Local- and Cluster Weighted Modeling for Prediction and State Estimation of Nonlinear Dynamical Systems

Dissertation

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

vorgelegt von

David Engster
aus Göttingen

Göttingen 2010

D7
Referent: Prof. Dr. Ulrich Parlitz
Korreferent: Prof. Dr. Florentin Wörgötter
Tag der mündlichen Prüfung: 24.8.2010

Summary

This thesis deals with black-box modeling techniques, in particular local models based on nearest neighbors, and Cluster Weighted Models, which combine a stochastic clustering of the input space with a deterministic parametric model in each cluster. Given observations obtained from a dynamical system, often corrupted by noise, those models are then used to predict future states, or to reconstruct the current internal state of the system. The performance of both techniques will be evaluated on various examples, from numerical chaotic oscillators to experimental friction data, where for the latter it is shown that using an ensemble of Cluster Weighted Models enhances the stability of the model output.

Cluster Weighted Models produce a full probability distribution as output. Using the concept of probabilistic scoring, Cluster Weighted Models will be compared against other modeling techniques which produce a probabilistic output, showing that they perform well in the stochastic as well as the deterministic regime. Also, several regularization techniques and their effect on the model's score will be discussed. Cluster Weighted Models will then be used for the concept of Active Learning, where one strives to actively choose data points for measurements which yield the most information. The models will be used to find points with a high information gain, also in terms of detecting interesting features like extremal values.

Lastly, tackling the problem of long term prediction, a new method based on nearest neighbors will be introduced, which tries to maximize the overlap between the original and the model's attractor. This method is then used to fit the coefficients of a system of ordinary differential equations, targeting numerical as well as experimental systems.

Contents

Notation

If not explicitly noted otherwise in the text, the following notation will be used:

x	scalar value, $\in \mathbb{R}$
$\mathbf{x}$	vector, $\in \mathbb{R}^d$
$\mathbf{X}$	matrix, $\in \mathbb{R}^{m \times n}$
$\mathbf{X}^\dagger$	pseudoinverse of matrix $\mathbf{X}$
$\hat{x}$	an estimate of x
$\bar{x}$	the mean of a set $\{x_i\}_1^N$
$E[Z\|\mathbf{\Psi}]$	conditional expectation value of the r.v. Z, dependent on $\mathbf{\Psi}$
$< Z\|\mathbf{\Psi} >$	dto., short form
$< Z >_m$	cluster weighed expectation value for cluster with index m
$p(Z)$	probability distribution of the r.v. Z
$p(Z\|x)$	conditional probability, short form of $p(Z\|X = x)$
$\mathbb{P}(A)$	probability of observation A

Abbrevations

CWM	Cluster Weighted Model/Modeling
KDE	Kernel Density Estimator
IPF	Interacting Particle Filter
SNR	Signal-to-Noise Ratio
QbC	Query by Committee
ODE	Ordinary Differential Equation
(N)MSE	(Normalized) Mean Squared Error
GMM	Gaussian Mixture Model
RBF	Radial Basis Function
(T)PCR	(Truncated) Principal Component Regression
PCTR	Principal Components Threshold Regression
PCA	Principal Component Analysis
ML	Maximum Likelihood
EM	Expectation Maximization (algorithm)
(LOO)CV	(Leave-one-out) Cross Validation

Introduction

In many modern fields of science, one is confronted with huge amounts of data and little to no information about the underlying process which generated it. Even if one happens to have knowledge of the generating process, it might be too complex to derive an analytical model, or the model cannot be applied with the amount of data available. Be it the biologist struggling with DNA sequences [7], or the physicist trying to make sense out of the data coming from CERN's massive detector arrays [49], black-box modeling can often assist in extracting features, separating important from unimportant data, reducing measurement noise, or the classification of data.

There is a vast amount of literature regarding black-box modeling, targeted at different disciplines and hence often using different vocabulary: Machine Learning, Supervised and Unsupervised Learning, Statistical Learning Theory, Function Approximation, Density Estimation — it is difficult to get an overview and correctly classify the different theories and algorithms. This thesis will naturally use the view of a physicist: our "black box" stands for an experiment, depending on some input terms and producing some kind of output. One can think of the model as a function $f(\mathbf{x}) = y$, with $\mathbf{x}$ being the input data and y the outcome of the experiment. The underlying process is usually not known, and as little information as possible should be needed for modeling. Thus, the model $f(\cdot)$ should ideally be generated entirely through the data itself.

One can divide black-box modeling very roughly into two categories: the local and the global approach [52]. The former tries to build models only in a neighborhood of a certain query input vector $\mathbf{x}_q$, i.e., the model output $y_q = f(\mathbf{x}_q)$ only depends on the neighborhood of $\mathbf{x}_q$ in input space and ignores everything else. The global approach however depends on an often time consuming training process, involving the whole data set, to find a general model which is able to map the dependencies between input and output variables.

For local modeling, the question arises how one should define neighborhoods. The most common approach is based on *nearest neighbors* of the query point $\mathbf{x}_q$. Despite the quite simple architecture, this approach is among the most powerful black-box modeling techniques, but the problems lie in the details: how to find a good size of the neighborhood, which metric should be used, how should the neighbors be weighted and what kind of functional model should be employed.

Technically a global modeling technique, but somewhat adhering to the local paradigm, in most of this thesis we will use Cluster Weighted Modeling (CWM), first described by Gershenfeld et al. in [30]. The locality is provided by a clustering technique, using Gaussian Mixture Models [51], but combined with a parametric function for modeling functional dependence in each cluster. In the end, we have a global model, which can be written down in closed form, but still have locality due to the clustering of the input space.

Another important feature which make Cluster Weighted Models stand out, is that their output comes in the form of a full probability distribution, whereas many other models only provide a scalar output, maybe combined with a variance. There are of course other modeling algorithms which provide a probabilistic output, like particle filters [15] or kernel density estimators [73], and we will compare their performance against CWMs. This leads to the question how one should measure the performance of a distribution, in terms of its prediction quality. The usual measure for a model's performance, the mean squared error, does not account for full probability distributions, but only depends on their expectation value. Therefore, the notion of *scores* will be introduced, and according to such a score the comparison will be done by performing state estimations for different numerical and experimental systems.

For experiments where generating data is costly, be it in terms of time or money, it is important to wisely choose for which input values one would like to measure the experimental outcome. When working on such an experiment, one usually strives to measure those parameters which one expects to yield the most information. The problem here is how to define and find those points; this is the field of *Experimental Design* and *Active Learning*. Cluster Weighted Models will be used to find points with a high information gain, also in terms of detecting interesting features like extremal values.

Lastly, the problem of long term prediction will be discussed. It is a well known problem that models which perform good for short predictions can show vastly different behavior on longer time scales [38]. However, sometimes one is not primarily interested in a good short-term prediction, but would like to have a model which qualitatively shows a behavior similar to the original system when the model is freely iterated for many steps. A new local method based on nearest neighbors is introduced, which tries to maximize the overlap between the original and the model's attractor. This method is then used to fit the coefficients of an ODE system.

The structure of this thesis is as follows:

First, a few basic notions of nonlinear dynamics will be introduced which are needed for the following chapters. Chapter two will first deal with modeling in general, before turning to local modeling based on nearest neighbors and Cluster Weighted Modeling, with a focus on the latter. Several numerical and experimental systems will be used to demonstrate how both modeling techniques perform for various purposes like noise reduction, signal reconstruction and the modeling of friction data. For the latter, the usage of CWM ensembles is discussed to enhance

the stability of the model's output.

Chapter three will then turn to the probabilistic evaluation of Cluster Weighted Models. The concept of scores is introduced and CWMs are compared against several other probabilistic modeling techniques, again using numerical as well as experimental data.

Since CWMs are prone to overfitting, chapter four will deal with different regularization techniques and their effect on the resulting score, demonstrated on numerical data of the Chua oscillator. In the following chapter six, we will give a short introduction on Design of Experiments and Active Learning, and then discuss how CWMs can be used in this context. The final chapter will deal with the method of attractor comparison for creating models with respect to long-term behavior, which again will be demonstrated on numerical and experimental data.

The thesis will close with a summary and outlook to possible future developments.

Chapter 1

Basic notions of nonlinear dynamics

This chapter will define the basic terms of nonlinear dynamical systems used in the following chapters, but necessarily in a very brief way. For further details, the reader is referred to texts like [3, 58, 66, 75].

1.1 Dynamical systems

In general, a *dynamical system* is a set of objects with continuously or discretely observable states, which change with time according to certain rules. The objects are described through state vectors $\mathbf{x} \in M \subset \mathbb{R}^d$ in a finite dimensional space. The dynamical system is defined through a continuous mapping

$$\Phi : \mathbb{K} \times M \to M \tag{1.1}$$

with the following properties [43]

$$\Phi(0, \mathbf{x}) = \mathbf{x} \text{ for all } \mathbf{x} \in M\,, \tag{1.2}$$

$$\Phi(d, \Phi(t, \mathbf{x})) = \Phi(t + d, \mathbf{x}) \text{ for all } d, t \in \mathbb{K}, \mathbf{x} \in M\,. \tag{1.3}$$

As one can see from those properties, the mapping Φ defines the evolution of a state $\mathbf{x}$, with the parameter t denoting the time. This parameter may either be discrete ($\mathbb{K} = \mathbb{Z}$) or continuous ($\mathbb{K} = \mathbb{R}$), leading to discrete or continuous dynamical systems, resp.; in the latter case, the mapping Φ is also called the *flow* of the dynamical system. If Φ is non-invertible, the parameter t must be restricted to positive values.

Following the time evolution of a certain state $\mathbf{x} \in M$ yields a *trajectory* or *orbit* in state space, which can be described through a mapping

$$\begin{aligned} \alpha_{\mathbf{x}} : \mathbb{K} &\to M \\ t &\mapsto \Phi(t, \mathbf{x})\,. \end{aligned} \tag{1.4}$$

Since the trajectory is uniquely defined through its initial state, it is impossible for two different trajectories to cross each other in state space.

We will now look at an *autonomous* system, where the time derivative is given by a continuously differentiable vector field $F : M \to \mathbb{R}^d$:

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x})\,. \tag{1.5}$$

This defines an autonomous system of first order ordinary differential equations, which is general in the sense that any system of higher order can be reduced to a system of first order equations by introducing additional variables. Similarly, by adding the variable $x_{d+1} = t$ and the trivial differential equation $\dot{x}_{d+1} = 1$, one can transform every non-autonomous system into an autonomous one. A trajectory of the form (1.4) is a solution to this ODE system.

For continuously differentiable vector fields (which can in fact be further reduced to fields which fulfill the Lipschitz condition), the solution of this autonomous system is uniquely defined through the initial values. The flow Φ can be obtained by integrating the differential equation over time t, from which it follows that this flow is always invertible.

Systems with discrete time are described through a difference equation

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1})\,, \tag{1.6}$$

with $\mathbf{x}_n$ being the state of the system for time index $n \in \mathbb{Z}$. Again, the time evolution is uniquely defined through the initial states, thereby making (1.6) a dynamical system, with the flow coinciding with the mapping itself.

1.2 Lyapunov exponents and chaotic motion

The Lyapunov exponents describe the exponential divergence or convergence of closely adjacent trajectories; if they diverge exponentially, one says the system has a *sensitive dependence on the initial conditions* and thus is *chaotic*.

To define the Lyapunov exponent, we start with two different initial states $\mathbf{x}_0$ and $\mathbf{x}_0 + \delta\mathbf{x}_0$, with $\delta\mathbf{x}_0$ being an infinitesimal perturbation, and observe how the trajectories beginning at those states evolve under the flow, and in particular the absolute value of the perturbation. With a dynamical system of the form (1.5), the time derivative is given by

$$\frac{d(\mathbf{x} + \delta\mathbf{x})}{dt} = \mathbf{F}(\mathbf{x} + \delta\mathbf{x})\,. \tag{1.7}$$

Linearization in the neighborhood of $\mathbf{x}$ yields

$$\begin{aligned} \frac{d\mathbf{x}}{dt} + \frac{\delta\mathbf{x}}{dt} &= \mathbf{F}(\mathbf{x}) + \frac{d\mathbf{F}}{d\mathbf{x}} \cdot \delta\mathbf{x} \\ \Rightarrow \quad \delta\dot{\mathbf{x}} &= \mathbf{J}(\mathbf{x}) \cdot \delta\mathbf{x}\,, \end{aligned} \tag{1.8}$$

with $\mathbf{J}(\mathbf{x}) = d\mathbf{F}/d\mathbf{x}$ being the Jacobian of the ODE system. The time evolution of the perturbation can be obtained through the transfer matrix $\mathbf{U}^t$, which solves $\dot{\mathbf{U}} = \mathbf{J}\mathbf{U}$ with $\mathbf{U}^0 = \mathbf{I}$, hence

$$\delta\mathbf{x}_t = \mathbf{U}^t \delta\mathbf{x}_0 \,. \tag{1.9}$$

The Lyapunov-Exponent in the direction of $\mathbf{u}_0 = \delta\mathbf{x}_0/\|\delta\mathbf{x}_0\|$ is now given by

$$\lambda(\mathbf{x}_0, \delta\mathbf{x}_0) = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\|\delta\mathbf{x}_t\|}{\|\delta\mathbf{x}_0\|} = \lim_{t\to\infty} \frac{1}{t} \ln \|\mathbf{U}^t(\mathbf{x}_0)\mathbf{u}_0\| \,. \tag{1.10}$$

For discrete mappings of the form (1.6), the flow coincides with the mapping itself, hence one can calculate the time evolution of the perturbation directly through its Jacobian.

Therefore, in a d-dimensional phase space, there exist d Lyapunov exponents, describing the time evolution of the perturbation in the different directions. For ergodic systems, they are invariant towards their initial conditions $\mathbf{x}_0$ and $\delta\mathbf{x}_0$. Continuous systems always have one Lyapunov exponent equal zero in the tangential direction of the trajectory, except for trajectories tending to a stable fixed point, where all Lyapunov exponents are negative. Chaotic motion is defined through (at least) one positive Lyapunov exponent, thus calculating the largest Lyapunov exponent allows to definitely state whether a system is chaotic. However, getting a robust estimation of the Lyapunov exponents can be a fairly difficult task, especially when the system equations are not known and only an observed time series is available; for details on the various available methods the reader is referred to [27, 59].

1.3 Attractors

Regarding the trajectories (1.4), one is mostly interested in their asymptotic behavior with $t \to \infty$. In a conservative dynamical system, the flow is volume preserving, meaning that during time evolution, a set of states will always occupy the same volume in phase space. In dissipative systems however, this volume shrinks under the flow, i.e., for continuous systems the divergence of the vector field is negative

$$\boldsymbol{\nabla} \cdot \mathbf{F} < 0 \,, \tag{1.11}$$

and for dissipative discrete systems, the determinant of the Jacobian of $\mathbf{f}$ is smaller than one.

Under the influence of the flow, it is typical for dissipative systems that the initial volume of a set of states approaches a compact subset $A \subset \mathbb{R}^n$ in phase space. Because of its "attracting" property, this subset is called an *attractor*, which has the following properties [66, 75]:

- *Attractivity*: It exists a neighborhood U of A such that $\Phi(U, t) \subset U$ for $t > 0$ and which contracts under the flow, so that

$$A = \bigcap_{t>0} \Phi(U, t) \,. \tag{1.12}$$

- *Invariance*: The attractor A is invariant under the flow, i.e., for all $\mathbf{x} \in A$ and $t > 0$ it holds that $\Phi(\mathbf{x}, t) \in A$.
- *Irreducibility*: With growing t and for almost all $\mathbf{x}_0$ it holds that $\Phi(\mathbf{x}_0, t) \in U_\mathbf{a}$ for arbitrary neighborhoods $U_\mathbf{a}$ for all $\mathbf{a} \in A$, meaning that the attractor A cannot be divided into two non-overlapping, closed invariant sets.

The set of all initial values leading to the attractor is called the *basin of attraction*. If the initial value of a trajectory lies within such a basin, it will, after a certain time called the *transient*, lie within the attractor. Such an attractor also exists for chaotic systems; when observing a volume element on such an attractor, it grows exponentially in the direction(s) corresponding to the positive Lyapunov exponent(s), and shrinks or stagnates in the other directions. Since the attractor is a compact set, the volume cannot grow indefinitely, but will be bounded and folded back when reaching the attractor's border. Through this stretching and folding, the attractor shows a self-similar structure on different scales, forming a *fractal set* with a non-integer dimension, which is why they are also called *strange attractors*. While regular attractors like fixed points or limit cycles are smooth manifolds, a strange attractor is a fractal subset of the phase space, which however is usually *embedded* in a lower dimensional manifold. The fractal structure of the attractor is a necessary condition for chaotic dynamics, but it is not sufficient. For instance, in quasiperiodically forced systems, one can have a strange but nonchaotic attractor [58].

1.4 Attractor dimension

In general, the dimension of an attractor can be seen as the amount of information needed to uniquely locate a point on the attractor with a certain accuracy. It is a measure for the number of degrees of freedom and thus for the complexity of the motion on the attractor.

