific position. The results of this task are shown in Fig. 6.26. The following performance criteria were obtained

$$\mathrm{NMSE_s} = (0.955 \pm 0.005)\,\%\,, \qquad \mathrm{MAX_s} = 0.632 \pm 0.007\,.$$

As already mentioned, the modeling of friction system 2 was much more difficult than that of friction system 1, the reason behind it being the more complex generic friction model. This difficulty is revealed in the peformance values, which are slightly worse than for friction system 1. Nevertheless, the results indicate that the RNN was still able to capture all the main characteristics of the system.

In the next step the RNN was used in the control environment (see Fig. 6.27). An evalutation yielded the following quality values for this dynamical task

$$\begin{aligned} \mathrm{NMSE_P} &= (6.051 \pm 0.009) \cdot 10^{-1}\,\%\,, & \mathrm{MAX_P} &= (2.35 \pm 0.02) \cdot 10^{-1}\,, \\ \mathrm{NMSE_{ff}} &= (2.76 \pm 0.05) \cdot 10^{-2}\,\%\,, & \mathrm{MAX_{ff}} &= (6.92 \pm 0.06) \cdot 10^{-2} \\ \alpha &= (95.43 \pm 0.23)\,\%\,, & \beta &= (70.50 \pm 0.69)\,\%\,. \end{aligned}$$

The values indicate that the RNN performed worse in the dynamical environment than the model for friction system 1. This was to be expected because it was already less accurate for static predictions than the latter. Nevertheless, the task to improve the control by an additional feedforward model can be considered successful as the averaged error was also in this case considerably lowered.

Chapter 7

Conclusion

7.1 Summary

The main focus of this thesis was to investigate the two different concepts of modeling: static modeling and dynamical modeling. Motivated by mechanisms in the human brain, the dynamical modeling approach was formulated as an alternative to the usual static modeling. It was interesting to see if a model with fading memory can hold up to the results which can be achieved with a static model, where the memory is practically lossless. Hoping to get results that were not much worse, we were pleasantly surprised to find that the dynamical modeling approach has even some advantages compared to the static approach and sometimes produces even better results than the latter.

Starting with the conventional modeling approach, static models were introduced to the reader. It was shown that static models follow a simple scheme by reducing dynamical modeling tasks to static ones. The separation of the embedding step and the regression step was marked as one weak spot of the static modeling approach. Through the separation the parameters of the model are divided in two groups and cannot be optimized in one step. Instead a try-and-error scheme has to be applied, which involves great computational effort and is not guaranteed to yield a satisfactory solution.

Another weakness of the static approach was found to be the transformation from sequential data with inherent time information to static patterns itself. Through this transformation the development phase of static models is mainly focused on one-step predictions. It was demonstrated that this inclination leads to biased estimations of model parameters if the training data is noisy. A solution to this problem was shown to be a nonlinear optimization of the parameters in the scope of free-running predictions. However, it was also shown that in the case of polynomial NARMAX models the refinement of the model parameters through the optimization on free-running predictions could not solve the bias problem completely, the reason for this being the selection process of the basis functions which is necessarily based on one-step

predictions.

An alternative approach to modeling was presented that overcomes both weaknesses of the static approach. By using the internal response signals of a dynamical system to an external input signal any desired output signal that is deterministically related to the input signal can be approximated. Thereby the internal states of the driven system function as a memory, storing the history of the input signal and the previous states of the system. Through the internal transition from one state to another the state variables produce different dynamics and can be employed as dynamical basis functions. Embedding step and regression step, which were two distinct steps in the static modeling approach, are combined into one step in the dynamical approach. Further, because of the internal storage of previous states, dynamical models are suited to be optimized on free-running predictions.

It was shown that dynamical modeling is related to synchronization. The driven system, which is utilized as a model, has to adapt to a driving signal in the same way every time the experiment is repeated. Otherwise, only qualitative but no quantitative results can be expected. The adaptation process of the driven system is exactly the same one as in generalized synchronization, where a driven system adapts to the driver. The concept of reliability was introduced that is based on generalized synchronization but is more rigorous in its demands on the driven dynamical system. It requires the dynamical system to synchronize to every possible input signal independent of the initial conditions, i.e. only one global basin of attraction is allowed. With these requirements it is possible to use reliable systems as pools of different dynamics for dynamical modeling.

It was argued that low-dimensional dynamical systems are not suited as pools of response signals because they obviously cannot create the necessary diversity. On the other hand, reliability is difficult to ensure in high-dimensional systems. Therefore, models with a network structure consisting of coupled low-dimensional elements were favored as dynamical models where reliability can be proved more easily. The theoretical concept of dynamical modeling was demonstrated with a network-like model that uses synchronized Lorenz systems as elements.

It was shown that dynamical modeling is already applied for some time in the modeling with Recurrent Neural Networks (RNNs). Especially the concept of Echo State Networks (ESNs) fits very nicely into the philosophy of dynamical modeling. A stable RNN is created and used as a reservoir of different dynamics which are combined to a common output. Instead of optimizing every connection in the network, the ESNs use a randomly connected structure and compensate the missing nonlinear optimizations by the greater number of neurons. It was demonstrated that the usual stability criterion of ESNs, which requires the spectral radius of the connection matrix to be smaller unity, sometimes produces unreliable RNNs and has to be used with great care.

Further, the usage of an external loop was examined for the prediction and cross-prediction tasks. It was shown that in most of the cases the loop leads to unreliable networks, which has to be used in special way that involves a tuning-in phase in which the network has to find the correct initial state. Although networks with external loops seem to produce better prediction results in some cases, they are not reliable and thus prone to produce unexpected results. As such they are also to be treated carefully and have to be thoroughly tested for instabilities. As improvements to the application of RNNs a selection scheme based on the Fast Orthogonal Search was presented and a refinement algorithm that can cope with bias in RNNs with external loops.