For defining the dimension of a strange attractor with its fractal structure, one has to look at the statistical distribution of the points in phase space. The most well known method is the *box-counting* method, where one divides the phase space into identical boxes with volume ε^d and counts the number of boxes $N(\varepsilon)$ containing points of the attractor. In the ideal case, this yields a correlation of the form

$$N(\varepsilon) \sim \varepsilon^{-D_0}\,, \tag{1.13}$$

with the exponent D_0 being the *box-counting dimension*

$$D_0 = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(1/\varepsilon)}\,. \tag{1.14}$$

However, the dimension D_0 does not account for the fact that trajectories might reach certain regions of the attractor much more frequently than others (which is in

fact a typical behavior of chaotic motion, due to unstable periodic orbits embedded in the attractor). This can be described through the *natural measure*

$$\mu(N_i) = \lim_{T\to\infty} \frac{\eta(N_i, T)}{T}\,, \tag{1.15}$$

where $\eta(N_i, T)$ is the amount of time a trajectory stays in the box N_i during time $0 \leq t \leq T$. This can be interpreted as the probability that a point can be found in box N_i. One can now define the *information dimension* through

$$D_1 = \lim_{\varepsilon\to 0} \frac{\sum_{i=1}^{N(\varepsilon)} \mu(N_i) \ln \mu(N_i)}{\ln(\varepsilon)} = \lim_{\varepsilon\to 0} \frac{I(\varepsilon)}{\ln(1/\varepsilon)}\,, \tag{1.16}$$

with $I(\varepsilon) = -\sum_{i=1}^{N(\varepsilon)} \mu(N_i) \ln \mu(N_i)$ being the Shannon entropy, which can be seen as a measure for the amount of information being gained when using an edge length ε, hence the information dimension is the fraction between information gain and edge length used for dividing the phase space.

Both dimensions can be derived as special cases from Renyi's *generalized dimension*

$$D_q = \lim_{\varepsilon\to 0} \frac{1}{q-1} \cdot \frac{\ln\left(\sum_{i=1}^{N(\varepsilon)} (\mu(N_i))^q\right)}{\ln(\varepsilon)}\,, \tag{1.17}$$

with $q \in \mathbb{R}$, thus leading to an infinite number of dimensions, holding the relation $D_q \leq D_p$ for $q \geq p$. The box-counting dimension (1.14) can be obtained for $q = 0$, whereas the information dimension (1.16) can be derived with $q \to 1$ and employing L'Hôspital's rule. It shall be noted that the often used *correlation dimension*, which describes the spatial correlation of point pairs on the attractor, is an approximation for D_2. This dimension is quite popular since it can be calculated fairly easily through nearest neighbor algorithms [3].

1.5 Attractor reconstruction

The theory described so far always operated in phase space. In practice however, we usually only have access to one (or maybe several) scalar time series recorded from the dynamical system. Therefore, what is needed is a unique mapping from the one-dimensional time series to states in phase space.

We shall first restrict ourselves to continuous systems of the form (1.5), with the dynamics taking place in a manifold $S \subset \mathbb{R}^k$ with dimensionality $d < k$. Let us assume we can perform n independent measurements $u_1, \ldots, u_n$ for a certain time, then we can describe this through a *measurement function* $\mathbf{h}(\mathbf{x})$, mapping states from S to $\mathbb{R}^n$:

$$\begin{aligned} \mathbf{h}: \quad & S \subset \mathbb{R}^k \to \mathbb{R}^n \\ & \mathbf{x} \mapsto \mathbf{h}(\mathbf{x}) = (u_1, \ldots, u_n)\,. \end{aligned} \tag{1.18}$$

To reconstruct the system's original dynamics, its attractor $A \subset S$ must be invariant under this measurement function, meaning that two measurements $\mathbf{h}(\mathbf{x}_i)$ and $\mathbf{h}(\mathbf{x}_j)$ with $i \neq j$ may only be identical if and only if $\mathbf{x}_i = \mathbf{x}_j$. Additionally, we must make sure that the attractor's differential properties are retained, which is called an *embedding* of the attractor, effectively making this a nonlinear coordinate transformation.

The question under which preconditions (1.18) is an embedding was first answered for smooth manifolds by Whitney's embedding theorem in 1936, and was later extended in 1991 by Sauer et al. for compact subsets with a fractal structure [67]. In summary, it says that if A is compact in $\mathbb{R}^k$ with a box-counting dimension d, and Φ is a flow on $\mathbb{R}^k$ and n is a natural number with $n > 2d$, then almost all continuously differentiable mappings $\mathbf{h}$ are an embedding from A to $\mathbb{R}^n$, with the latter also being called *reconstruction space*.

1.5.1 Takens' theorem and delay embedding

In practice, the preconditions described in the previous chapter are very difficult or impossible to satisfy, since one has to simultaneously measure $n > 2d$ different properties. However, Takens' theorem [76] provides the striking result that the continuous measurement of only one property is usually sufficient to reconstruct the systems' attractor.

This property is measured by applying the continuous mapping $h(\mathbf{x})$, relating each state in phase space with one scalar value s_t for time t. Using the flow Φ_t, the *delay coordinate mapping* is defined through

$$\mathbf{F}(h, \Phi, \tau)(\mathbf{x}) = \big(h(\mathbf{x}), h(\Phi_\tau(\mathbf{x})), h(\Phi_{2\tau}(\mathbf{x})), \ldots, h(\Phi_{(n-1)\tau}(\mathbf{x}))\big) \ , \tag{1.19}$$

with τ being the *delay time*.

Takens' theorem [76] and its extension by Sauer et al. [67] states that given a continuous dynamical system with a compact invariant smooth manifold A with box-counting dimension d, that if $n > 2d$, the delay coordinate mapping $\mathbf{F}$ is an embedding for almost all (h, τ), given that A

- contains no periodic orbits of period τ or 2τ,
- contains only a finite number of equilibria,
- contains only a finite number of periodic orbits of period $p\tau$ with $3 \leq p < n$, and the linearizations of those orbits have distinct eigenvalues.

Note that the theorem only holds for continuous measurement functions with noiseless data, whereas in practice we can only perform discrete measurements, which will also be corrupted by measurement noise. Though Takens' theorem in theory does not apply for this case, it is still usually possible to obtain an embedding, provided

that the sampling time is not too large and a proper delay time is used, so that the attractor gets unfolded. For discrete values, the delay coordinate mapping is reduced to the construction of *delay vectors*

$$\mathbf{x}_{t-\tau(d-1)} = (s_t, s_{t-\tau}, \ldots, s_{n-(d-1)\tau}), \quad t = \tau(d-1)+1, \ldots, n \tag{1.20}$$

with τ being a multiple of the sampling time.

Chapter 2

Modeling based on nearest neighbors and weighted clusters

2.1 The modeling problem

Given a data set of N pairs of points

$$\mathbf{\Omega} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \tag{2.1}$$

with vector inputs $\mathbf{x}_i \in \mathbb{R}^d$ and corresponding scalar outputs $y_i \in \mathbb{R}$ of an unknown system, the nonlinear modeling problem is to find an estimate $\hat{y}$ of the system output for a new vector input $\mathbf{q} \notin \mathbf{\Omega}$, which is often simply called the *query*.

A different and perhaps more familiar approach arises from the statistical viewpoint where one tries to find a good approximation for the *regression* $\mathrm{E}[Y \mid \mathbf{X}]$. Here the pairs $(\mathbf{x}_i, y_i)$ are seen as realizations of the random variables $\mathbf{X}$ and Y, where Y and $\mathbf{X}$ are drawn from an unknown joint probability P. The regression $\mathrm{E}[Y \mid \mathbf{X}]$ is the random variable which gives the conditional expectation $m(\mathbf{x}) \equiv \mathrm{E}[Y \mid \mathbf{X} = \mathbf{x}]$. It is the best approximation for the output values y_i in a least squares sense [28].

2.2 Time series prediction

We now want to specialize the described modeling problem to the case of time series prediction, where given a series $s_1, \ldots, s_N$, $s_i \in \mathbb{R}$, the model should be able to predict the p next time steps $s_{N+1}, \ldots, s_{N+p}$. We assume that the time series is the result of a measurement with a certain sampling frequency, performed on a nonlinear dynamical system with a deterministic time evolution. In the case of chaotic systems, even the exact knowledge of the underlying system does not allow the prediction of an arbitrary number of time steps due to the sensitivity on the initial conditions, i.e. the *prediction horizon* is limited. Additionally, if the time series was measured in an experiment, it will always be corrupted by some measurement noise.

The input vectors $\mathbf{x}_i \in \mathbb{R}^d$ for the modeling algorithm can be obtained by reconstructing the attractor of the underlying dynamical system. As described in section 1.5.1, this can be accomplished by using a delay embedding of the time series with proper dimension d and delay τ, leading to the input vectors

$$\mathbf{x}_t = \left(s_t, s_{t-\tau}, \ldots, s_{t-(d-1)\tau}\right), \tag{2.2}$$

with t ranging from $(d-1)\tau + 1$ to $N - p$ (considering that we predict p steps into the future). It is also possible to choose a *non-uniform* embedding, which instead of the fixed delay τ allows varying delays τ_i, $i = 1, \ldots, d-1$ between the components of the input vector [42].

To predict one step ahead, the corresponding output is given by $y_t = s_{t+1}$. For a further prediction of the next p steps, one can add the model output $\hat{s}_{N+1}$ to the given time series and repeat the modeling procedure until $\hat{s}_{N+p}$ is obtained, leading to an *iterated prediction*. However, if one is only interested in the model output $\hat{s}_{N+p}$, it is possible to do a *direct prediction* by using $y_t = s_{t+p}$ for the corresponding outputs. With iterated prediction, the errors of the model output accumulate, whereas for direct prediction the system output becomes more complex and is therefore more difficult to model correctly, especially for chaotic systems. There has been much discussion regarding whether iterated or direct prediction is the better choice [17]. This question cannot be answered in general, as it depends on the complexity of the system, the step size p and the sampling time. However, for chaotic systems iterated prediction has often shown to be superior in practice [52].

2.2.1 Cross prediction

A more general case is the *cross prediction* of a time series, where one or more input time series $s^{(1)}_{1\ldots N}, \ldots, s^{(n)}_{1\ldots N}$ are given and one output time series $u_{1\ldots N}$ has to be predicted. The previous case of time series prediction can be seen as a special case of cross prediction, where the output time series is simply the input time series shifted p steps into the future. In the more general form with several different input time series, the construction of the input vector becomes more complicated. For every given input time series, a delay embedding must be performed. The delay vectors can then be concatenated to form the input vectors

$$\mathbf{x}_t = \left(s^{(1)}_t, s^{(1)}_{t-\tau_1}, \ldots, s^{(1)}_{t-(d_1-1)\tau_1}, s^{(2)}_t, s^{(2)}_{t-\tau_2}, \ldots, s^{(n)}_{t-(d_n-1)\tau_n}\right) \tag{2.3}$$

for the modeling algorithm.

However, even if the output of the dynamical system is completely determined by the input time series, in some cases the modeling problem becomes much easier if past values of the output time series are included in the input vector, effectively introducing a feedback into the modeling procedure. This can lead to an improvement of the prediction, but may lead to stability problems if the model is iterated over

several time steps since the errors in the prediction accumulate. A practical example of such a cross prediction with feedback is shown in section 2.9 with the modeling of friction phenomena.

2.3 Bias/Variance trade-off

For finding the mapping $y_i = f(\mathbf{x}_i)$ between dependent and independent variables, one has to consider that the model should not only be able to describe the given realization, but ideally also every other realization which is drawn from the joint probability $P(y, \mathbf{x})$. Even if one finds a perfect approximation for the regression $\mathrm{E}[y \,|\, \mathbf{x}]$ *for one particular realization*, this does not in general lead to a model which will perform well on new data sets. In other words, the model should have the ability to *generalize* with respect to new data.

Given a realization $\mathbf{\Omega} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ of the data generating process, the model based on this particular realization is written as $f(\mathbf{x}; \mathbf{\Omega})$. The expectation value of the squared error, given this realization, can be split into two parts

$$\mathrm{E}\left[(y - f(\mathbf{x}; \mathbf{\Omega}))^2 \,|\, \mathbf{x}, \mathbf{\Omega}\right] = \underbrace{E\left[(y - \mathrm{E}[y \,|\, \mathbf{x}])^2 \,|\, \mathbf{x}, \mathbf{\Omega}\right]}_{\text{variance}_y} + \underbrace{(f(\mathbf{x}; \mathbf{\Omega}) - \mathrm{E}[y \,|\, \mathbf{x}])^2}_{\text{model error}}, \quad (2.4)$$

where the expectation is taken with respect to the joint probability P. The first term $E\left[(y - \mathrm{E}[y \,|\, \mathbf{x}])^2 \,|\, \mathbf{x}, \mathbf{\Omega}\right]$ is the variance of y for a given $\mathbf{x}$ and is independent from the realization $\mathbf{\Omega}$ and the model $f(\mathbf{x})$. Therefore, the variance is a lower bound on the expectation value of the squared error, although it is of course through interpolation always possible to get a zero squared error for one particular realization. However, a model which simply interpolates the data will on other realizations lead to a larger squared error than the regression $\mathrm{E}[y \,|\, \mathbf{x}]$, as it also tries to model realization dependent features. This effect is called *overfitting* and can be avoided by introducing a bias which limits the variance of the model.

To see the connection between bias and variance, one has to examine the second term $(f(\mathbf{x}; \mathbf{\Omega}) - \mathrm{E}[y \,|\, \mathbf{x}])^2$, which describes the actual model error. It may be that for the particular realization $\mathbf{\Omega}$ our model perfectly approximates the regression $\mathrm{E}[y \,|\, \mathbf{x}]$. However, the model might vary strongly depending on the given realization, or it might on average over all possible realizations be a bad approximator for the regression, making the model $f(\mathbf{x}; \mathbf{\Omega})$ an unreliable predictor of y. Since we want to have a model which has the ability to generalize, we must look at the expectation value of $(f(\mathbf{x}; \mathbf{\Omega}) - \mathrm{E}[y \,|\, \mathbf{x}])^2$ over all possible realizations, in the following denoted by $E_{\mathbf{\Omega}}[\cdot]$. This term can again be split into two parts [28], the squared bias and the

variance:

$$\begin{aligned} \mathrm{E}_{\boldsymbol{\Omega}}\left[(f(\mathbf{x};\boldsymbol{\Omega}) - \mathrm{E}\left[y \,|\, \mathbf{x}\right])^2\right] = \underbrace{\left(\mathrm{E}_{\boldsymbol{\Omega}}\left[f(\mathbf{x};\boldsymbol{\Omega})\right] - \mathrm{E}\left[y \,|\, \mathbf{x}\right]\right)^2}_{\text{bias}^2} \\ + \underbrace{E_{\boldsymbol{\Omega}}\left[\left(f(\mathbf{x};\boldsymbol{\Omega}) - \mathrm{E}_{\boldsymbol{\Omega}}\left[f(\mathbf{x};\boldsymbol{\Omega})\right]\right)^2\right]}_{\text{variance}_f} . \end{aligned} \tag{2.5}$$

The bias is the expectation value for the deviation between model output and the regression over all possible realizations. Therefore, a model with high bias will give similar results for different realizations, whereas a model with low bias and high variance can lead to very different model outputs and has a greater chance of overfitting. If the bias is zero we obtain $\mathrm{E}_{\boldsymbol{\Omega}}\left[f(\mathbf{x};\boldsymbol{\Omega})\right] = \mathrm{E}\left[y \,|\, \mathbf{x}\right]$, i.e., our model is *on average* equal to the regression. However, from this we cannot conclude that *for one particular realization* the model $f(\mathbf{x};\boldsymbol{\Omega})$ is a good approximation for the regression $\mathrm{E}\left[y \,|\, \mathbf{x}\right]$. A low bias typically comes with a large variance, making the model unreliable and leading to overfitting and therefore to an increase of the model error. This fact is known as the *bias variance dilemma* [28], which states that it is often not possible to have a low bias and a low variance at the same time. Instead, one has to find a good trade-off between these two.

2.4 Model validation

As just described in section 2.3, it usually not possible to generate a model which offers low bias and low variance at the same time. The most common procedure for finding a good trade-off between bias and variance lies in the training of the model by using *cross validation*. Here the data set is split into two parts, the

- training set, used for training and the
- test set, used for validating the model.

The usual iterative procedure is to switch between training and validation using the training and test data, respectively. With further training, the model error on the training set will usually monotonically decrease as the model is able to describe more and more features of the training data. At some point however, the model begins to overfit on the training data and the error on the test data will then begin to increase. Therefore, the minimum of the test error yields the optimal set of parameters and leads to a model which has still the ability to generalize. For comparing the performance between different models, another test data set is necessary which is only used to calculate one final error measure after the model training is completely finished. In this case, the test set for the cross validation is sometimes referred to as the *validation set* to distinguish it from the test data set for the model comparison.

The drawback of cross validation is the reduced number of points available for training. Therefore, the possibility remains a better model could have been obtained without cross validation [69]. To minimize this possibility, the size of the test set should be chosen as small as possible. This leads to an "extreme" form of the cross validation, the *leave-one-out cross validation* (LOOCV), where the test set is reduced to one single test point. Of course, one has to repeat this validation procedure with enough different test points to get a good estimation of the actual model error. Local models are very well suited for LOOCV, as they are lazy learners which wait with the actual model calculations until they are queried. To implement LOOCV, they simply have to exclude the test point from their set of possible nearest neighbors.

2.4.1 Error measures

The most common choice for calculating the model error is the mean squared error (MSE)

$$\mathrm{MSE}_1 = \frac{1}{|T_{\mathrm{ref}}|} \sum_{t \in T_{ref}} \left(y_t - f^t(\mathbf{x}_t)\right)^2 , \tag{2.6}$$

where $|T_{\mathrm{ref}}|$ is the number of test points and $f^t(\mathbf{x})$ is the model which was constructed without the point $\mathbf{x}_t$.

For time series prediction, the MSE_1 gives the model error for predicting one step ahead in the future, but it is often desirable to have a model which predicts several steps p. As described in section 2.2, this can be achieved by using iterated prediction, where the model is used p times successively. One has to consider however that the model error accumulates during the prediction. Otherwise, when the model is solely validated using the above MSE_1, one will mostly obtain models which are good for one-step but inferior for iterated prediction. Therefore, the MSE should be extended to average the error over p successive steps:

$$\mathrm{MSE}_p = \frac{1}{p|T_{\mathrm{ref}}|} \sum_{t \in T_{ref}} \left[\left(s_{t+1} - f^t(\mathbf{x}_t)\right)^2 + \sum_{i=1}^{p-1} \left(s_{t+i+1} - f^{t+i}(\hat{\mathbf{x}}_{t+i})\right)^2 \right] . \tag{2.7}$$

The first point $\mathbf{x}_t$ is taken from the data set, whereas all further predictions depend on previous model outputs $\hat{\mathbf{x}}_{t+i}$.

When using models based on nearest neighbors, and if the time series is densely sampled, one has to take into account that the nearest neighbors of a test point $\mathbf{x}_t$ will mostly be points which are also close in time, i.e. points which are directly before or after this point on the same trajectory in phase space. Therefore, it is necessary to exclude not only the test point $\mathbf{x}_t$, but a whole segment of points lying in a certain interval $[t-c, t+c]$. For the new parameter c the average return time of the system can be used.

The model excluding these indices is denoted by $f^{t \pm c}(\mathbf{x}_t)$. Furthermore, it is good practice to normalize the model error with the variance of the time series. The

normalized mean squared error (NMSE) over p steps is then given by

$$\begin{aligned}\mathrm{NMSE}_{p,c} =& \frac{N}{p|T_{\mathrm{ref}}| \sum_{t=1}^{N}(s^t - \bar{s})^2} \sum_{t \in T_{ref}} \Bigg[\left(s_{t+1} - f^{t \pm c}(\mathbf{x}_t)\right)^2 \\ &+ \sum_{i=1}^{p-1} \left(s_{t+i+1} - f^{(t+i) \pm c}(\hat{\mathbf{x}}_{t+i})\right)^2 \Bigg] .\end{aligned} \tag{2.8}$$

2.5 Local modeling with nearest neighbors

Given the modeling problem defined in section 2.1, the most common procedure for getting an estimate for a new input vector $\mathbf{q}$ is to first fit a parametric function $f(\mathbf{x}, \boldsymbol{\theta})$ on the data set $\boldsymbol{\Omega}$, where $\boldsymbol{\theta}$ is a set of parameters which has to be optimized (e.g. with a maximum likelihood approach). After fitting the function $f(\mathbf{x}, \boldsymbol{\theta})$, an estimate for $\mathbf{q} \notin \boldsymbol{\Omega}$ can be obtained by evaluating $f(\mathbf{q}, \boldsymbol{\theta})$. This procedure is also known as *global parametric modeling*, since a parametric function is fitted to the whole data set before the model can be queried.

In contrast to these global models, pure local models delay any computation until queried with the new vector input $\mathbf{q}$. A small neighborhood of $\mathbf{q}$ is located in the training set and a simple model using only the points lying in this neighborhood is constructed [24]. In statistical learning theory, local models are also referred to as *lazy learners* [4].

As the model is constructed in a neighborhood of the query $\mathbf{q}$, local modeling falls in the category of non-parametric regression, where no kind of functional form is preconditioned for the whole model. The data set is an inseparable part of the model construction and the quality of the resulting model highly depends on it. In contrast, in parametric regression the model $f(\mathbf{x}, \boldsymbol{\theta})$ has a fixed functional form and the data points are only used to calculate or train the model parameters. After training, the resulting model can be separated from the data set and written down in closed form. Therefore, the model has a fixed functional form, which is particularly useful if this functional form is assumed or even known beforehand, e.g. by a physical theory where the parameters may also have a meaningful interpretation. In this case, parametric models are also more efficient than non-parametric ones, since they need less data for obtaining an accurate model that describes the data. However, if there does not exist any a priori knowledge, the functional form used may be unable to describe the data generating process and the model will fail completely.

The neighborhood of the query in which the local model is constructed can be chosen in two different ways. The most common choice is to locate the k nearest neighbors $\mathbf{x}_{nn_1}, \ldots, \mathbf{x}_{nn_k}$ of $\mathbf{x}$, i.e. the k points in the data set which have the smallest distance to the query point according to some arbitrary metric $\|\cdot\|$ (usually euclidean). This type of neighborhood is also known as *fixed mass*, because the number of nearest neighbors remains constant. Alternatively, one can search for all

points lying in some fixed neighborhood of the query point (*fixed size*) so that the actual number of neighbors varies. The fixed mass neighborhood is easier to handle, since it varies its size according to the density of points and empty neighborhoods cannot occur.

The problem of finding nearest neighbors is very well studied and there are numerous algorithms for this task [8, 26, 53]. We use an algorithm called *ATRIA*, which relies on a binary search tree built in a preprocessing stage [55]. This algorithm is particularly effective when the points are close to a low dimensional manifold, even when the actual dimension of the input space is large. Therefore, it is very well suited for the case where most of the data lie on a low dimensional attractor.

2.5.1 Local polynomial modeling

Local models use only a neighborhood of the query $\mathbf{q}$ to calculate the model output. Since the neighborhood is usually small, the actual model used should not be too complex. A good choice is to implement a polynomial model with low degree m, where the coefficients are calculated using the well known least squares method. In the following, we choose a fixed mass neighborhood with k nearest neighbors.

One drawback of these simple local models is that they do not produce continuous output, because shifting the query point results in points suddenly entering or leaving the neighborhood. To smooth the model output, one can apply some kind of weights on the nearest neighbors, so that farther neighbors have a lesser effect on the output than the ones lying nearer to the query point.

To apply a weighted least squares method, we define

$$\mathbf{X} = \begin{pmatrix} 1 & M(\mathbf{x}_{nn_1})_1^m \\ \vdots & \vdots \\ 1 & M(\mathbf{x}_{nn_k})_1^m \end{pmatrix} \tag{2.9}$$

where $M(\mathbf{x})_1^m$ denotes all monomials of $\mathbf{x} \in \mathbb{R}^d$ with degree $1 \leq i \leq m$. The output vector is given by $\mathbf{y} = [y_{nn_1}, \ldots, y_{nn_k}]^{\mathrm{T}}$ and the coefficient vector by $\boldsymbol{\nu} = [\nu_1, \ldots, \nu_l]^{\mathrm{T}}$ with $l = |M(\mathbf{x})_1^m| + 1$. Additionally, we introduce the weight matrix $\mathbf{W} = \operatorname{diag}\{w_1, \ldots, w_k\}$. The weighted sum of squared errors, which is to be minimized is now given by

$$P(\boldsymbol{\nu}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\nu})^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\nu})\,. \tag{2.10}$$

Setting the gradient of this function to zero leads to the solution for the coefficient vector. With $\mathbf{X}_W = \mathbf{W} \cdot \mathbf{X}$ and $\mathbf{y}_W = \mathbf{W} \cdot \mathbf{y}$, we get

$$\boldsymbol{\nu} = (\mathbf{X}_W^{\mathrm{T}}\mathbf{X}_W)^{-1}\mathbf{X}_W^{\mathrm{T}}\mathbf{y}_W = (\mathbf{X}_W)^{\dagger}\mathbf{y}_W\,, \tag{2.11}$$

where $(\mathbf{X}_W)^{\dagger}$ denotes the pseudo inverse of $\mathbf{X}_W$, which can be calculated by using singular value decomposition [64] (see also section 2.5.5).

2.5.2 Locally constant models

Setting the degree of the polynomial model to zero gives the local averaging model, where all input vectors are eliminated from the matrix $\mathbf{X}$. The coefficient vector can now be written as

$$\begin{aligned} \nu &= (\mathbf{1}_k^{\mathrm{T}}\mathbf{W}\mathbf{1}_k)^{-1}\mathbf{1}_k^{\mathrm{T}}\mathbf{W}\mathbf{y} && (2.12)\\ &= \frac{\sum_{i=1}^{k} w_i^2 y_{nn(i)}}{\sum_{i=1}^{k} w_i^2} && (2.13)\\ &= \hat{y}\,, \end{aligned}$$

i.e. a weighted average of the output values of the k nearest neighbors. Although this seems to be an overly simplistic approach, this model can produce quite remarkable results [54]. It has several advantages over more complex models. Most important of all, local averaging models are always stable, as the model output is bounded by the output values of the nearest neighbors.

Furthermore, these models are very fast as they require almost no computation besides nearest neighbor searching. Another advantage, especially when dealing with small data sets, is the ability of local averaging models to work with very small neighborhoods, as even one nearest neighbor is enough to produce a stable model output.

2.5.3 Locally linear models

Choosing a degree of $m = 1$ gives the locally linear model, where a weighted linear regression is performed on the output values of the nearest neighbors. The model output is now given by

$$\hat{y} = \langle [1 \;\; \mathbf{q}^{\mathrm{T}}], \boldsymbol{\nu} \rangle \,. \tag{2.14}$$

In many cases, especially when many data points are available, the locally linear model gives far better results. However, to guarantee a stable model one usually has to perform some kind of regularization, which will be discussed in section 2.5.5. Locally linear models are also computationally more expensive since they require a least squares optimization of the coefficients.

2.5.4 Parameters of local modeling

Number of nearest neighbors

The number of nearest neighbors k is the most crucial parameter, as it directly affects the bias and variance of the resulting local model. A small number of nearest neighbors leads to a model with high variance and low bias. In the extreme case, a local averaging model with one nearest neighbor simply interpolates the outputs of the nearest neighbors of the data points. Conversely, a large number of neighbors

leads to a model with high bias and low variance and in the extreme case to a very simple global model.

Weight function

A good choice for weighting the nearest neighbors are functions of the form

$$w_n(r) = (1 - r^n)^n \quad , \quad r = \frac{d_i}{d_{\max}} \, , \tag{2.15}$$

where $d_{\max} = \|\mathbf{x}_k - \mathbf{q}\|$ is the distance to the furthest nearest neighbor and $d_i = \|\mathbf{x}_i - \mathbf{q}\|$ the distance to the nearest neighbor with index $i < k$. Depending on the exponent n, different kinds of weight functions can be obtained: with $n = 0$ no kind of weighting is applied, whereas $n = 1$ leads to linear weighting. Choosing $n = 2$ leads to biquadratic, $n = 3$ to tricubic weight functions. It is obvious that the type of weight function and the number of nearest neighbors are connected: choosing a high exponent n effectively reduces the number of nearest neighbors which affect the model output. However, the main motivation for using a weight function is to smooth the model output. Its effect on the accuracy of the model is mostly not very high, as long as any kind of weighting is done. Therefore, it is usually sufficient to choose n between 0 and 3.

Distance metric

The kind of distance metric used has a strong influence on the neighborhood of the query point. Although the euclidean metric will often be a good choice, other metrics can sometimes significantly improve model accuracy. By using the *diagonally weighted euclidian distance*

$$\begin{gathered} d_{\mathrm{dwe}}(\mathbf{x}, \mathbf{q})^2 = \sum_{i=1}^{d} \lambda_i^2 (x_i - q_i)^2 = (\mathbf{x} - \mathbf{q})^{\mathrm{T}} \mathbf{\Lambda}^2 (\mathbf{x} - \mathbf{q}) \\ \text{where} \quad \mathbf{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda}), \ \boldsymbol{\lambda} \in \mathbb{R}^d \, , \end{gathered} \tag{2.16}$$

one can specify which components of the input vectors should be more relevant when searching for nearest neighbors and which components should be more or less dropped. Unfortunately, one does not usually know beforehand which components are vital for modeling the data set and which are irrelevant or corrupted by noise. However, one can use an algorithm which uses the leave-one-out cross-validation error for optimizing the metric parameters (see also section 2.5.6).

In the case of time series prediction, the input vectors are delay vectors of the form

$$\mathbf{x}_t = (s_t, s_{t-\tau}, \ldots, s_{t-(d-1)\tau}) \, . \tag{2.17}$$

It is questionable why certain components of the input vectors should be favoured, as a certain value s_i exists in different delay vectors at different positions. Nevertheless,

in some cases the prediction can be improved by applying a special case of the diagonally weighted distance, the *exponentially weighted distance*

$$d_{\exp}(\mathbf{x}, \mathbf{q}) = \left(\sum_{i=1}^{d} \lambda^{i-1} (x_i - q_i)^2 \right)^{1/2} . \tag{2.18}$$

In the case of delay vectors, this method favours those components of $\mathbf{x}$ which are closer in time to the prediction. Furthermore, only one parameter has to be optimized and the standard euclidian metric can still be obtained by setting $\lambda = 1$. Therefore, an optimization procedure which optimizes λ can only improve the prediction accuracy compared to the euclidean metric.

2.5.5 Regularization

Given enough data points, locally linear models are usually more precise than locally averaging models. One problem though lies in the calculation of the inverse of the matrix product $\mathbf{X}_W^{\mathrm{T}} \mathbf{X}_W$, which leads to unstable models if the resulting matrix is ill-conditioned. Therefore, some kind of regularization method must be applied. The two most common choices are the *principal components regression* (PCR) and the *ridge regression* (RR), which will be described in the following section.

Principal Component and Ridge Regression

The basic principle of this method relies on the singular value decomposition of $\mathbf{X}_W$, which is given by

$$\mathbf{X}_W = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}} , \tag{2.19}$$

where $\mathbf{U} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{l \times l}$ are orthonormal, and $\mathbf{S} = \mathrm{diag}(\sigma_1, \ldots, \sigma_p)$ is a diagonal matrix with the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$ with $p = \min\{k, l\}$. The pseudoinverse of $\mathbf{X}_W$ can now be written as

$$\mathbf{X}_W^{\dagger} = \mathbf{V}\mathbf{S}^{+}\mathbf{U}^{\mathrm{T}} , \tag{2.20}$$

where $\mathbf{S}^{+} = \mathrm{diag}(1/\sigma_1, \ldots, 1/\sigma_r, 0, \ldots, 0)$ and $r = \mathrm{rank}(\mathbf{X}_W)$ [33], hence setting $1/\sigma_i$ to zero if $\sigma_i = 0$. In practice however, these singular values are usually not exactly zero. The matrix $\mathbf{X}_W$ is not singular but ill-conditioned. The principal components regression (PCR) works by first setting these small singular values to zero and then calculating $\mathbf{S}^{+}$ as just noted. For this procedure it is crucial that the nearest neighbors are centered around the origin by subtracting the mean. This also simplifies the calculation of the locally linear model since the constant is now given by the mean of the images of the nearest neighbors $\bar{y}$. The column of ones in the design matrix (2.9) can therefore be omitted.

However, there exist different possibilities in how to decide whether a singular value is so small that it should be dropped. The easiest way is the *truncated* PCR

(TPCR), where a fixed number of the smallest singular values is automatically set to zero, without looking at their actual values. Alternatively, in *principal components threshold regression* (PCTR) every singular value below some previously defined threshold $\sigma_{\min}$ is dropped. This procedure can be further generalized by applying weights to the inverse singular values, which leads to a PCTR with *soft-thresholding*. The model output for the locally linear model can then be written as

$$\hat{y} = \bar{y} + \sum_{i=1}^{p} \langle (\mathbf{q} - \bar{\mathbf{x}})^{\mathrm{T}}, \mathbf{v}_i \rangle \left(\frac{f(\sigma_i)}{\sigma_i} \right) \langle \mathbf{u}_i^{\mathrm{T}}, \mathbf{y}_W \rangle \,, \tag{2.21}$$

where in general any kind of weight function $f(\sigma)$ can be used. McNames [52] has suggested to use a modified biquadratic weight function

$$f(\sigma) = \begin{cases} 0 & s_{\min} > \sigma \,, \\ \left(1 - \left(\dfrac{s_{\max} - \sigma}{s_{\max} - s_{\min}} \right)^2 \right)^2 & s_{\min} \leq \sigma < s_{\max} \,, \\ 1 & s_{\max} \leq \sigma \,, \end{cases} \tag{2.22}$$

where $s_{\min}$ and $s_{\max}$ are given by

$$s_{\min} \equiv s_c(1 - s_w) \tag{2.23}$$

$$s_{\max} \equiv s_c(1 + s_w) \,. \tag{2.24}$$

The parameters s_c and s_w define the center and width of the threshold in which the singular values are weighted down to zero. Singular values above $s_{\max}$ remain unchanged, whereas the ones smaller than $s_{\min}$ are set to zero. With $s_w = 0$ we get $s_{\min} = s_{\max} = s_c$ and therefore a hard threshold at s_c.