It was shown that randomly created networks can display a rather diverse performance. In order to produce a good model the random creation process should be repeated for a certain number of times and the best network chosen afterwards. With not much more computational effort smaller networks with better performance values can be produced by adjusting the network parameters to the data. This adjustment does not depend on a thorough optimization algorithm like Backpropagation Through Time, which is often applied for small networks. By using the fact that a middle sized network already represents a pool of dynamics with an acceptable diversity, only minor changes to some network parameters are necessary. It was demonstrated that middle sized networks optimized with a rather crude algorithm based on Simulated Annealing proved superior to large random networks.

The application of static NARMAX models and dynamical RNNs was tested on friction data. On various examples ranging from cross-predictions on simulated and measured data to control applications it was shown that dynamical models are well suited to capture the different phenomena of friction. In most cases RNNs could well compete even with the physics based models and was sometimes superior to them.

7.2 Outlook

As probably all works do, this thesis gives answers to some questions but also leads to more questions itself. Some of the ideas for further investigations are given here.

The foremost question might be: Is there an easy way to produce reliable high-dimensional dynamical models? A possible route to this goal was shown in constructing reliable systems by coupling low-dimensional reliable systems. An example was given with the coupled Lorenz systems. Unfortunately, the coupling scheme had to be limited to forward connections without any recurrent loops for the sake of stability. A greater diversity could be expected from coupling schemes with recurrent loops. Such systems were presented with the Recurrent Neural Networks, where the question of reliability can be reduced

to properties of the connection matrix. However, the problem of producing reliable systems is not really solved. Can we find a simple rule to produce reliable dynamical models, whose elements have a more interesting dynamics than the saturating sigmoid transformation in neurons?

It was shown that the introduction of an external loop can improve the modeling results for RNNs. However, the price is often an unreliable system. Is there a way to ensure that the instability introduced with the loop is as small as possible? The goal should be finding a parameter that can balance prediction quality and model stability.

Numerical experiments indicated that dynamical models can deal with data that is slightly non-stationary. Further studies should be made to investigate this phenomenon.

One of the main disadvantages of the presented modeling techniques concerning dynamical modeling is that they are not user-friendly. There are many free parameters that have to be chosen without having a simple rule how to do so. In our numerous simulations we have gained the expertise or rather a 'feeling' for which parameter values are more suited than others in certain circumstances. However, a future goal has to be developing an automatic procedure that enables non-experts to use the models.

Appendix A

Training of black-box models

This chapter gives an short review of the main techniques of model learning[1] that were used throughout the thesis. More specifically, it deals with methods that belong to the category of so-called *supervised learning* of *black-box models*. Supervised means that the desired behavior of the model is known on a set of examples and that this knowledge is used during the learning process to tune the model parameters. Black-box means that besides these examples no other information is incorporated in the model structure. This is different with the so called *white-box* and *gray-box* models, which both make use of prior knowledge about the specific process that is to be modeled (see [49]).

The set of examples is a finite data set, also known as *training set*. During the learning procedure the model performance on the training set is evaluated and optimized by adjusting the model parameters.

Roughly speaking, supervised learning copes with two tasks. The fist one is to find the right means to evaluate model performance. The knowledge about the desired behavior of a model has to be made mathematically seizable in form of a function that measures the performance of models. Such a function is referred to as *evaluation function* or *cost function* (see Section A.1). The second task is to adjust the model parameters with the help of the cost function. This is done within the scope of an optimization problem (see Section A.2).

A.1 Cost functions

After having chosen the type and the basic structure of the model, its free parameters have to be adapted in the training procedure. In supervised learning information about the model behavior is present in form of examples, which is called the *training set*. Compliance with this behavior marks good models, while noncompliance marks bad models. An *evaluation function* translates the

[1] Following the common usage, *training* and *learning* are treated as synonyms in this chapter.

criteria 'good' and 'bad' into numbers by looking at the performance of the model on the training set. Usually, it is formulated as a *cost* or *loss function*[2], which measures the deviations of the model from the training examples. In this sense, good models have low cost values and bad models have high cost values. The optimal model, i.e. the model with the best possible parameters, is characterized by producing the lowest cost value.

As an example consider a model function $g(\cdot|\boldsymbol{w})$, which depends on the parameters $\boldsymbol{w} = (w_1, \ldots, w_M)^T \in \mathbb{R}^M$. Further, a training set is given with N patterns $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$, consisting of regressors $\boldsymbol{x}_i$ and targets y_i (see Section 2.1). If the model is applied to the regressors $\boldsymbol{x}_i$ it produces outputs

$$\hat{y}_i = g(\boldsymbol{x}_i\,|\boldsymbol{w})\,, \tag{A.1}$$

which depend on the parameters $w_1, \ldots, w_M$. Usually, the goal of the learning procedure is to find the parameters, for which the model outputs $\hat{y}_i$ deviate as less as possible from the original targets y_i. How these deviations are measured depends on the cost function.

The most popular choice for the cost function is the sum of squared errors (SSE)

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - g(\boldsymbol{x}_i|\boldsymbol{w})\right)^2 . \tag{A.2}$$

Minimization of this cost function is called *least squares minimization*. Note that the mean squared error $\mathrm{MSE} = \mathrm{SSE}/N$ and the normalized mean squared error $\mathrm{NMSE} = \mathrm{MSE}/\sigma_y^2$ are equivalent to the SSE for minimization purposes, because the constant factors do not change the position of the minimum.

Other cost functions are the sum of absolute errors

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{N} \left|y_i - g(\boldsymbol{x}_i|\boldsymbol{w})\right| , \tag{A.3}$$

or the maximum error

$$\mathcal{L}(\boldsymbol{w}) = \max_i \left\{\left|y_i - g(\boldsymbol{x}_i|\boldsymbol{w})\right|\right\} . \tag{A.4}$$

It depends on the modeling objective which of these possible cost functions is adequate. Each one has advantages and disadvantages in combination with certain noise properties of the data (see e.g. [49]).