Another good choice for the weight function is given by

$$f(\sigma) = \frac{\sigma^2}{\mu^2 + \sigma^2} \,, \tag{2.25}$$

so that for $\sigma \gg \mu$ we get $f(\sigma) \approx 1$, and for $\sigma \to 0$ the weight function becomes zero. The parameter $\mu \geq 0$ therefore defines the degree of regularization. This particular weight function leads to a special case of a regularization procedure known as *Ridge Regression* or *Tikhonov Regularization* [10, 33]. Here, the cost function (2.10) is modified by adding a penalty term which penalizes large values in the coefficient vector, leading to

$$P(\boldsymbol{\nu})_{\mathrm{RR}} = (\mathbf{y} - \mathbf{X}\boldsymbol{\nu})^{\mathrm{T}} \mathbf{W}^{\mathrm{T}} \mathbf{W} (\mathbf{y} - \mathbf{X}\boldsymbol{\nu}) + \boldsymbol{\nu}^{\mathrm{T}} \mathbf{R}^{\mathrm{T}} \mathbf{R} \boldsymbol{\nu} \,. \tag{2.26}$$

The diagonal *ridge matrix* $\mathbf{R} \equiv \mathrm{diag}(r_1, \ldots, r_l)$ weighs the different components of the coefficient vector. The solution for $\boldsymbol{\nu}$ can now be written as

$$\boldsymbol{\nu} = (\mathbf{X}_W^{\mathrm{T}} \mathbf{X}_W + \mathbf{R}^T \mathbf{R})^{-1} \mathbf{X}_W^{\mathrm{T}} \mathbf{y}_W \,. \tag{2.27}$$

Therefore, the modified cost function (2.26) is equivalent to adding the values $r_1^2, \ldots, r_l^2$ to the diagonal of $\mathbf{X}_W^{\mathrm{T}}\mathbf{X}_W$. A simple (and popular) choice for the ridge matrix is $\mathbf{R} = \mu^2\mathbf{I}$, i.e. all components of $\boldsymbol{\nu}$ are weighted with the same factor μ^2. The solution (2.27) can now be easily obtained by using the singular value decomposition $\mathbf{X}_W = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$ and this leads to

$$\boldsymbol{\nu} = \sum_{i=1}^{k} \frac{\sigma_i}{\sigma_i^2 + \mu^2} \langle \mathbf{u}_i^{\mathrm{T}}, \mathbf{y}_W \rangle \mathbf{v}_i \tag{2.28}$$

and therefore to the above mentioned weight function (2.25).

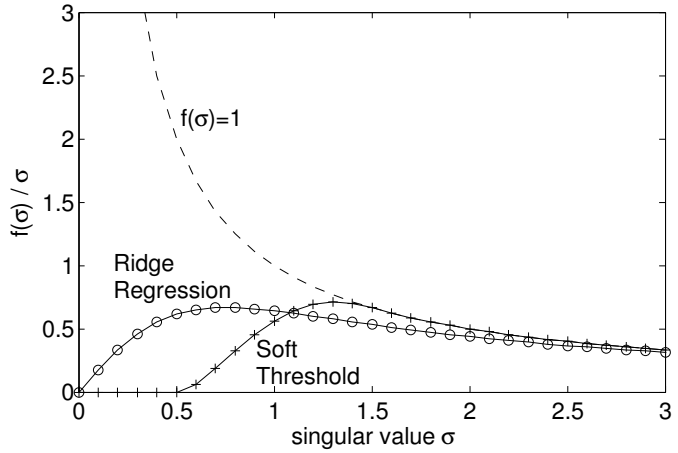


Figure 2.1: Example for regularization with Ridge Regression and PCTR with soft-threshold. The dashed line shows the inverse $1/\sigma$ which goes to infinity for $\sigma \to 0$. The circles show the regularized singular value with Ridge Regression and $\mu = 0.75$, while the crosses show PCTR with soft-threshold and $s_c = 1, s_w = 0.5$.

An illustration of both regularization techniques can be seen in figure 2.1. While PCTR has the advantage that it can locally adapt to the dimensionality of the data, Ridge Regression in its general form (2.26) can put different regularization parameters on each component through the regularization matrix $\mathbf{R}$. Both methods can produce good results, however, it has been shown that in the case of time series prediction of chaotic systems, principal component regression with thresholding is superior to ridge regression [52].

Local projection

Another possibility for regularization is to reduce the dimensionality of the points found in the neighborhood of the query before performing the least squares optimization. This can be done by performing a Principal Component Analysis (PCA) on

the nearest neigbors and then projecting them onto the subspace which covers most of the variance of the data [13, 41]. Given the following matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{x}_{nn_1}^{\mathrm{T}} - \bar{\mathbf{x}}_{nn}^{\mathrm{T}} \\ \dots \\ \mathbf{x}_{nn_k}^{\mathrm{T}} - \bar{\mathbf{x}}_{nn}^{\mathrm{T}} \end{pmatrix} \tag{2.29}$$

containing the centered nearest neighbors of the query, the eigenvalues and eigenvectors of the empirical covariance matrix $\mathbf{C} = \mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}$ are calculated. The eigenvalues correspond to the variance in the direction given by the corresponding eigenvalue. Keeping only the first r eigenvectors with eigenvalues above some given threshold σ, we can define through these remaining eigenvectors a lower dimensional subspace which covers most of the variance of the data. The nearest neighbors projected into this subspace are given by $\tilde{\mathbf{A}} = \mathbf{A} \cdot \mathbf{P}_r$, with the projection matrix given by

$$\mathbf{P}_r = (\mathbf{v}_1 \dots \mathbf{v}_r) \ , \tag{2.30}$$

consisting of the eigenvectors corresponding the the first r largest variances. This also effectively removes noise present in the data, given that the noise is small so that it only contributes a small amount to the variance. The coefficients for the local model can then be calculated in this lower dimensional subspace.

This procedure is very closely related to TPCR. In fact, for local linear models it is equivalent, given that the nearest neighbors are centered around their mean, since the design matrix $\mathbf{X}$ from (2.9) is then equal to $\mathbf{A}$. It follows that $\tilde{\mathbf{A}} = \mathbf{A} \cdot \mathbf{P}_r = \mathbf{U}_r \cdot \mathbf{S}_r$, where $\mathbf{U}_r$ and $\mathbf{S}_r$ denote the matrices $\mathbf{U}$ and $\mathbf{S}$ from the SVD in (2.20), but reduced to the r largest singular values. The pseudo inverse of $\tilde{\mathbf{A}}$ is then given by

$$\tilde{\mathbf{A}}^{\dagger} = (\tilde{\mathbf{A}}^{\mathrm{T}}\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^{\mathrm{T}} = \mathbf{S}_r^{-1}\mathbf{U}_r^{\mathrm{T}} \, . \tag{2.31}$$

Given the query $\mathbf{q}$, we obtain for the model output

$$\hat{y} = \mathbf{q} \cdot \mathbf{P}_r \cdot \tilde{\mathbf{A}}^{\dagger}\mathbf{y} = \mathbf{q} \cdot \mathbf{P}_r \cdot \mathbf{S}_r^{-1}\mathbf{U}_r^{\mathrm{T}}\,\mathbf{y} \tag{2.32}$$

which is equivalent to TPCR since $\mathbf{P}_r = (\mathbf{v}_1 \dots \mathbf{v}_r) = \mathbf{V}_r$.

For locally quadratic models or local models with other model types like radial basis function networks, the local projection and TPCR are not equivalent anymore. TPCR with soft thresholding introduced in the previous section is more flexible and can often lead to better results than PCTR without soft-thresholding or local projection.

2.5.6 Parameter optimization for local modeling

Several different parameters have to be set for local modeling. Most of these parameters deal with the neighborhood of the query point: the kind of metric used

for calculating the distances between the query point and its neighbors, the number of nearest neighbors k and the weight function applied. The other parameters deal with the model used in the neighborhood: one has to choose between locally averaging or locally linear models, and for the latter, one has to choose a regularization method. The regularization itself has additional parameters associated, which have a large influence on the stability and accuracy of the model, especially in the case where the model is iterated over several steps.

Although all these parameters have a more or less intuitive appeal, it is difficult to find good values based on simple "trial and error". Furthermore, these parameters are not independent from each other: the distance metric and weight function directly affect the form and size of the neighborhood which is primarily controlled by the number of nearest neighbors. On the other hand, changing the type of model or the regularization parameters often demands other forms of neighborhoods.

Good parameter values can be found by applying an optimization algorithm using the leave-one-out cross-validation error. Although local models allow an efficient calculation of this error, it is still a time consuming task, especially for large data sets combined with multiple step prediction. Moreover, gradient-based optimization algorithms are mostly not applicable, as only the regularization and metric parameters allow the calculation of a gradient.

One popular approach for such an optimization problem is to use genetic algorithms [32], as they do not need a gradient and are able to deal with integer and floating point parameters at the same time. They are well suited for optimizing embedding parameters, especially when a non-uniform embedding is used [6].

Genetic algorithms start with a randomly chosen population of parameter vectors which can contain the delays of the embedding as well as the number of nearest neighbors or any other parameter for local modeling. This population is then "evolved" by using different types of inheritance, mutation and selection operators. The algorithm stops after a certain number of iterations.

However, it is not advisable to optimize all parameters at once with a genetic algorithm, as the initial population and the number of iterations has to be very large for the algorithm to converge. This may be due to the fact that the parameters are not of equal importance. Therefore, we first use a genetic algorithm to optimize only the delays for a non-uniform embedding and the number of nearest neighbors, since these are the most crucial parameters for a good model performance. During this optimization step, the other parameters are held constant; we used biquadratic weights with an euclidean distance, and for locally linear models the regularization procedure given by (2.22) with $s_c = 1 \cdot 10^{-4}$ and $s_w = 0.6$.

After this primary step, the other parameters are optimized using a simple type of cyclic optimization, where all parameters are successively optimized with an exhaustive search in the case of integer parameters and with a semi-global line search for floating point parameters [52]. Although one has a good value for the number of nearest neighbors, it should be included in this second optimization step since it is the most crucial parameter.

Because of local minima, this optimization procedure will not necessarily lead to the global minimum in parameter space, but nevertheless it will usually improve prediction accuracy compared to manually chosen parameters.

2.6 Cluster Weighted Modeling

Cluster Weighted Modeling (in the following denoted as CWM), an algorithm first introduced by Gershenfeld et al. [29,30], lies between the local and the global modeling approach. It is global in the sense that the model has to be trained beforehand with the whole data set, hence it lacks the flexibility of lazy learning. But it is also local in the sense that usually only the points lying in a neighborhood of the query point are crucial for the model output.

The central idea is to describe the compound density $p(y, \mathbf{x})$ of input vectors $\mathbf{x}$ and outputs y by means of a sum of simple models, each of which approximates the true density in the vicinity of a *cluster center*. Once a model of the compound density $p(y, \mathbf{x})$ is successfully built, one is able to compute derived quantities like the conditional forecast $\langle y|\mathbf{x}\rangle$ for new query points. This idea is also known under the name *Finite Mixture Model*, including the well known and widely used Gaussian Mixture Model (GMM), often employed for problems involving clustering of data [51]. This is also an example for unsupervised classification, since there is no correction or calculation of the model error with respect to some known output values. CWMs have much in common with Gaussian Mixture Models, but they offer one crucial extension: in each cluster, they include a model for the *functional* dependence between the input vectors and the target values, and therefore combine the stochastic Mixture Model approach with deterministic modeling. This also implies that CWMs are used in supervised learning scenarios, while Finite Mixture Models are usually aimed at unsupervised learning problems (like the already mentioned clustering of data vectors).

Generally, CWMs are written as a sum

$$p(\mathbf{x}, y) := \sum_{m=1}^{M} p_m(y, \mathbf{x}), \tag{2.33}$$

where m enumerates the clusters, and $p_m(y, \mathbf{x})$ is a density of a specific form discussed below. The total number of clusters M must be chosen beforehand and can be optimized using cross validation.

The density $p_m(y, \mathbf{x})$ is written as

$$p_m(y, \mathbf{x}) := p_m(y|\mathbf{x}) \cdot p_m(\mathbf{x}) \cdot w_m. \tag{2.34}$$

The terms in (2.34) have the following interpretation (the terminology was adopted from the original reference [30]):

Cluster Weights: The cluster weight $w_m \in [0,1]$ specifies the fraction of the data set explained by cluster m. The w_m are chosen subject to the normalization constraint

$$\sum_m w_m = 1. \tag{2.35}$$

Domains of Influence: The density $p_m(\mathbf{x})$ describes the domain of influence of cluster m, that is, the distribution of the inputs $\mathbf{x}$ around the cluster. They are chosen as multivariate normal probability densities, i.e.

$$p_m(\mathbf{x}) = \frac{|\mathbf{C}_m^{-1}|^{1/2}}{(2\pi)^{d/2}} \exp\left(-(\mathbf{x}-\boldsymbol{\mu}_m)^{\mathrm{T}} \cdot \mathbf{C}_m^{-1} \cdot (\mathbf{x}-\boldsymbol{\mu}_m)/2\right) , \tag{2.36}$$

with mean $\boldsymbol{\mu}_m$ and covariance matrix $\mathbf{C}_m$, effectively describing, respectively, the location and the range of influence of the cluster m. When dealing with high dimensional spaces, it is advisable to reduce these input domains to separable Gaussians, with single variances in each dimension, i.e. $\mathbf{C}_m = \mathrm{diag}\{\sigma_{m,1}^2, \ldots, \sigma_{m,d}^2\}$. Thus, the input term becomes

$$p_m(\mathbf{x}) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_{m,i}^2}} \exp\left(-(x_i-\mu_{m,i})^2/2\sigma_{m,i}^2\right) , \tag{2.37}$$

which is also what will be used in the following sections and practical examples. Working with such a reduced input domain means that the axes of the clusters are always aligned within the given coordinate system, while they can rotate in any direction when working with full covariance matrix.

The $p_m(\mathbf{x})$ are normalized:

$$\int p_m(\mathbf{x}) \mathrm{d}\mathbf{x} = 1 \qquad \forall m. \tag{2.38}$$

Output terms: The density $p_m(y|\mathbf{x})$ is the conditional density of the outputs y given the inputs $\mathbf{x}$ around the cluster m.

They are also chosen as multivariate normal probability densities

$$p_m(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi\Sigma_m^2}} \exp\left(-[y-f(\mathbf{x},\boldsymbol{\beta}_m)]^2/2\Sigma_m^2\right) , \tag{2.39}$$

with mean $f(\mathbf{x},\boldsymbol{\beta}_m)$ and variances Σ_m^2, effectively describing, respectively, the local cluster model and the corresponding error statistics around the cluster m. The vector $\boldsymbol{\beta}_m$ denotes the parameters for the function in cluster m. These functions $f(\mathbf{x},\boldsymbol{\beta}_m)$ are often called local models but will in the following be denoted as *cluster functions*, to avoid confusion with the local nearest neighbor models introduced earlier.

The $p_m(y|\mathbf{x})$ are normalized thus

$$\int p_m(y|\mathbf{x})\mathrm{d}y = 1 \qquad \forall m, \mathbf{x}. \tag{2.40}$$

The cluster functions are chosen as linear combinations

$$f(\mathbf{x}, \boldsymbol{\beta}_m) = \sum_{i=1}^{I} \beta_{m,i} f_i(\mathbf{x}) \tag{2.41}$$

of basis functions $f_i(\mathbf{x})$ (e.g. monomials). Next to the number of clusters M, the number I and type f_i of the basis functions directly determines the complexity of the resulting CWM and hence control bias and variance.

The reason for choosing the output term in the above way becomes clear when we look at the model output, i.e. the conditional forecast which we can obtain by integrating the output values with respect to the conditional density $p(y|\mathbf{x})$,

$$\begin{aligned} \langle y|\mathbf{x}\rangle = \int y\, p(y|\mathbf{x})\, dy = \int y\, \frac{p(y,\mathbf{x})}{p(\mathbf{x})}\, dy = \frac{\sum_{m=1}^{M} \int y\, p_m(y|\mathbf{x})\, dy\, p_m(\mathbf{x})\, w_m}{\sum_{m=1}^{M} p_m(\mathbf{x}) w_m} \\ = \frac{\sum_{m=1}^{M} f(\mathbf{x}, \boldsymbol{\beta}_m) p_m(\mathbf{x})\, w_m}{\sum_{m=1}^{M} p_m(\mathbf{x}) w_m}\,. \end{aligned} \tag{2.42}$$

The model output of the CWM is therefore given by a weighted average of the cluster functions $f(\mathbf{x}, \boldsymbol{\beta}_m)$. The Gaussians, which are given by the input domains $p_m(\mathbf{x})$, control the interpolation of the cluster functions and therefore do not serve as approximators like in conventional radial basis function networks. This will become clearer in the example given in the next section.

One is now confronted with the problem of finding good values for

- the weights w_m,
- the means $\boldsymbol{\mu}_m$ and variances $\boldsymbol{\sigma}_m^2$ of the input domains,
- the variances of the output terms Σ_m^2, and
- the parameters of the cluster functions $\boldsymbol{\beta}_m$.

The task of parameter optimization is done using an *Expectation-Maximization* (EM) algorithm, which will be further explained in section 2.6.2, in combination with a Maximum Likelihood estimation of the cluster function parameters.

2.6.1 Illustrative example

To further convey the basic principle of CWMs, a simple one-dimensional example shall be shown in this section. We want to model a small data set which stems from the sine function, using a model with only two clusters, each containing a linear model as cluster function. As data set, we take 50 equidistantly sampled points from the sine function. Figure 2.2 shows the resulting CWM, with two clusters with weights $w_1 = 0.33$ and $w_2 = 0.66$, meaning that they cover about one third and two thirds of the data points, respectively. In the next plot, the two linear cluster functions are plotted, which approximate the data points in the vicinity of the cluster. The resulting model output can be seen as essentially converging towards these two linear functions, interpolated through the two Gaussians where the clusters overlap.

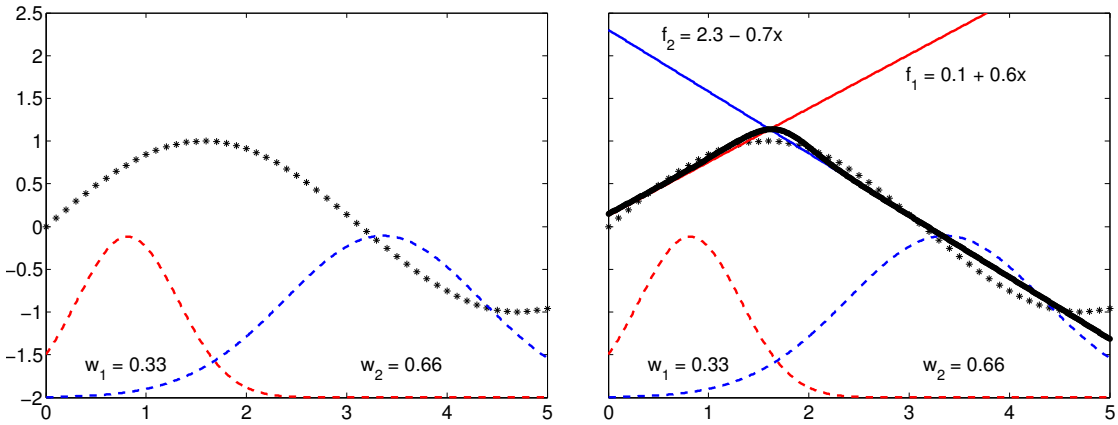


Figure 2.2: Example for a simple CWM. Left plot: Data points (*) covered by two clusters (red/blue dashed lines) with cluster weights w_1, w_2, which can be interpreted as the percentage of points the clusters cover. The right plot shows the two linear cluster functions f_1, f_2 (red/blue) and the resulting model output (black line).

Two important remarks regarding this example:

1. As can be seen in the right plot, it is the cluster functions f_1 and f_2 which approximate the data set, while the Gaussians only serve as interpolants and define the area of influence for the local functions. This is the crucial difference to a Radial Basis Function Network, where the data is approximated by a superposition of the Gaussians themselves (or any other RBF, by that matter).

2. Cluster Weighted Models produce a whole probability distribution (pdf) as output; in fact, the right plot in figure 2.2 shows only the expectation of this pdf. A plot of the full distribution can be seen in figure 2.3. It is clear that

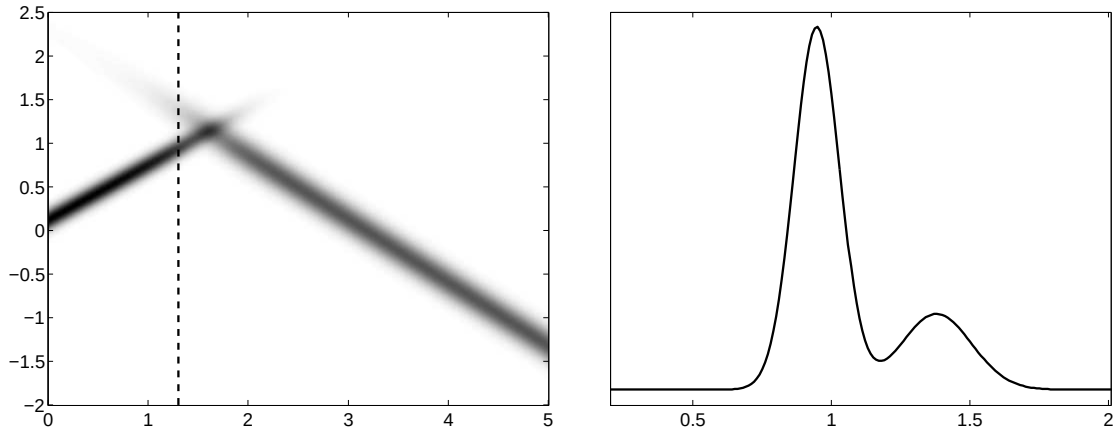


Figure 2.3: Plot of the full probability distribution for the above example. The model output plotted in figure 2.2 is the expectation value of this distribution. The distribution along the dashed line is shown in the right plot, which exhibits a bimodal structure.

this full distribution contains much more information than just the mean; for example, this distribution becomes bimodal in parts of the area where the clusters overlap, an example of which is shown in the right plot of the figure. Also, the distribution generated by the second (right) cluster is less sharp, since it covers more of the nonlinear features of the sine function than the first one.

It is instructional to see how the CWMs behave in areas where data is unavailable. In figure 2.4, we again see a CWM with two clusters with data sampled from the sine function, but we now have an interval [1.5 3] where the data points are missing. The two clusters now cover roughly the same amount of points and therefore have almost equal weights. They still produce a legit model output, but from the resulting mean it does not become clear that the model is completely uncertain in the area with no available data points. While in this simple one-dimensional example the missing data is obvious and can be directly seen by simply plotting it, this is usually no longer the case for more complex and higher-dimensional problems.

The uncertainty of the model can clearly be seen when looking at the full probability distribution, which is shown in Figure 2.5. Right in the middle of the data gap, at $x = 2.25$, the resulting distribution is bimodal, with both peaks almost the same height, albeit slightly different widths.

Therefore, reducing the model output to the mean implies ignoring additional and often important information. But this directly leads to another problem: How can we quantify full probability distributions, and how can we compare different models regarding the full distribution? The most simple answer is to calculate higher moments, like the variance, to express model uncertainty. This is also what will

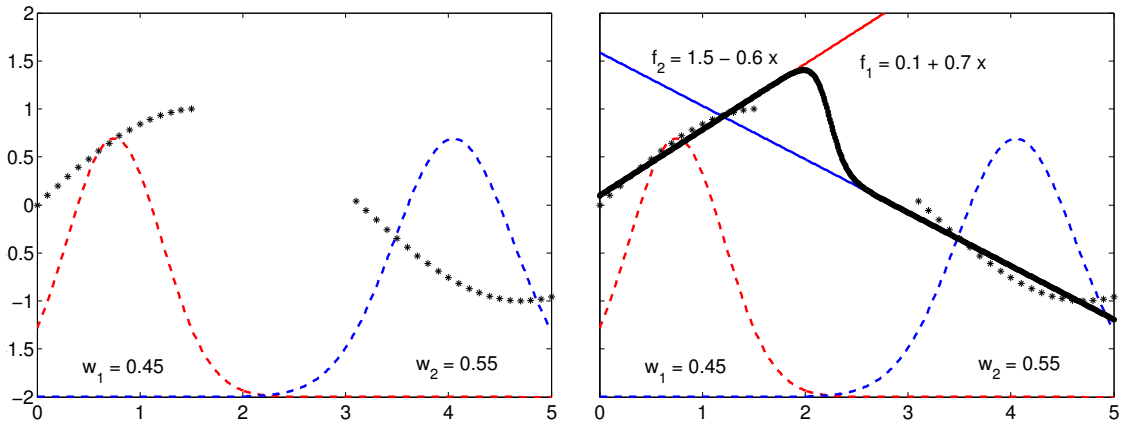


Figure 2.4: CWM example but with missing data points in [1.5 3]. The clusters now almost have the same weights since they cover roughly the same amount of points. In the right plot, the cluster functions f_1, f_2 (red/blue) are seen with the resulting model output (black line).

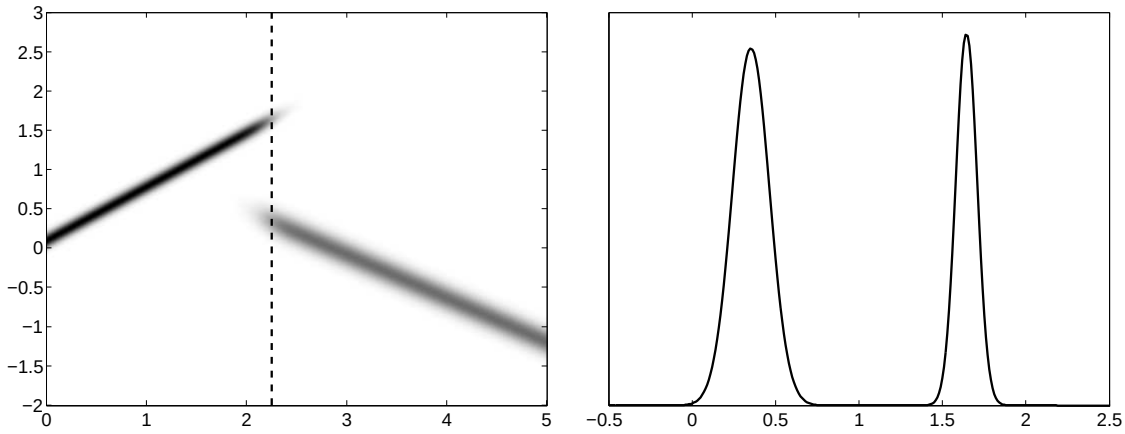


Figure 2.5: Full distribution for the CWM with missing data points in $[1.5 \;\; 3]$. In the right plot, the distribution along the dashed line is shown, which is at $x = 2.25$ and therefore in the middle of the missing data interval. The distribution there is completely bimodal.

be done when dealing with Active Learning in chapter 5. However, for the task of comparing different models with probabilistic output, we will introduce the concept of *scores* in section 3.2.

2.6.2 Expectation Maximization algorithm

The EM algorithm is an iterative maximum-likelihood estimator and is typically used when one is confronted with *incomplete data* or when the likelihood function involves *latent variables*. However, the distinction of these two cases is more a matter of interpretation, since we can always think of latent variables as data which we could not observe, thus leading to incomplete data. Therefore, it is possible to use the EM algorithm by introducing artificial additional variables which are simply declared as unobserved.

The EM algorithm was discovered and used independently by several different researchers, but it was first described fully by Dempster et al. in [21], who also coined the term "EM algorithm". One important problem which motivated this algorithm, also in the original paper, was the parameter estimation for Gaussian Mixture Models [51], and since Cluster Weighted Models are closely related to this problem, it is evident to use this algorithm here as well.

Since Dempster's paper, a huge amount of material was published which further investigated and used the algorithm for various purposes. It is especially important for tomographic image reconstruction problems, where for example in Positron Emission Tomography (PET) it is used to calculate from which locations the positrons where emitted, based on the data from the detectors around the patient [50]. Other applications include the training of hidden Markov models, especially for speech recognition, pattern recognition, and neural network training.

General formulation

For describing the EM algorithm in a general fashion, let us consider a parametric density function $p(x|\mathbf{\Psi})$, where $\mathbf{\Psi}$ are the parameters; in the case of CWMs, these parameters are the cluster weights and the positions and variances of the clusters. The random variable X is assumed to be i.i.d. according to this distribution, and we have a data set $\mathbf{\Omega} = \{x_1, \ldots, x_N\}$ which shall be a realization of this random variable. The likelihood function is then given by

$$L(\mathbf{\Psi}|X) = p(X|\mathbf{\Psi}) = \sum_{i=1}^{N} p(x_i|\mathbf{\Psi})\,. \tag{2.43}$$

For obtaining an estimate of the parameters $\mathbf{\Psi}$, the most common approach is to use those parameters with the *Maximum Likelihood*, i.e. those parameters $\hat{\mathbf{\Psi}}_{ML}$ which fulfill

$$\hat{\mathbf{\Psi}}_{ML} = \arg\max_{\Psi} \quad L(\mathbf{\Psi}|X)\,. \tag{2.44}$$

In many practical cases, it is easier to calculate this maximization by using the log-likelihood $\mathcal{L}(\mathbf{\Psi}|X) \equiv \log L(\mathbf{\Psi}|X)$, which shall also be used in the following. Still, calculating this maximization analytically is often intractable or even impossible, so that one has to resort to approximative procedures.

As already mentioned, the EM algorithm is especially suited for cases with incomplete data. This covers two different scenarios: we could have incomplete data because we actually could not observe some data, e.g. due to technical difficulties or other limitations of our observation process. The second scenario is that we have all data we can obtain, but if we would introduce additional, fictitious parameters, also called *hidden variables*, the maximization of the likelihood (2.44) would become much easier. This latter scenario is usually more common, and is also the case with Mixture- and Cluster Weighted Models.

The basic strategy of EM is to first formulate the complete data problem, given a hypothetical complete data set $\mathbf{\Omega}_C = \{x_1, \ldots, x_N, z_1, \ldots, z_M\}$, where the z_i denote the hidden variables.

The complete log-likelihood is given by

$$\mathcal{L}_C(\mathbf{\Psi}|X, Z) = \log p(X, Z|\mathbf{\Psi}) \,, \tag{2.45}$$

and we further assume that the joint density can be further decomposed according to Bayes' rule into

$$p(X, Z|\mathbf{\Psi}) = p(Z|X, \mathbf{\Psi}) \cdot p(X|\mathbf{\Psi}) \,. \tag{2.46}$$

Since the hidden variables z_i are realizations of the random variable Z, the complete log-likelihood $\mathcal{L}_C$ is also a random variable, and we can calculate its expectation value with respect to Z, given the observed data $\{x_i\}$ and a current parameter estimate $\mathbf{\Psi}^{(k)}$. This expectation is usually written as the *Q function*

$$Q(\mathbf{\Psi}; \mathbf{\Psi}^{(k)}) = \mathrm{E}\left\{ \log p(X, Z|\mathbf{\Psi}) \,|\, X, \mathbf{\Psi}^{(k)} \right\} \,. \tag{2.47}$$

This is the *Expectation step* (E-step) of the algorithm. It is important to note that $\mathbf{\Psi}$ and Z are random variables, but $\mathbf{\Psi}^{(k)}$ and X are constant. While our observed data always stays the same, the $\mathbf{\Psi}^{(k)}$ usually changes with each iteration, with k denoting the current step.

In the *Maximization step* (M-step), the just calculated expectation $Q(\mathbf{\Psi}; \mathbf{\Psi}^{(k)})$ is maximized with respect to $\mathbf{\Psi}$, the result being our parameter estimate for the next iteration:

$$\mathbf{\Psi}^{(k+1)} = \arg\max_{\Psi} \quad Q(\mathbf{\Psi}; \mathbf{\Psi}^{(k)}) \,. \tag{2.48}$$

As already mentioned, this assumes that maximizing the complete log-likelihood is feasible. But even if this is not the case, it is in fact sufficient to just find a parameter estimate $\mathbf{\Psi}^{(k+1)}$ with a larger likelihood than the previous one. It is therefore possible to use e.g. a numerical steepest ascent method, which may only find a local maximum of Q.

Convergence of the EM algorithm

In every iteration, one E- and M-step is performed, as described in the previous section. The problem now is to determine if doing these iterations will ever converge, and if so, if it will converge to a ML estimator.