A.2 Optimization

In the $(M+1)$-dimensional space $\mathbb{R}^{M+1}$ the cost function $\mathcal{L} : \mathbb{R}^M \to \mathbb{R}$, with the parameters $\boldsymbol{w} \in \mathbb{R}^M$ of the model as arguments, describes an M-dimensional

[2]The terms 'cost function' or 'loss function' originate from the economics, where bad solutions to a problem are associated with higher costs or losses, respectively.

manifold

$$\mathcal{M} = \{(v, \boldsymbol{w}) \in \mathbb{R} \times \mathbb{R}^M | v = \mathcal{L}(\boldsymbol{w}), \boldsymbol{w} \in \mathbb{R}^M\} . \tag{A.5}$$

Picturing this manifold as a 3-dimensional landscape with mountains and valleys, the parameters of the optimal model are the coordinates of the lowest point in the deepest valley of this landscape. This is the global minimum of the cost function, and it depends on the smoothness and the curvature of the manifold if it is easy or difficult to find.

If $\mathcal{L}(\cdot)$ depends linearly on the parameters $\boldsymbol{w}$, the manifold $\mathcal{M}$ is a plane and the minimum can be found at the boundaries of the parameters[3]. If $\mathcal{L}(\cdot)$ is quadratic in the parameters $\boldsymbol{w}$, the manifold $\mathcal{M}$ is a parabolic surface with a unique global minimum. Both cases, the linear and the quadratic, belong to the class of problems that can be solved by *global optimization*, i.e. the optimal solution can be computed taking into consideration the entire parameter space $\mathbb{R}^M$.

However, in general the dependencies of cost function $\mathcal{L}(\cdot)$ on the parameters $\boldsymbol{w}$ are neither linear nor quadratic but much more complex. Typically, manifold $\mathcal{M}$ is wildly shaped and the global minimum coexists next to many local minima. In this case there is seldom a simple analytical solution and global optimization cannot be used. Instead, one has to resort to *local optimization*. Starting from an arbitrary point on the manifold $\mathcal{M}$ subsequent points are chosen in such a way as to improve the performance in the long run. How the points are chosen depends on the algorithm that is used. There are many such algorithms but all have in common that they can operate only locally on the manifold and make only small improvements in an iterative way. If the algorithm is sound, chances are high that the path of small improvements eventually leads to the global minimum of $\mathcal{L}(\cdot)$. Unfortunately, there is no guarantee that the global minimum will be found at all.

Local optimization algorithms can be divided into two main streams: the *gradient-based* and the *non-gradient-based methods*. The former use information about the first and sometimes even the second derivative of function $\mathcal{L}(\cdot)$ for improvements, while the latter do not rely on this knowledge but employ direct comparisons of function values.

A.2.1 Quadratic cost function

If the cost function is quadratic in the parameters, the learning problem is convex and can be solved with a global method. This is usually preferable to nonlinear, local optimization techniques, because the latter are notorious for getting stuck in local minima. For this reason many model architectures try

[3] All statements in this section are restricted to optimization without additional constraints on the parameters aside from boundedness in the linear case. By introducing constraints even the simple linear and quadratic optimization problems can become very difficult to solve.

to incorporate parameters in a linear way, so that the popular cost function based on squared errors becomes quadratic. For example NARMAX models are weighted sums of basis functions

$$g(\boldsymbol{x}_t|w_1,\ldots,w_M) = \sum_{i=1}^{M} w_i g_i(\boldsymbol{x}_t)\,. \tag{A.6}$$

with weights $w_1,\ldots,w_M$ as linear parameters (see Section 3.1.2). A similar case are Echo State Networks, where the basis functions are represented by internal states and the output weights are also linear parameters of the model (see Section 5.2.3).

For these models the optimal set of weights is determined by minimizing the sum of squared errors

$$\mathcal{L}(w_1,\ldots,w_M) = \sum_{t=1}^{N} \left(y_t - g(\boldsymbol{x}_t|w_1,\ldots,w_M)\right)^2 = \sum_{t=1}^{N} \left(y_t - \hat{y}_t\right)^2\,, \tag{A.7}$$

or in matrix notation

$$\mathcal{L}(\boldsymbol{w}) = (\boldsymbol{y}-\hat{\boldsymbol{y}})^T \cdot (\boldsymbol{y}-\hat{\boldsymbol{y}}) = (\boldsymbol{y}-\tilde{\boldsymbol{X}}\boldsymbol{w})^T \cdot (\boldsymbol{y}-\tilde{\boldsymbol{X}}\boldsymbol{w})\,, \tag{A.8}$$

with the original target vector $\boldsymbol{y} = (y_1,\ldots,y_N)^T$, the estimated target vector $\hat{\boldsymbol{y}} = (\hat{y}_1,\ldots,\hat{y}_N)^T$, the weight vector $\boldsymbol{w} = (w_1,\ldots,w_M)^T$, and the regressor matrix

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} g_1(\boldsymbol{x}_1) & \cdots & g_M(\boldsymbol{x}_1) \\ \vdots & \ldots & \vdots \\ g_1(\boldsymbol{x}_N) & \cdots & g_M(\boldsymbol{x}_N) \end{pmatrix}\,. \tag{A.9}$$

Since the model is linear in the parameters w_i, the minimization problem is convex and thus has a unique minimum if the rank of $\tilde{\boldsymbol{X}}$ is not smaller than M. An important requirement for the latter condition is that $N \geq M$, i.e. there have to be at least as many data points as parameters. The solution is easily computed via

$$\hat{\boldsymbol{w}} = \left(\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^T\boldsymbol{y}\,. \tag{A.10}$$

The matrix

$$\tilde{\boldsymbol{X}}^\dagger \equiv \left(\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^T\,, \tag{A.11}$$

is usually referred to as the *pseudo inverse* of matrix $\tilde{\boldsymbol{X}}$ [49]. The solution in Eq. (A.10) is not always well defined. If for example there are two basis functions that are highly correlated or even equal, there are infinitely many solutions for the weights $\boldsymbol{w}$. Because of this reason the weights are usually not determined by explicitly computing the pseudo inverse. Instead the *Singular Value Decomposition* (SVD) of $\tilde{\boldsymbol{X}}$ is used [62].