Let $\mathcal{Z}$ be the space of all possible values of Z, so that

$$\sum_{Z\in\mathcal{Z}} p(Z|X,\mathbf{\Psi}^{(k)}) = 1\,, \tag{2.49}$$

and hence it holds that

$$\log p(X|\mathbf{\Psi}) = \sum_{Z\in\mathcal{Z}} p(Z|X,\mathbf{\Psi}^{(k)}) \log p(X|\mathbf{\Psi})\,. \tag{2.50}$$

Using Bayes' rule (2.46), we obtain

$$\log p(X|\mathbf{\Psi}) = \sum_{Z\in\mathcal{Z}} p(Z|X,\mathbf{\Psi}^{(k)}) \log p(Z,X|\mathbf{\Psi}) - \sum_{Z\in\mathcal{Z}} p(Z|X,\mathbf{\Psi}^{(k)}) \log p(Z|X,\mathbf{\Psi})\,. \tag{2.51}$$

Using the definition of the Q-function (2.47), we can see that the first term in this equation is $Q(\mathbf{\Psi},\mathbf{\Psi}^{(k)})$. Therefore, the fraction of the two incomplete likelihoods before and after an iteration can be written as

$$\begin{aligned}\log\frac{p(X|\mathbf{\Psi})}{p(X|\mathbf{\Psi}^{(k)})} = \log p(X|\mathbf{\Psi}) - \log p(X|\mathbf{\Psi}^{(k)}) &= Q(\mathbf{\Psi},\mathbf{\Psi}^{(k)}) - Q(\mathbf{\Psi}^{(k)},\mathbf{\Psi}^{(k)}) \\ &\quad + \sum_{Z\in\mathcal{Z}} p(Z|X,\mathbf{\Psi}^{(k)}) \log\frac{p(Z|X,\mathbf{\Psi}^{(k)})}{p(Z|X,\mathbf{\Psi})}\end{aligned} \tag{2.52}$$

The last term is the Kullback-Leibler divergence between the two densities in the fraction, which by definition is always positive, hence it holds that

$$\log p(X|\mathbf{\Psi}) - \log p(X|\mathbf{\Psi}^{(k)}) \geq Q(\mathbf{\Psi},\mathbf{\Psi}^{(k)}) - Q(\mathbf{\Psi}^{(k)},\mathbf{\Psi}^{(k)})\,. \tag{2.53}$$

We now calculate $\mathbf{\Psi}^{(k+1)}$ through the M-step as described in the previous section, i.e. through a maximization of the Q function

$$\mathbf{\Psi}^{(k+1)} = \arg\max_{\Psi} \quad Q(\mathbf{\Psi},\mathbf{\Psi}^{(k)})\,. \tag{2.54}$$

Putting this into (2.53) yields

$$\begin{aligned}\log p(X|\mathbf{\Psi}^{(k+1)}) - \log p(X|\mathbf{\Psi}^{(k)}) &\geq Q(\mathbf{\Psi}^{(k+1)},\mathbf{\Psi}^{(k)}) - Q(\mathbf{\Psi}^{(k)},\mathbf{\Psi}^{(k)}) \\ &\geq Q(\mathbf{\Psi},\mathbf{\Psi}^{(k)}) - Q(\mathbf{\Psi}^{(k)},\mathbf{\Psi}^{(k)}) \\ &\geq 0\end{aligned} \tag{2.55}$$

and therefore

$$\log p(X|\mathbf{\Psi}^{(k+1)}) \geq \log p(X|\mathbf{\Psi}^{(k)}) \,, \tag{2.56}$$

which is the result we were looking for: the likelihood increases with each iteration, until the conditions for equality are satisfied, which implies that a fixed point is reached. It is clear from (2.54) that every maximum-likelihood parameter estimate $\mathbf{\Psi}^{ML}$ is a fixed point of the iteration, and given the assumption that the likelihood function is bounded (which is usually the case in practice), the sequence of parameter estimates $\mathbf{\Psi}^0, \mathbf{\Psi}^1, \ldots, \mathbf{\Psi}^k$ gives rise to a non-decreasing sequence $\mathcal{L}(\mathbf{\Psi}^{(0)}|X) \leq \mathcal{L}(\mathbf{\Psi}^{(1)}|X) \leq \ldots \leq \mathcal{L}(\mathbf{\Psi}^{(k)}|X)$ which must converge as $k \to \infty$. While ML estimates are stationary points, the inverse is not necessarily true. It is possible to reach a saddle point or even a minimum of the likelihood, but only under very special conditions which are usually not met in practice; further details regarding the convergence can be found in [79].

However, the fact remains that the EM algorithm is essentially a steepest ascent method and therefore is not guaranteed to reach a global maximum of the likelihood. For more complex shapes of the likelihood function, it is likely that the algorithm will only converge to a local maximum, dependent on the initial parametrization $\mathbf{\Psi}^{(0)}$ with which the algorithm is started. Therefore, it is advisable to execute the algorithm several times with different initial conditions, and then compare the performance of the resulting parameters with some model selection procedure.

2.6.3 EM algorithm applied to Cluster Weighted Models

As described in the general formulation in section 2.6.2, one scenario for using the EM algorithm is when the maximization of the likelihood becomes much easier by introducing an additional, albeit fictitious random variable which we can interpret as unobserved or hidden data.

In the case of CWMs, this unobserved random variable can be introduced by imagining that each sample $(\mathbf{x}_i, y_i)$ is in fact described by a *single cluster*, which gets chosen at random according to some probability.

Let $Z_i \in \{1 \ldots M\}$ be the label of the cluster that gave rise to $(\mathbf{x}_i, y_i)$. This variable Z_i serves as the unobserved random variable. The cluster weights w_m (2.34) are interpreted as the probabilities of the event $Z_i = m$ for all $m = 1 \ldots M$, which implies that the Z_i are distributed according to a multinomial distribution with the cluster weights w_m as parameters.

If one would know from which cluster each sample $(\mathbf{x}_i, y_i)$ was generated, that is, if Z_i were known for all i, calculating an estimate for the cluster model parameters through maximum likelihood would be straightforward. For example, each cluster center would be simply given by the mean of all the points with a certain label. However, since we do not have this data, estimating the cluster parameters through maximum likelihood becomes a constrained nonlinear optimization problem, which

is usually difficult to solve directly. For such problems involving latent variables, the EM algorithm is an elegant and efficient technique.

The realizations of the Z_i are written as the indicator vectors $\mathbf{z}_i = (z_{i1}, \ldots, z_{iM})^T$, where

$$z_{im} = \begin{cases} 1, & \text{if data point } (\mathbf{x}_i, y_i) \text{ belongs to cluster } m, \\ 0, & \text{otherwise.} \end{cases}$$

We write the augmented data set (or training set) as

$$\mathbf{\Omega}_c = \{(\mathbf{x}_1, y_1, z_1), \ldots, (\mathbf{x}_N, y_N, z_N)\}$$

and the complete log-likelihood $\mathcal{L}_c$ as

$$\mathcal{L}_c(\mathbf{\Omega}_c|\mathbf{\Psi}) = \sum_{i=1}^{N} \sum_{m=1}^{M} \delta_{z_i,m} \log p_m(y_i|\mathbf{x}_i) p_m(\mathbf{x}_i) w_m \,,$$

where $\mathbf{\Psi}$ denotes the entire set of parameters of the cluster weighted model, namely the weights w_m, the positions and variances $\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m$ of the cluster centers as well as the parameters $\boldsymbol{\beta}_m, \Sigma_m$ of the local models.

The EM optimization starts with an initial estimate $\mathbf{\Psi}^{(0)}$ of these parameters. One possibility, which was also used in the following, is to start with uniform weights $w_m = 1/M$, random cluster position $\boldsymbol{\mu}_m$ (by simply picking M points randomly from the training data), and all variances $\boldsymbol{\sigma}_m$ set to the same value, so that the clusters roughly cover the points of the training data.

As described in section 2.6.2, in the expectation step (E-step) of the algorithm, one has to compute the conditional expectation of $\mathcal{L}_c$ with respect to the current parameter estimate, which in the first iteration is given by $\hat{\mathbf{\Psi}}$, giving rise to the following Q-function:

$$Q(\mathbf{\Psi}; \hat{\mathbf{\Psi}}) = \mathrm{E}_{\hat{\mathbf{\Psi}}} \left\{ \mathcal{L}_c(\mathbf{\Omega}_c|\mathbf{\Psi}) | \mathbf{x}, y \right\} . \tag{2.57}$$

Since the terms in the logarithm depend solely on $\mathbf{x}_i$ and y_i, the conditional expectation affects $\delta_{z_i,m}$, only.

It follows that the E-step is effectively reduced to a calculation of the conditional expectation of $\delta_{z_i,m}$, given the observed data. We introduce the abbreviation

$$q_m(\mathbf{x}, y; \hat{\mathbf{\Psi}}) := \mathrm{E}_{\hat{\mathbf{\Psi}}}(\delta_{z_i,m}|\mathbf{x}, y) = \P_{\hat{\mathbf{\Psi}}}(Z_i = m|\mathbf{x}_i, y_i),$$

where $\P_{\hat{\Psi}}$ is the probability for parameters $\hat{\mathbf{\Psi}}$. According to the definitions from section 2.6, the posterior probability is in general given by

$$q_m(\mathbf{x}, y; \hat{\mathbf{\Psi}}) = \frac{p_m(y, \mathbf{x})}{p(y, \mathbf{x})} = \frac{p_m(y|\mathbf{x}) p_m(\mathbf{x}) w_m}{\sum_{l=1}^{M} p_l(y, \mathbf{x})} = \frac{p_m(y|\mathbf{x}) p_m(\mathbf{x}) w_m}{\sum_{l=1}^{M} p_l(y|\mathbf{x}) p_l(\mathbf{x}) w_l} \,, \tag{2.58}$$

where it is understood that $p_m(y|\mathbf{x})$, $p_m(\mathbf{x})$ depend on $\hat{\mathbf{\Psi}}$, and w_m is in fact part of $\hat{\mathbf{\Psi}}$. This distribution relates each cluster to each data point. Looking at the resulting

fraction, one can see that the posterior is the ratio between one and all clusters predicting one specific point. Given this expectation value, the Q-function is given by

$$Q(\mathbf{\Psi};\hat{\mathbf{\Psi}}) = \sum_{i=1}^{N}\sum_{m=1}^{M} q_m(\mathbf{x},y;\hat{\mathbf{\Psi}}) \log p_m(y_i|\mathbf{x}_i) p_m(\mathbf{x}_i) w_m \,.$$

In the maximization step (M-step), one obtains the next parameter estimate $\check{\mathbf{\Psi}}$ by a global maximization of the Q-function with respect to $\mathbf{\Psi}$ over the parameter space. The derivatives with respect to the desired parameters can be readily calculated and set equal to zero, thus obtaining a set of equations for the new parameters $\check{\mathbf{\Psi}}$ as functions of the old parameters $\hat{\mathbf{\Psi}}$. This procedure is iterated until convergence.

In order to compute the new parameters, it is convenient to decompose the Q-function as follows:

$$\begin{aligned} Q(\mathbf{\Psi};\hat{\mathbf{\Psi}}) = &\sum_{i=1}^{N}\sum_{m=1}^{M} q_m(\mathbf{x},y;\hat{\mathbf{\Psi}}) \log p_m(y_i|\mathbf{x}_i) \\ &+ \sum_{i=1}^{N}\sum_{m=1}^{M} q_m(\mathbf{x},y;\hat{\mathbf{\Psi}}) \log p_m(\mathbf{x}_i) \\ &+ \sum_{i=1}^{N}\sum_{m=1}^{M} q_m(\mathbf{x},y;\hat{\mathbf{\Psi}}) \log w_m \,. \end{aligned}$$

This decomposition is useful, as the local function parameters $(\boldsymbol{\beta}_m, \mathbf{\Sigma}_m)$ appear only in the first, the cluster parameters $(\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m)$ only in the second, and the weights w_m only in the last summand. Thus, equations for the new parameters $\check{\mathbf{\Psi}}$ turn out to be largely decoupled. The cluster weights w_m, for example, can be calculated independently of the other cluster parameters $(\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m, \boldsymbol{\beta}_m, \mathbf{\Sigma}_m)$. Since the weights are subject to the constraint $\sum w_m = 1$, we have to introduce a Lagrange multiplier λ and solve

$$\frac{\partial}{\partial w_m}\left[Q(\mathbf{\Psi};\hat{\mathbf{\Psi}}) + \lambda(1 - \sum_{l=1}^{M} w_l) \right] = 0 \,,$$

which, after elementary calculations, leads to

$$\check{w}_m = \frac{1}{N}\sum_{i=1}^{N} q_m(y_i,\mathbf{x}_i;\hat{\mathbf{\Psi}}) \,,$$

which can be interpreted as a "soft" version of $\sum_i \delta_{z_i,m}/N$, where the unknown labels are substituted with their expectation value.

The updated estimates of the means and variances of the clusters $(\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m)$ are also derived by maximizing $Q(\mathbf{\Psi};\hat{\mathbf{\Psi}})$. Thus, the updated means are given by

$$\check{\boldsymbol{\mu}}_m = \frac{\sum_{i=1}^{N} \mathbf{x}_i \, q_m(y_i,\mathbf{x}_i;\hat{\mathbf{\Psi}})}{\sum_{i=1}^{N} q_m(y_i,\mathbf{x}_i;\hat{\mathbf{\Psi}})}, \tag{2.59}$$

which can be written in a more condensed form by defining the *cluster weighted expectation* of a function $\phi(\mathbf{x})$ as

$$\langle \phi(\mathbf{x}) \rangle_m \equiv \frac{\sum_{i=1}^N \phi(\mathbf{x}_i)\, q_m(y_i, \mathbf{x}_i; \hat{\boldsymbol{\Psi}})}{\sum_{i=1}^N q_m(y_i, \mathbf{x}_i; \hat{\boldsymbol{\Psi}})}\,, \tag{2.60}$$

so that the new cluster means are given by $\langle \mathbf{x} \rangle_m$. In the same way, the new variances can be written as

$$\check{\boldsymbol{\sigma}}_m = \sqrt{\langle (\mathbf{x} - \check{\boldsymbol{\mu}}_m)^2 \rangle_m}\,. \tag{2.61}$$

Initial parameter estimate

In the beginning, the EM algorithm needs an initial parameter estimate $\boldsymbol{\Psi}^{(0)}$, which must be provided by the user. It is this parameter estimate which determines the resulting model; the algorithm itself does not have any stochastic component which could lead to different parameter estimates when the same initial values are chosen. This is important when one would like to build *model ensembles*, consisting of several CWMs.

The usual procedure is to start with with uniform weights $w_m = 1/M$ and the cluster centers $\boldsymbol{\mu}_m$ chosen randomly, simply by picking M points from the training data set. The initial variances $\boldsymbol{\sigma}_m$ are chosen so that the clusters roughly cover the whole data set; therefore, the initial values for $\boldsymbol{\sigma}_m$ should depend on the variance of the given data.

The parameters of the cluster functions are initialized by setting all $\beta_{m,i} = 1$ and the output variances Σ_m roughly equal to the variance of the target values.

Optimization of cluster functions

For updating the parameters $\boldsymbol{\beta}_m$ of the cluster functions (2.41), we maximize for each cluster m the log-likelihood with respect to $\boldsymbol{\beta}_m$ [70], that is, we must solve

$$\frac{\partial}{\partial \boldsymbol{\beta}_m} \log \prod_{i=1}^N p(y_i, \mathbf{x}_i) = 0\,, \tag{2.62}$$

leading to

$$\begin{aligned} 0 &= \sum_{i=1}^N \frac{1}{p(y_i, \mathbf{x}_i)} p_m(y_i, \mathbf{x}_i) \frac{y_i - f(\mathbf{x}_i, \boldsymbol{\beta}_m)}{\Sigma_m^2} \cdot \frac{\partial f(\mathbf{x}_i, \boldsymbol{\beta}_m)}{\partial \boldsymbol{\beta}_m} \\ &= \frac{1}{N\, w_m} \sum_{i=1}^N p(w_m | y_i, \mathbf{x}_i)\, [y_i - f(\mathbf{x}_i, \boldsymbol{\beta}_m)] \cdot \frac{\partial f(\mathbf{x}_i, \boldsymbol{\beta}_m)}{\partial \boldsymbol{\beta}_m} \\ &= \left\langle [y - f(\mathbf{x}, \boldsymbol{\beta}_m)] \cdot \frac{\partial f(\mathbf{x}, \boldsymbol{\beta}_m)}{\partial \boldsymbol{\beta}_m} \right\rangle_m\,, \end{aligned} \tag{2.63}$$

where we used the definition (2.60).

Since we use linear parametrized local models (2.41), we obtain

$$\begin{aligned} 0 &= \langle [y - f(\mathbf{x}, \boldsymbol{\beta}_m)]\, f_j(\mathbf{x}) \rangle_m \\ &= \langle y\, f_j(\mathbf{x}) \rangle_m - \sum_{i=1}^{I} \boldsymbol{\beta}_{m,i}\, \langle f_j(\mathbf{x})\, f_i(\mathbf{x}) \rangle_m \ . \end{aligned} \tag{2.64}$$

For each cluster m, we define the matrix $(\mathbf{B}_m)_{ij} = \langle f_j(\mathbf{x})\, f_i(\mathbf{x}) \rangle_m$ and the vector $(a_m)_j = \langle y\, f_j(\mathbf{x}) \rangle_m$, leading to the following simple update rule for the new cluster model parameters:

$$\boldsymbol{\beta}_m^{\text{new}} = \mathbf{B}_m^{-1} \cdot \mathbf{a}_m \ . \tag{2.65}$$

For dealing with singular or ill-conditioned matrices $\mathbf{B}_m$, an additional regularization procedure should be performed before or during the calculation of the matrix inverse. Further details on this topic are described later in section 4.2.

The updated output variances are now given by

$$\Sigma_m^{2,\text{new}} = \left\langle [y - f(\mathbf{x}, \boldsymbol{\beta}_m)]^2 \right\rangle_m \ . \tag{2.66}$$

2.7 Example: Noise reduction

A possible application for the cross prediction introduced in section 2.2.1 is the reduction of measurement noise from a deterministic dynamical system [9]. For this purpose, a noiseless time series from this dynamical system is necessary and a second time series is generated by corrupting the noiseless data with additive white noise. Afterwards, a model is trained to predict from this noisy time series to the noiseless one. This model can then be used as a tool for noise reduction on before unseen noisy data from the same dynamical system, given that the statistical properties of the noise are similar.

In our example, we want to reduce noise from the Rössler system, defined by the following system of ODEs:

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + a \cdot y \\ \dot{z} &= b + z \cdot (x - c) \ . \end{aligned} \tag{2.67}$$

For training, we generate a time series with 30,000 points and add white noise with a SNR of 10 dB. The embedding parameters obtained while training the local (nearest neighbor) model (see section 2.5.6) are also used for the the Cluster Weighted Model.

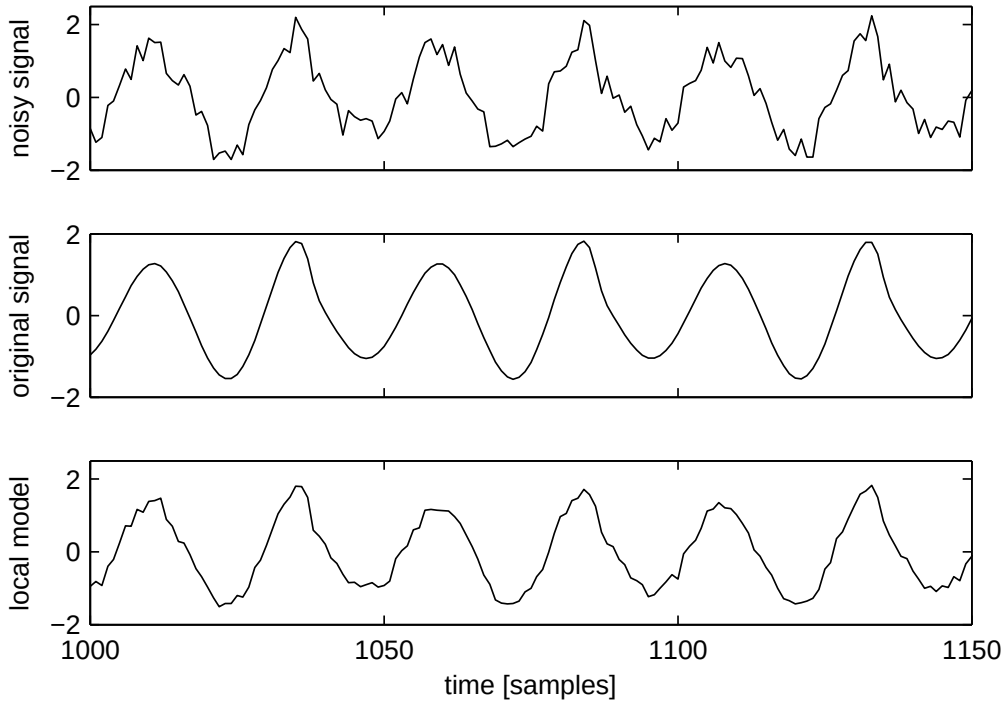


Figure 2.6: Noisy input (top), original signal (middle) and local model prediction (bottom). The CWM prediction looks almost identical.

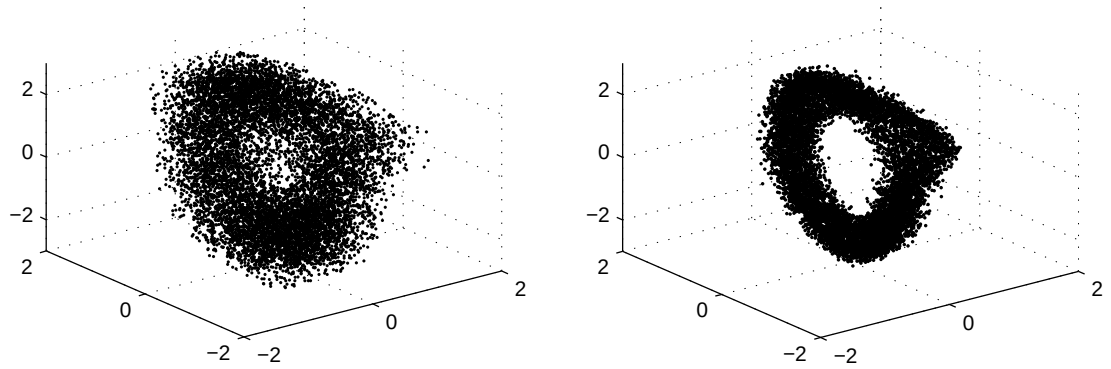


Figure 2.7: Original noisy attractor (left) and local model test data prediction (right)

Through the prediction, the SNR could be raised to 18 dB. The attractors reconstructed by an 3D embedding of the original and the predicted test data can be seen in figure 2.7. An example for the prediction of the local model is shown in figure 2.6.

2.8 Example: Signal through chaotic channel

Closely related to the previous example, where we subtracted measurement noise from a dynamical system, we now want to reconstruct a signal which is sent through a chaotic dynamical system, where the signal can be seen as a special case of dynamical noise. In our numerical example, a music wave file is taken as the signal and the Lorenz system as the chaotic system. The signal is added to the first ODE, while the y variable is taken as the output (see figure 2.8).

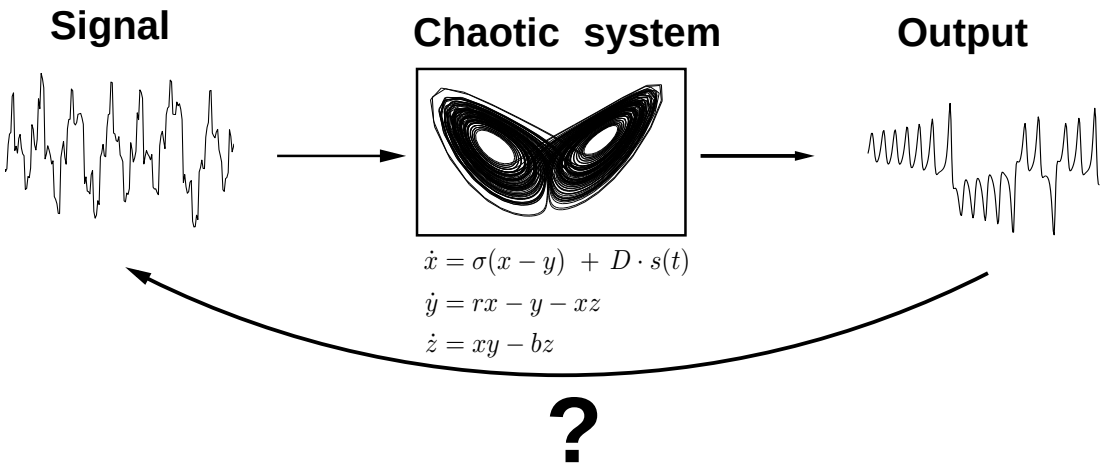


Figure 2.8: Illustration of chaotic channel

We now want to construct a model which is able to predict the original signal given the output, without providing any a-priori knowledge of the underlying dynamical system. Like the previous example, this is the case of a cross prediction without feedback. In the following example, the model is trained using 30,000 point pairs, consisting of the original signal and the output of the chaotic system.

First, we train a locally linear model which also yields good embedding parameters, which are also used for training a Cluster Weighted Model with locally quadratic functions. Both models are tested on 10,000 test data points. The locally linear model has a NMSE of 11.5%, while the Cluster Weighted Model (200 clusters) performed slightly better with a NMSE of 9.8%. The latter result can be seen in figure 2.9. Although the NMSE is quite large, which can also be observed in the

plot of the residuals, the model still shows a good reconstruction of the basic signal properties.

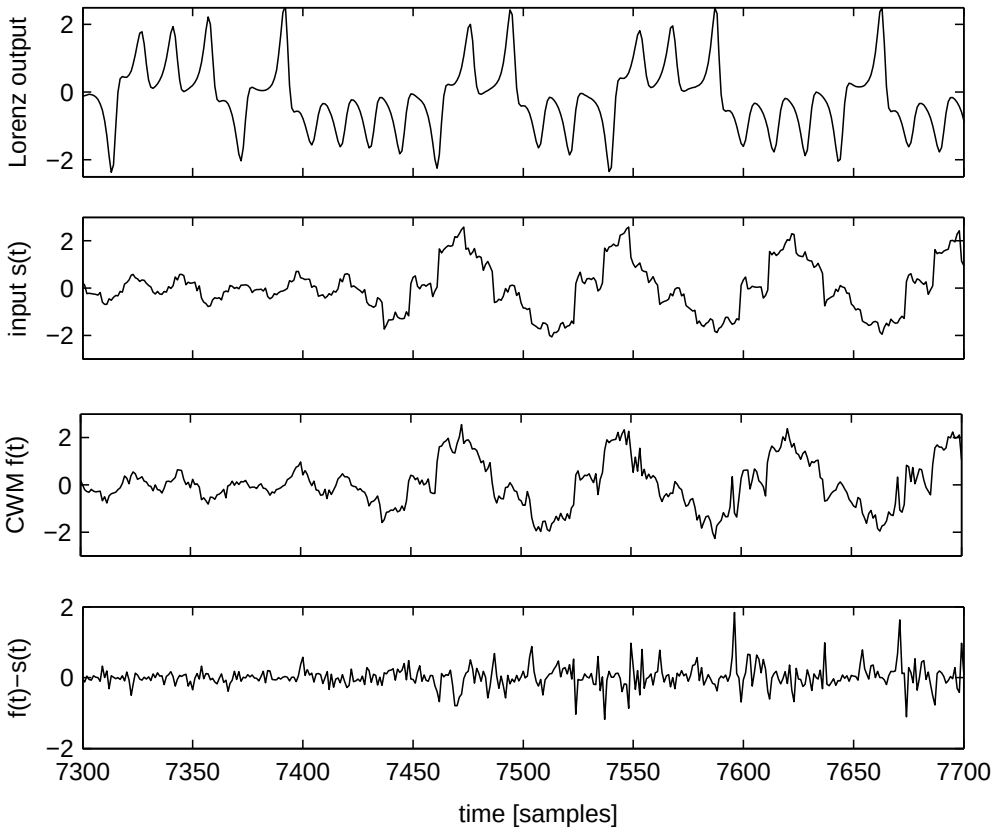


Figure 2.9: Prediction of test data for music signal using a Cluster Weighted Model. The first two plots show the output signal from the Lorenz system and the original input signal. The lower two plots show the CWM prediction and the residuals. The result from the local model looks almost the same.

2.9 Example: Friction modeling

Friction is a very complex and nonlinear phenomenon, comprising various regimes and behavioral facets. While there exist numerous analytical approaches for describing different aspects of friction phenomena, a model which could explain all aspects of friction is still missing. In practical control applications where high accuracy is demanded, the highly nonlinear dependence of the friction force on displacement is one of the main problems. Black-box models, which do not depend on any a-priori physical knowledge, can help to deal with this problem.

Experimental friction data, obtained from an experimental setup done by Al-Bender, Lampaert and Tjahjowidodo at the University of Leuven [60], is used to train Local Models as well as Cluster Weighted Models. The data consists of the (desired) displacement $P(t)$ for the model input and the friction force $F(t)$ (to be applied) for the model output. Therefore, we have again a cross prediction from $P(t)$ to $F(t)$, but in this case the accuracy of the modeling can be greatly improved by adding past values $F(t-\delta)$ of the friction force to the input vector, introducing a feedback into the modeling procedure. The training data set consisted of 90,000 data points and the models were tested on 20,000 points. Here, the models are freely iterated over the complete test data set, i.e. while the position values in the input vector are always exact, the friction force is always estimated (except for the starting value, which is also exact).

Like in the previous examples, we first trained the locally linear model to obtain good embedding parameters. In this case, the result was the following 5D embedding vector

$$\mathbf{x}(t) = (P(t), P(t-16), P(t-66), P(t-67), F(t-19)) \ , \tag{2.68}$$

therefore consisting of four position values and one past force value. It is important to note that the optimal delay for the past force value (in this case $\delta = 19$) can only be obtained through an optimization which depends on the multi-step prediction error. Since the time series is very densely sampled, the optimization on the 1-step prediction error would yield an "optimal" value for the delay of $\delta = 1$, with the model simply repeating the last force value. Of course, such a model will lead to bad prediction results when freely iterated over the test data set.

Another important effect of the multi-step prediction error is the better stability of the final model during iteration over several steps. In fact, as our tests show, the last position value $P(t-67)$ is crucial for the stability of the local model, though it may first seem redundant since it is almost equal to the previous one as they are only separated by a delay of one. However, even with this additional position value, the Cluster Weighted Model could not produce stable results when iterated over the test data, since it tends to oscillate with a period given by the delay of the past force value. Although one can enforce stability by simply clipping the model output with the minimum and maximum value of the given output data from the training set, the model error gets quite large. While it is possible to dampen the oscillations through filtering, the filter introduces new parameters (order, cut-off frequency) which somehow have to be optimized.

Our approach for solving this problem is to use not one, but three different CWMs, each having a slightly different delay for the past force value (in this case we used $17, 19$ and 21). This is called a *model ensemble*, and it is well known that such ensembles can often lead to better predictions than each single model in this ensemble could provide [36], although in our case we are more interested in stability features. The additional position value $P(t-67)$ was now omitted, as it was not

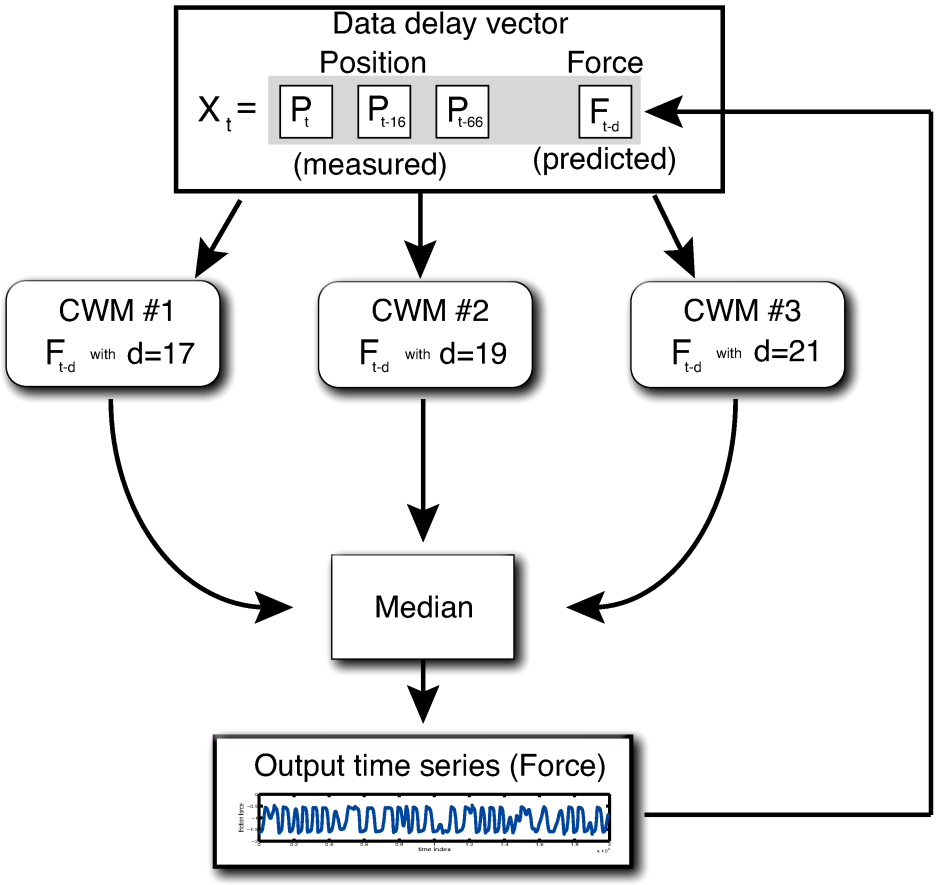


Figure 2.10: Schematic workflow of the CWM ensemble for friction prediction. The data vector consists of the measured position and one past predicted force value. Each CWM in the ensemble gets the same positional data but uses a different delay for the force value. The resulting output, which is also fed back into the ensemble, is the median of the three model values.

necessary for stability anymore and led in this case to slightly worse prediction results. Since every model has a different delay for the force value, each model will tend to oscillate with different periods. When predicting the test data, we first calculate the three model outputs for each point and simply take the median, i.e. in this case the output of the model lying between the other two. The median is fed back to all three models, practically dampening beginning oscillations. A schematic of this procedure is shown in figure 2.10; extending it to a larger number of models is straightforward, but didn't result in lower errors.