A.2.2 Fast Orthogonal Search

The usage of black-box models implies that besides the training data no additional assumptions are made concerning the functional relationship between regressors and targets. Therefore, there are no informations about the correct form of the model. For example, it is impossible to know beforehand which basis functions are useful in a NARMAX model and which are not. Similarly, it is not obvious how to choose the number of elements in RNNs. Therefore, a simple strategy is to create a flexible model, which can be adapted to all kinds of data in different situations. That means for NARMAX models and RNNs to include a large number of basis functions or elements, respectively, in order to be prepared for all eventualities.

However, this strategy has many flaws. One problem is that it introduces redundancies into the model by correlated basis functions or internal states, respectively. Even if numerical instabilities can be avoided by techniques like the SVD, redundant model components lead to unnecessarily large models, which can become unwieldy in applications. Worse, the effects of *overfitting* become relevant (see Section A.3).

A solution to the overfitting problem and the problem of too large models was found with the so called *forward orthogonal selection methods*, the *Fast Orthogonal Search* (FOS) being one of the most prominent among them [43], [44], [18]. The FOS was specifically designed for NARMAX models. Therefore, we will describe it in this scope here. However, FOS is not bound to NARMAX models. An application to RNNs is presented in Section 5.3.3.

Instead of including all basis functions from the start, the idea of FOS is to choose only the best ones into the model. One starts with an empty model and a large pool of possible basis functions. Then, step by step, functions from the pool are selected. The selection criterion is based on evaluations on the training data set. Every function is tested in its ability to reduce the model error. The basis function that yields the highest error reduction is then selected, while all the other basis functions in the pool are made orthogonal to it. The orthogonalization makes the evaluation of the basis functions in the pool independent of the already chosen basis functions. In this way redundant basis function cannot be chosen.

If $\mathcal{P} = \{g_i(\cdot) \,|\, i = 1, \ldots, K\}$ is the pool of basis functions and set $I_1 = \{1, \ldots, K\}$ denotes the indices of all basis function in the pool, the FOS algorithm can be expressed as

step 1:

$$\tilde{D}(1,1,i) = \langle \boldsymbol{g}_i \,|\, \boldsymbol{g}_i \rangle\,, \quad \tilde{C}(1,i) = \langle \boldsymbol{y} \,|\, \boldsymbol{g}_i \rangle\,, \quad \forall i \in I_1$$
$$\tilde{Q}(1,i) = \tilde{C}(1,i)^2/\tilde{D}(1,1,i)\,, \quad \forall i \in I_1$$
$$i_1 = \arg\max_{i \in I_1}(\tilde{Q}(1,i))\,, \quad I_2 = I_1 \setminus i_1$$
$$D(1) = \tilde{D}(1,1,i_1)\,, \quad C(1) = \tilde{C}(1,i_1)\,, \quad Q(1) = \tilde{Q}(1,i_1)$$

step 2:

$$\tilde{D}(2,1,i) = \langle \boldsymbol{g}_i \,|\, \boldsymbol{g}_{i_1} \rangle\,, \quad \forall i \in I_2$$
$$\tilde{D}(2,2,i) = \tilde{D}(1,1,i) - \tilde{D}(2,1,i)^2/D(1)\,, \quad \forall i \in I_2$$
$$\tilde{C}(2,i) = \tilde{C}(1,i) - \tilde{D}(2,1,i) \cdot C(1)/D(1)\,, \quad \forall i \in I_2$$
$$\tilde{Q}(2,i) = \tilde{C}(1,i)^2/\tilde{D}(2,2,i)\,, \quad \forall i \in I_2$$
$$i_2 = \arg\max_{i \in I_2}(\tilde{Q}(2,i))\,, \quad I_3 = I_2 \setminus i_2$$
$$\alpha(2,1) = \tilde{D}(2,1,i_2)/D(1)$$
$$D(2) = \tilde{D}(2,2,i_2)\,, \quad C(2) = \tilde{C}(2,i_2)\,, \quad Q(2) = \tilde{Q}(2,i_2)$$

$\vdots$

step m:

$$\tilde{D}(m,j,i) = \tilde{D}(m-1,j,i)\,, \quad j = 1,\ldots,m-2\,,\ \forall i \in I_m$$
$$\tilde{D}(m,m-1,i) = \langle \boldsymbol{g}_i \,|\, \boldsymbol{g}_{i_{m-1}} \rangle - \sum_{j=1}^{m-2} \alpha(m-1,j) \cdot \tilde{D}(m,j,i)\,, \quad \forall i \in I_m$$
$$\tilde{D}(m,m,i) = \tilde{D}(m-1,m-1,i) - \tilde{D}(m,m-1,i)^2/D(m-1)\,, \quad \forall i \in I_m$$
$$\tilde{C}(m,i) = \tilde{C}(m-1,i) - \tilde{D}(m,m-1,i) \cdot C(m-1)/D(m-1)\,, \quad \forall i \in I_m$$
$$\tilde{Q}(m,i) = \tilde{C}(m,i)^2/\tilde{D}(m,m,i)\,, \quad \forall i \in I_m$$
$$i_m = \arg\max_{i \in I_m}(\tilde{Q}(m,i))\,, \quad I_{m+1} = I_m \setminus i_m$$
$$\alpha(m,k) = \tilde{D}(m,k,i_m)/D(k)\,, \quad k = 1,\ldots,m-1$$
$$D(m) = \tilde{D}(m,m,i_m)\,, \quad C(m) = \tilde{C}(m,i_m)\,, \quad Q(m) = \tilde{Q}(m,i_m)$$

In the description above the vector values of the basis function and the target values are defined as

$$\boldsymbol{g}_i \equiv (g_i(\boldsymbol{x}_1,\ldots,g_i(\boldsymbol{x}_N))^t \quad \text{and } \boldsymbol{y} \equiv (y_1,\ldots,y_N)^T\,, \tag{A.12}$$

and the product $\langle \cdot \,|\, \cdot \rangle$ as

$$\langle \boldsymbol{g}_i \,|\, \boldsymbol{g}_j \rangle \equiv \sum_{t=1}^{N} g_i(\boldsymbol{x}_t) g_j(\boldsymbol{x}_t) \,. \tag{A.13}$$

Iteratively the basis functions $g_{i_1}(\cdot), g_{i_2}(\cdot), \ldots$ are chosen from the pool. The variable $\tilde{Q}(m,i)$ computed at the m-th iteration step is the error reduction that could be gained by choosing the i-th basis function $g_i(\cdot)$ into the model. Therefore, for every time step m the basis function with the maximum value of $\tilde{Q}(m,i)$ is selected.

The application of FOS makes it necessary to find an appropriate stopping criterion for the selection process. This can either be the training error falling below a threshold, which is set manually by the user, or the rising of the test error on a validation set (see Section A.3). If the selection algorithm was stopped after the M-th basis function was chosen, the parameters w_i, $i = 1, \ldots, M$, have to be computed. They can be easily gained by the recursive formula

$$w_m = \sum_{i=m}^{M} v_i \frac{C(i)}{D(i)} \,, \quad m = 1, \ldots, M \,, \tag{A.14}$$

with

$$v_m = 1 \,, \quad v_i = -\sum_{r=m}^{i-1} \alpha_{ir} v_r \,, \quad i = m+1, \ldots, M \,. \tag{A.15}$$

A.3 Overfitting

On one hand a model has to be flexible in order to describe various functional relationships between data points. This means that the model must have an appropriate number of free parameters that can be adapted to the data. On the other had a model can be fitted to any data set if it is only flexible enough. An extreme case is the fitting of a line to two points in the space. Too much flexibility leads to models that loose their ability to generalize. Instead these models try to describe artificial relationships in the data, which is caused by random factors such as measurement errors. They are overadapted to the data, a phenomenon referred to as *overfitting*. Keeping the balance between a model that is too rigid and one that is too flexible is an important problem in modeling and is known under the name *bias-variance dilemma* or *bias-variance trade-off* (e.g. see [49]).

In the FOS algorithm, described in Section A.2.2, the model is iteratively extended by adding new basis functions. This is equivalent to making the model more flexible, which eventually leads to overfitting. The selection

method helps avoiding redundancies but does not solve the overfitting problem. A criterion is necessary which is able to reveal the first signs of overfitting and which can be used to stop the selection process at this point.

This criterion is found by testing the performance of the model on an independent data set, which was not utilized for parameter adaptation. For this purpose the data is split up in three different sets, which are the *training set*, the *validation set*, and the *test set* (e.g. see [49]). The training set is used exclusively for the adaptation of the model parameters. The magnitude of model errors on this data set tends to decrease with every extra parameter in the model. That means that with every selection from the pool the error on the training set becomes smaller. An overadaptation is not revealed with the model error on the training set.

However, if the model is evaluated on an extra data set, namely the validation set, the situation changes. Since the model parameters are adjusted only to the training data, the model error on the validation set reveals the generalization abilities of the model. If the new selected basis function captures relevant features of the data, the error on the validation set decreases. However, if overfitting occurs, the error rises. This rising can be used as a stopping criterion for the selection process.

The last of the three data sets, the test data, is needed to make an unbiased estimate of the prediction performance of the model. This cannot be done with the validation set because, similarly to the training set, it was used for the adjustment of the model structure. Therefore, the error on the validation set would yield a too optimistic view on the model performance.

Appendix B

Miscellaneous

B.1 Biased parameter estimations

This section contains computations, which were left out of Section 3.2.2 for reasons of readability. To understand the motivation and notation of this section, background knowledge from Section 3.2.2 is essential.

In the following an uncorrelated, zero-mean input signal $\{u_t\}_{t\in\mathbb{I}}$ with normally distributed values $u_t \sim \mathrm{N}(0, \sigma_u^2)$ is assumed. As described in Section 3.2.2, in this case, the parameter a is biased while parameter b is bias-free. This is valid only if the ordinary cost function MSE_1 is used, which is an average of the squared one-step-ahead errors. In this section the effects of using multi-step prediction errors in the cost function are examined.

2-step predictions

Regard the errors $e_{t+1} = \tilde{y}_{t+1} - \hat{y}_{t+1}$ of a 2-step prediction. With

$$\begin{aligned} \tilde{y}_{t+1} &= a_0 y_t + b_0 u_{t+1} + \varepsilon_{t+1} \\ &= a_0^2 y_{t-1} + a_0 b_0 u_t + b_0 u_{t+1} + \varepsilon_{t+1} \end{aligned} \tag{B.1}$$

and

$$\begin{aligned} \hat{y}_{t+1} &= a\hat{y}_t + bu_{t+1} \\ &= a^2\tilde{y}_{t-1} + abu_t + bu_{t+1} \\ &= a^2 y_{t-1} + a^2\varepsilon_{t-1} + abu_t + bu_{t+1}\,, \end{aligned} \tag{B.2}$$

the errors can be written as

$$e_{t+1} = (a_0^2 - a^2)y_{t-1} - a^2\varepsilon_{t-1} + (a_0 b_0 - ab)u_t + (b_0 - b)u_{t+1} + \varepsilon_{t+1}\,. \tag{B.3}$$

If we insert these 2-step prediction errors in Eq. (3.31) instead of the one-step-

ahead errors, the cost function reads

$$\begin{aligned} \mathrm{MSE}_2(a,b) &= \lim_{N\to\infty}\frac{1}{N}\sum_{t=1}^{N} e_t^2 \\ &= (a_0^2-a^2)^2\sigma_y^2 + a^4\sigma_\varepsilon^2 + (a_0b_0-ab)^2\sigma_u^2 + (b_0-b)^2\sigma_u^2 + \sigma_\varepsilon^2 \end{aligned} \tag{B.4}$$

Compared to the previous cost function the influence of the interfering term $a^4\sigma_\varepsilon^2$ is decreased. This leads to a smaller bias in the parameter a.

$(m+1)$-step predictions

More general, look at the errors $e_{t+m} = \tilde{y}_{t+m} - \hat{y}_{t+m}$ of $(m+1)$-step predictions. With

$$\tilde{y}_{t+m} = a_0^{m+1}y_{t-1} + b_0\sum_{k=0}^{m} a_0^k u_{t+m-k} + \varepsilon_{t+m}\,, \tag{B.5}$$

and

$$\hat{y}_{t+1} = a^{m+1}y_{t-1} + a^{m+1}\varepsilon_{t-1} + b\sum_{k=0}^{m} a^k u_{t+m-k}\,, \tag{B.6}$$

the error is

$$e_{t+m} = (a_0^{m+1}-a^{m+1})y_{t-1} - a^{m+1}\varepsilon_{t-1} + \sum_{k=0}^{m}(b_0a_0^k - ba^k)u_{t+m-k} + \varepsilon_{t+m}\,. \tag{B.7}$$

Inserting these errors into the cost function yields

$$\begin{aligned} \mathrm{MSE}_{(m+1)}(a,b) &= \lim_{N\to\infty}\frac{1}{N}\sum_{t=1}^{N} e_t^2 \\ &= (a_0^{m+1}-a^{m+1})^2\sigma_y^2 - a^{2m+2}\sigma_\varepsilon^2 + \sum_{k=0}^{m}(b_0a_0^k - ba^k)^2\sigma_u^2 + \sigma_\varepsilon^2 \end{aligned} \tag{B.8}$$

The influence of the interfering term $a^{2m+2}\sigma_\varepsilon^2$ is decreased even more if the number of prediction steps $m+1$ is increased.

free-running predictions

The transition to free-running prediction errors is simply made by looking at the limes $m\to\infty$ for the $(m+1)$-step prediction errors. Then the cost function reads

$$\begin{aligned} \mathrm{MSE}_\infty(a,b) &= \lim_{m\to\infty}\mathrm{MSE}_{(m+1)}(a,b) \\ &= \sum_{k=0}^{\infty}(b_0a_0^k - ba^k)^2\sigma_u^2 + \sigma_\varepsilon^2\,. \end{aligned} \tag{B.9}$$

The terms $(a_0^{m+1} - a^{m+1})^2\sigma_y^2$ and $a^{2m+2}\sigma_\varepsilon^2$ vanish for $m \to \infty$ because of the conditions $|a_0| < 0$ and $|a| < 0$ for stable systems and models, respectively.

Necessary conditions for the minimum of the cost function are vanishing derivations

$$\frac{\partial \mathrm{MSE}_\infty(a,b)}{\partial a} = -2\sigma_u^2 \sum_{k=0}^{\infty} k a^{k-1}(b_0 a_0^k - b a^k) \stackrel{!}{=} 0\,, \tag{B.10}$$

$$\frac{\partial \mathrm{MSE}_\infty(a,b)}{\partial b} = -2\sigma_u^2 \sum_{k=0}^{\infty} a^k(b_0 a_0^k - b a^k) \stackrel{!}{=} 0\,. \tag{B.11}$$

With the known limit values of the geometric series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{and} \quad \sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2} \quad \text{for } |x| < 1\,, \tag{B.12}$$

it follows from Eq. (B.10) and Eq. (B.11) that

$$b_0 a_0 (1-a^2)^2 = b a (1 - a_0 a)^2\,, \tag{B.13}$$

and

$$b = \frac{1-a^2}{1-a_0 a} b\,. \tag{B.14}$$

Inserting the latter equation into the former one yields

$$a = a_0\,. \tag{B.15}$$

and following from that

$$b = b_0\,. \tag{B.16}$$

Thus, the main results of this section is: *Both parameters a and b are bias-free if they are estimated by minimizing the MSE_∞ cost function based on free-running errors.*

Bibliography

[1] H. D. I. Abarbanel, N. F. Rulkov, and M. M. Sushchik. Generalized synchronization of chaos: The auxilliary system approach. *Physical Review E*, 53(5):4528–4535, 1996.

[2] V.S. Afraimovich, N.N. Verichev, and M.I. Rabinovich. Stochastic synchronization of oscillation in dissipative systems. *Radiophysics and Quantum Electronics*, 29:747–751 (795–803), 1986.

[3] L. A. Aguirre and S. A. Billings. Retrieving dynamical invariants from chaotic data using NARMAX models. *International Journal of Bifurcation and Chaos*, 5(2):449–474, 1995.

[4] F. Al-Bender, V. Lampaert, S. D. Fassois, D. D. Rizos, K. Worden, D. Engster, A. Hornstein, and U. Parlitz. Measurement and identification of pre–sliding friction dynamics. In *4th International Symposium, Investigations of Non–Linear Dynamics Effects in Production Systems (Volkswagenstiftung)*, 2003.

[5] F. Al-Bender, V. Lampaert, S. D. Fassois, D. D. Rizos, K. Worden, D. Engster, A. Hornstein, and U. Parlitz. *Measurement and identification of pre–sliding friction dynamics*, pages 349–367. in: Nonlinear Dynamics of Production Systems (G. Radons and R. Neugebauer, Eds.). Wiley–VCH Verlag, Weinheim, 2004.

[6] F. Al-Bender, V. Lampaert, and J. Swevers. Modeling of dry sliding friction dynamics: From heuristic models to physically motivated models and back. *CHAOS*, 14(2):446–460, 2004.

[7] F. Al-Bender, V. Lampaert, and J. Swevers. A novel generic model at asperity level for dry friction force dynamics. *Tribology Letters*, 16(1):81–93, 2004.

[8] G. Amontons. De la résistance causée dans les machines. *Mémoires de l'Academie des Sciences*, pages 203–222, 1699.

[9] B. Armstrong-Hélouvry, P. Dupont, and C. Canudas de Wit. A survey of models, analysis tools and compensation methods for the control of machines with friction. *Automatica*, 30(7):1083–1138, 1994.

[10] D. G. Aronson, G. B. Ermentrout, and N. Koppel. Amplitude response of coupled oscillators. *Physica D*, 41(3):403–449, 1990.

[11] P. Bérenyi, G. Horváth, V. Lampaert, and J. Swevers. Non–local hysteresis function identification and compensation with neural networks. *Periodica Polytechnica/Electrical Engineering*, 47((3–4)):253–267, 2003.

[12] S. A. Billings and D. Coca. Discrete wavelet models for identification and qualitative analysis of chaotic systems. *International Journal of Bifurcation and Chaos*, 9(7):1263–1284, 1999.

[13] S. Boccaletti, J. Kurths, G. Osipov, D. L. Valladares, and C. S. Zhou. The synchronization of chaotic systems. *Physics Reports*, 366(1–2):1–101, 2002.

[14] F. P. Bowden and D. Tabor. *The Friction and Lubrication of Solids (Part I)*. Clarendon Press, Oxford, UK, 1950.

[15] F. P. Bowden and D. Tabor. *The Friction and Lubrication of Solids (Part II)*. Clarendon Press, Oxford, UK, 1964.

[16] C. Canudas de Wit, H. Olsson, K. Åström, and P. Lischinsky. A new model for control of systems with friction. *IEEE Transactions on Automatic Control*, 40:419–425, 1995.

[17] S. Chen and S. A. Billings. Representation of non–linear systems: the NARMAX model. *International Journal of Control*, 49(3):1013–1032, 1989.

[18] S. Chen, S. A. Billings, and W. Luo. Orthogonal least squares methods and their application to non–linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

[19] K. H. Chon, J. K. Kanters, R. J. Cohen, and N. Holstein-Rathlou. Detection of chaotic determinism in time series from randomly forced maps. *Physica D*, 99:471–486, 1997.

[20] M. V. Corrêa, L. A. Aguirre, and Mendes E. M. A. M. Modelling chaotic dynamics with discrete nonlinear rational models. *International Journal of Bifurcation and Chaos*, 10(5):1019–1032, 2000.

[21] C. A. Coulomb. Théorie des machines simples. *Mémoires de Mathématique et de Physique de l'Academie des Sciences*, pages 161–331, 1785.

[22] J. Courtney-Pratt and E. Eisner. The effect of a tangential force on the contact of metallic bodies. *Proceedings of the Royal Society*, A238:529–550, 1957.

[23] P. Dahl. A solid friction model. Technical Report TOR-0158(3107-18), The Aerospace Corporation, El Segundo, CA, 1968.

[24] J. T. Desaguliers. Some experiments concerning the cohesion of lead. *Phil. Trans. Roy. Soc. London*, 33, 1725.

[25] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[26] L. Euler. Sur le frottement des corps solids. *Histrorie de l'Academie Royale a Berlin*, pages 122–132, 1748.

[27] H. Fujisaka and T. Yamada. Stability theory of synchronized motion in coupled–oscillator systems. *Progress of Theoretical Physics*, 69(1):32–47, 1983.

[28] K. Geist, U. Parlitz, and W. Lauterborn. Comparison of different methods for computing lyapunov exponents. *Progress of Theoretical Physics*, 83(5):875–893, 1990.

[29] Jr. D. A. Haessig and B. Friedland. On the modelling and simulation of friction. *Journal of Dynamic Systems, Measurement, and Control*, 113(3):354–362, 1991.

[30] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[31] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994.

[32] D. P. Hess and A. Soom. Friction at a lubricated line contact operating at oscillating sliding velocities. *Journal of Tribology*, 112(1):147–152, 1990.

[33] F. C. Hoppensteadt and E. M. Izhikevich. *Weakly connected neural networks*. Springer, New York, 1997.

[34] A. Hornstein and U. Parlitz. Bias reduction for time series models based on support vector regression. *International Journal of Bifurcation and Chaos*, 14(6):1947–1956, 2004.

[35] B. Hunt, E. Ott, and J. Yorke. Differentiable generalized synchronization of chaos. *Physical Review E*, 55(4):4029–4034, 1997.

[36] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001.

[37] H. Jaeger. Short term memory in echo state networks. GMD Report 152, German National Research Center for Information Technology, 2001.

[38] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[39] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

[40] D. Karnopp. Computer simulation of slip–stick friction in machnical dynamic systems. *Journal of Dynamic Systems, Measurement, and Control*, 107(1):100–103, 1985.

[41] L. Kocarev and U. Parlitz. General approach for chaotic synchronization with applications to communication. *Physical Review Letters*, 74(25):5028–5031, 1995.

[42] L. Kocarev and Parlitz U. Generalized synchronization, predictability, and equivalence of unidirectionally coupled dynamical systems. *Physical Review Letters*, 76(11):1816–1819, 1996.

[43] M. Korenberg, S. A. Billings, Y. P. Liu, and P. J. McIlroy. Orthogonal parameter estimation for non–linear stochastic systems. *International Journal of Control*, 48(1):193–210, 1988.

[44] M. J. Korenberg. A robust orthogonal algorithm for system identification and time–series analysis. *Biological Cybernetics*, 60:267–276, 1989.

[45] V. Lampaert, F. Al-Bender, and J. Swevers. A generalized maxwell–slip friction model appropriate for control purposes. In *Porceedings of the International Conference on Physics and Control*, 2003.

[46] V. Lampaert, F. Al-Bender, and J. Swevers. Experimental characterization of dry friction at low velocities on a developed tribometer setup for macroscopic measurements. *Tribology Letters*, 16(1):95–105, 2004.

[47] V. Lampaert, J. Swevers, and F. Al-Bender. Modification of the leuven integrated friction model structure. *IEEE Transactions on Automatic Control*, 47(4):683–687, 2002.

[48] W. Maass, T. Natschläger, and H. Markram. Real–time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[49] O. Nelles. *Nonlinear System Identification*. Springer, 2001.

[50] H. Olsson. *Control Systems with Friction*. PhD thesis, Lund Institute of Technology, University of Lund, 1996.

[51] H. Olsson, K. J. Åström, C. Canudas de Wit, M. Grävert, and P. Lischinsky. Friction models and friction compensation. *European Journal of Control*, 29(4):176–195, 1998.

[52] U. Parlitz. Common dynamical features of periodically driven strictly dissipative oscillations. *International Journal of Bifurcation and Chaos*, 3(3):703–715, 1993.

[53] U. Parlitz. Synchronization of uni-directionally coupled chaotic dynamical systems (habil. thesis), 1999.

[54] U. Parlitz and A. Hornstein. Detecting generalized synchronization from time series. In D. H. van Campen, M. D. Lazurko, and W. P. J. M. van der Oever, editors, *Fifth EUROMECH Nonlinear Dynamics Conference ENOC-2005*, 09-224, pages 1174–1181. Eindhoven University of Technology, 2005.

[55] U. Parlitz, A. Hornstein, D. Engster, F. Al-Bender, V. Lampaert, T. Tjahjowidodo, S. D. Fassois, D. Rizos, C. X. Wong, K. Worder, and G. Manson. Identification of pre–sliding friction dynamics. *CHAOS*, 14(2):420–430, 2004.

[56] U. Parlitz and L. Kocarev. *Handbook of Chaos Control*, chapter Synchronization of Chaotic Systems, pages 271–303. Wiley–VCH, Weinheim, Germany, 1998.

[57] U. Parlitz and W. Lauterborn. Superstructure in the bifurcation set of the duffing equation. *Physics Letters*, 107A(8):351–355, 1985.

[58] L. M. Pecora and T. L. Carroll. Synchronization in chaotic systems. *Physical Review Letters*, 64(8):821–824, 1990.

[59] L. M. Pecora, T. L. Carroll, and J. F. Heagy. Statistics for mathematical properties of maps between time series embeddings. *Physical Review E*, 52(4):3420–3439, 1995.

[60] A. S. Pikovsky. On the interaction of strange attractors. *Zeitschrift für Physik B – Condensed Matter*, 55:149–154, 1984.

[61] A. Pikowsky, M. Rosenblum, and J. Kurths. *Synchronization, A universal concept in nonlinear sciences*. Cambridge University Press, 2001.

[62] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C (Second Edition)*. Cambridge University Press, 1992.

[63] K. Pyragas. Weak and strong synchronization of chaos. *Physical Review E*, 54(5):R4508–R4511, 1996.

[64] E. Rabinowicz. The nature of the static and kinetic coefficients of friction. *Journal of Applied Physics*, 22(11):1373–1379, 1951.

[65] N. F. Rulkov, M. M. Sushchik, and L. S. Tsimring. Generalized synchronization of chaos in directionally coupled chaotic systems. *Physical Review E*, 51(2):980–994, 1995.

[66] R. Stribeck. Die wesentlichen Eigenschaften der Gleit– und Rollenlager – the key qualities of sliding and roller bearings. *Zeitschrift des Vereines Deutscher Ingenieure*, 46(38,39):1342–1348, 1432–1437, 1902.

[67] J. Swevers, F. Al-Bender, C. G. Ganseman, and T. Prajogo. An integrated friction model structure with improved presliding behavior for accurate friction compensation. *IEEE Transactions on Automatic Control*, 45(4):675–686, 2000.

[68] A. C. Tsoi and Back A. D. Locally recurrent globally feedforward networks: A critical review of architectures. *IEEE Transactions on Neural Networks*, 5(2):229–239, 1994.

[69] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[70] K. Worden, C. X. Wong, U. Parlitz, A. Hornstein, D. Engster, T. Tjahjowidodo, F. Al-Bender, S. D. Fassois, and D. D. Rizos. Identification of pre–sliding and sliding friction. In *European Nonlinear Oscillation Conference (ENOC-2005)*, number 17-408, pages 1985–1995, 2005.

[71] G. L. Zheng and S. A. Billings. Radial basis function network configuration using mutual information and the orthogonal least squares algorithm. *Neural Networks*, 9(9):1619–1637, 1996.

[72] Q. M. Zhu and S. A. Billings. Parameter estimation for stochastic nonlinear rational models. *International Journal of Control*, 57(2):309–333, 1993.

[73] Q. M. Zhu and S. A. Billings. Fast orthogonal identification of nonlinear stochastic models and radial basis function neural networks. *International Journal of Control*, 64(5):871–886, 96.

Acknowledgments

I thank Prof. Dr. Ulrich Parlitz for his guidance and support. With his numerous and valuable ideas and suggestions he contributed a great deal to this thesis. I thank Prof. Dr. Werner Lauterborn for letting me being part of the Third Physical Institute, in which I always felt at home. Of course I am grateful to the VW-Stiftung for financial support and for the great opportunity to work in an international project. In this context I thank Prof. Dr. Fahrid Al-Bender, Prof. Dr. Spilios Fassois, Prof. Dr. Keith Worden, Tegoeh Tjahjowidodo, Demos Rizos, and Chian Wong for their cooperation in this project. It was a pleasure and fun to be working with them.

I thank my parents, Lioudmila and Viktor Hornstein, for supporting me and standing by my side at all times. I thank Ramona Emmrich who gives me strength and makes my life so much richer. I thank all my friends for moral support. Especially, I thank Sebastian Grieb and Bernhard Wolfrum, who alway tried to motivate me through the hard times. I thank Jörg Dittmar, David Engster, Elke Hanke, Philipp Koch, Immo Wedekind, Karsten Peters, and other guys from the Third Physical Institute for stimulating discussions and friendly chats.

Lebenslauf

Alexander Hornstein
geboren am 31.05.1973 in Leninabad (Tadschikistan, ehem. UdSSR)
deutsche Staatsangehörigkeit

1981	Umzug nach Deutschland, Oldenburg in Niedersachsen
1981–1985	Grundschule Nadorst, Oldenburg
1985–1987	Orientierungsstufe Flötenteich, Oldenburg
1987–1994	Altes Gymnasium Oldenburg mit Abschluss Abitur
1994–1995	Zivildienst beim Deutschen Roten Kreuz, Oldenburg
1995–2001	Physikstudium in der Universität Göttingen mit Abschluss Diplom
2001–2005	Promotionsstudium in Physik am Dritten Physikalischen Institut (Universität Göttingen)