The local model (260 neighbors, linear weight function, euclidean distance, PCTR with soft threshold and $s_c = 3 \cdot 10^{-3}$ and $s_w = 0.67$) yields a NMSE of 1.01% over

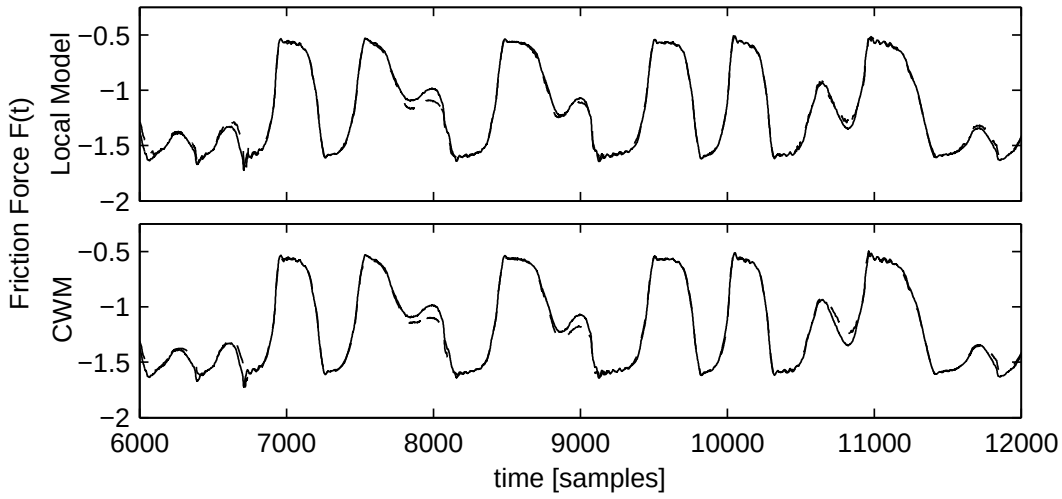


Figure 2.11: Local model (upper panel) and CWM prediction (lower panel) for a section of friction test data; predictions are given by dashed lines.

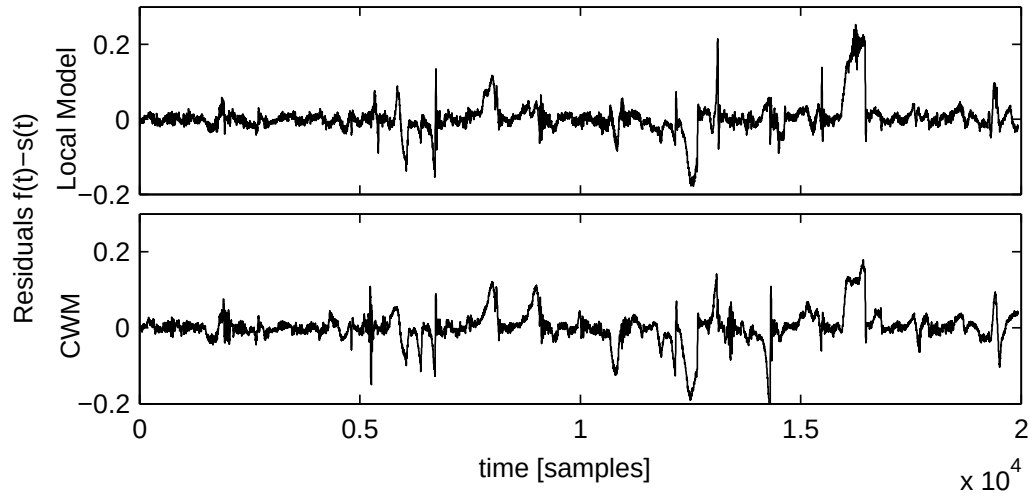


Figure 2.12: Residuals of local model (upper panel) and CWM prediction (lower panel) for complete friction test data.

the 20.000 test data points. The CWM ensemble, where each CWM used quadratic functions and 600 clusters, has almost the same performance with a NMSE of 1.05% (see figure 2.11). This ensemble error is lower than each of the single model outputs (though only slightly). The residuals for both models can be seen in figure 2.12.

2.10 Example: Chua's oscillator

In its original form, the Chua oscillator is an electronic circuit featuring a nonlinear resistor, which can be realized in different ways, usually using a combination of operational amplifier(s), diodes and linear resistors [18]. It can be described in a dimensionless form through the following system of ODEs:

$$\begin{aligned} \dot{x} &= \alpha\,(y - h(x)) \\ \dot{y} &= x - y + z \\ \dot{z} &= -\beta y \\ \text{with}\quad h(x) &= m_1 x + \frac{1}{2}(m_0 - m_1)\left[|x+1| - |x-1|\right] . \end{aligned} \tag{2.69}$$

The nonlinear resistor is described through the function $h(x)$, which is a piecewise-linear curve. With parameters $\{\alpha, \beta, m_0, m_1\} = \{9, 14.286, -1/7, 2/7\}$ this system features a chaotic attractor, the so called *double scroll*, which can be seen on the right-hand side of figures 2.14 and 2.15.

In the following, a time series consisting of 6000 points is used, generated numerically from (2.69) with a sampling rate of 15Hz. By embedding this time series in three dimensions with a delay of 10, we reconstruct the attractor and train a CWM to predict 50 time steps into the future (direct prediction, see section 2.2). Using a set of 500 training points which where not used for training, the model output in form of the resulting distribution is shown in the lower plot of figure 2.13. The upper plot was generated by integrating the ODE system with 50 slightly different initial values, all lying in a small neighborhood of the input vector from the test data. The value reached after 50 time steps was plotted, resulting in 50 points for each time step, generating a visual approximation of the actual distribution, which can be compared against the model.

We can see that both plots exhibit a similar behavior: small regions which are stable over the 50 time steps, interrupted by larger regions where the outcome is very uncertain, corresponding to the two different scrolls which the trajectories can reach (see right-hand side of figure 2.14).

An example for a stable region is time index 0, where we see a fairly small distribution of the estimated prediction result. Figure 2.14 shows the actual time series (using the x component of (2.69)), starting with slightly different initial values

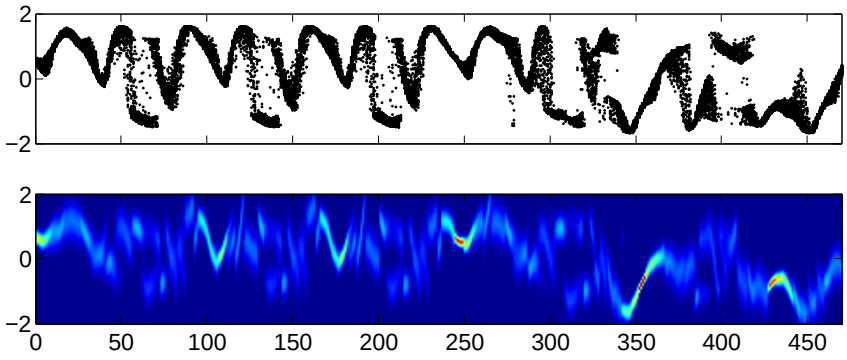


Figure 2.13: Upper panel: Estimation of empirical distribution of the 50-steps prediction for the Chua oscillator by starting with 50 slightly different initial conditions for each time step. Lower panel: Output in form of a distribution from Cluster Weighted Model, trained on 50-steps prediction.

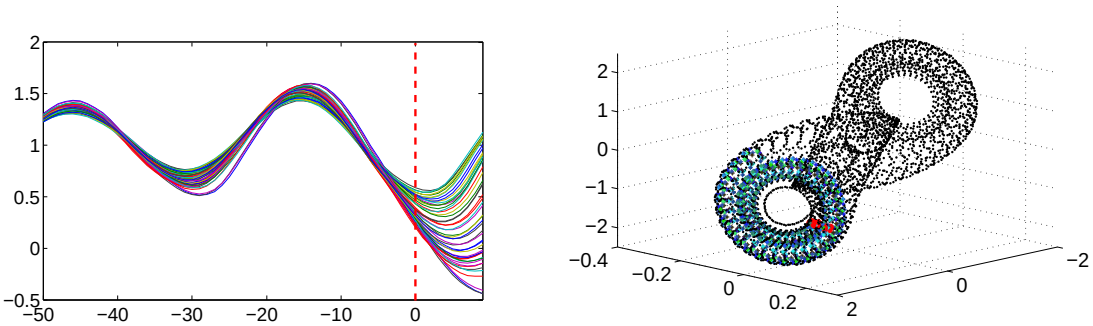


Figure 2.14: Example for a stable region on the Chua attractor. Shown are several trajectories with slightly different initial values, starting at time index -50. On the left the resulting time series is shown (x component), with the red dashed line at time 0 denoting the prediction step. The right plot shows some of the trajectories on the attractor, with the red circles denoting their starting positions.

at time index -50. We see that all trajectories stay in each others vicinity and remain on the same scroll, and begin to diverge just after the prediction goal of 50 steps.

Figure 2.15 shows an unstable region, beginning at time index 15 with the prediction goal at time step 65. We see that the trajectories diverge, with only some staying on the initial scroll and most others moving to the other. Those regions are visible in the CWM output in figure 2.13, sometimes as a bimodal distribution or in a rapid succession of lower/upper scroll probabilities.

In figure 2.16, we again see the Chua attractor, color-coded with the logarithmic variance of the CWM. It is important to note that we color-coded the *starting point* of the expected variance, meaning that when the point has a high variance (red),

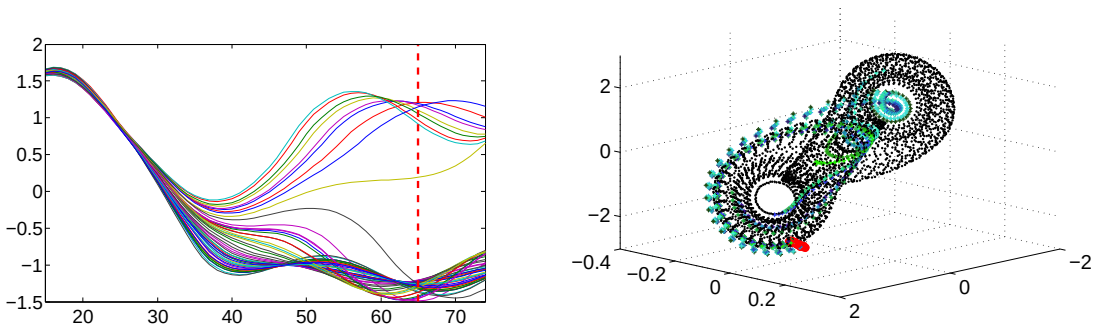


Figure 2.15: Same setup as in figure 2.14, but this time starting with time index 15, with most of the trajectories reaching the lower scroll. The effect is a bimodal distribution at the prediction step (time index 65).

trajectories starting from this region will most likely diverge in the next 50 time steps, whereas those with a low variance (blue) will most likely remain in each others vicinity. CWMs can therefore serve as a means to identify stable and unstable regions on the attractor for predicting a certain amount of steps into the future.

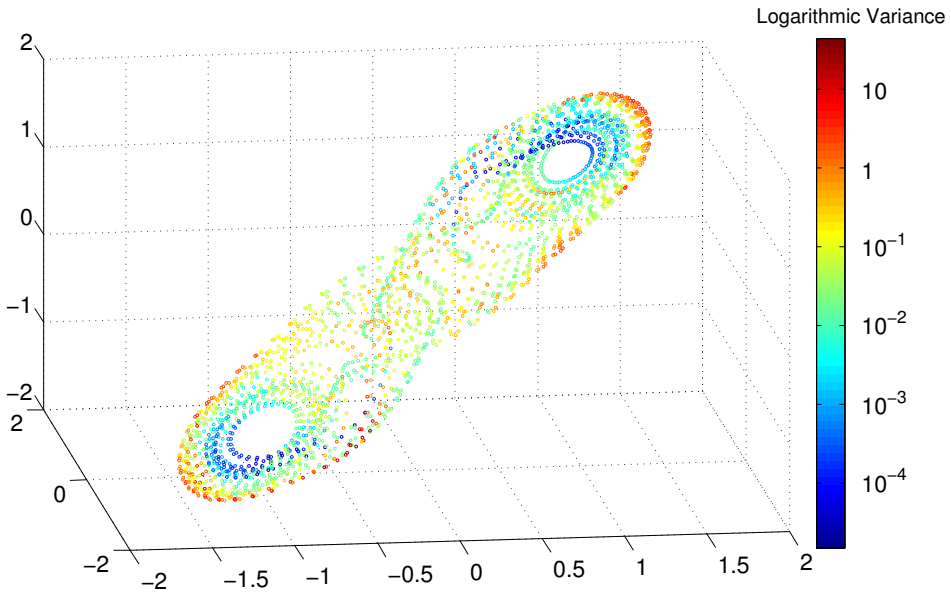


Figure 2.16: Chua attractor with color coded logarithmic variance (rescaled to the interval [0 10]) for the 50-steps prediction of the CWM.

Chapter 3

Probabilistic evaluation of Cluster Weighted Models

An interesting feature of CWMs is that they allow for an output in the form of an entire probability distribution. This feature is shared by quite a few other modeling approaches, such as kernel density estimators and various techniques based on nearest neighbors. In principle, probability assignments allow for a more informed decision making than "deterministic" or unequivocal forecasts, as the former allow to estimate the impending risks as well as the expected losses associated with particular decisions.

In the following chapter, the quality of probabilistic output from CWMs shall be investigated and compared to a couple of other approaches of varying complexity and requirements as to the knowledge about the underlying system that generated the data.

Several methods are examined which allow to produce forecasts for time series in the form of probability assignments. The necessary concepts are presented, addressing questions such as how to assess the performance of a probabilistic forecast. Two examples are presented. The first involves estimating the state of (numerically simulated) dynamical systems from noise corrupted measurements, a problem also known as filtering. There is an optimal solution to this problem called the optimal filter, to which the considered time series models are compared. (The optimal filter requires the dynamical equations to be known.) In the second example, we aim at forecasting the chaotic oscillations of an experimental bronze spring system.

The work presented in this chapter was done in collaboration with Dr. Jochen Bröcker from the Max Planck Institute for the Physics of Complex Systems in Dresden. Dr. Bröcker provided the concept of probabilistic scores [16], the calculations for the invariant measure and the Interacting Particle Filter, while the author did the calculations for the CWMs and the local/global density estimation approach. Regarding the collaboration in the textual content, Dr. Bröcker wrote the introductory section 3.2 on probabilistic scores, which is repeated in this thesis since otherwise this chapter could not be properly understood. He was also largely responsible for presenting the concept of particle filters, with the details moved to the appendix.

3.1 Introduction

This chapter aims at presenting a framework for the probabilistic assessment of time series models. Using this framework, the quality of probabilistic output from CWMs and several other approaches which generate probabilistic forecasts is investigated and compared.

Evaluating probabilistic forecasts requires different measures of performance than evaluating deterministic forecasts. The mathematical concept of *scores* provides appropriate performance measures for probabilistic forecasts. They were developed in stochastics and econometrics to elicit personal probabilities [68]. See for example [16, 31, 56] for more information on this concept. Furthermore, scores appear in various guises and under various names in several branches of stochastics and related fields, and have received considerable attention in the weather forecasting community as the appropriate tool to evaluate probabilistic weather forecasts [56].

In this chapter, the performance of Cluster Weighted Models is analyzed by means of scores and compared to other probabilistic forecast systems of varying complexity and requirements as to the knowledge about the underlying system that generated the data. As a sample problem, we consider the problem of *state estimation*, which aims at reconstructing the internal state of a dynamical system from noisy observations. In weather forecasting and econometrics, this problem is often referred to as "now-casting", since the state to be reconstructed is in fact the present state of the system. However, we will keep using the term "forecasting" for this chapter, since it is more commonly used in physics. Like in the examples presented so far, when CWMs are applied to this problem, the dynamics of the underlying system are not assumed to be known, making this a pure black-box approach. One might wonder to what use the internal state of a dynamical system might be when the dynamical equations are unknown, in which case we would, for example, be unable to generate predictions of the dynamical system's future behavior. The internal state of a dynamical system might be of interest for its own sake, for example for monitoring purposes and fault detection. Another motivation was to have a problem for which there exist other established approaches. In particular, it is known that the interacting particle filter represents an (asymptotically) optimal solution to this problem, but in contrast to CWMs, one needs knowledge of the underlying equations to use this method. While comparing these two methods must obviously be considered "unfair", we can use the results from interacting particle filter as a benchmark, giving us an estimate of how close the purely data-based methods are to an optimal solution. As a further example, we consider experimental time series from a bronze spring subjected to an alternating magnetic field, whereby the spring exhibits chaotic oscillations. Here, the probabilistic forecasting systems are employed to predict the future evolution of the spring's motion.

3.2 Probabilistic Forecasts and Scoring

In this section the reader will be provided with a brief introduction to probabilistic forecasting. Suppose there is a quantity Y, referred to as the *verification*[1], the exact value of which is unknown (at least at the time we would like to know it). A *probabilistic forecast* is a probability assignment to possible outcomes of Y. If Y is continuous, probabilistic forecasts can be expressed by means of a density function $p(y)$, or alternatively by the corresponding cumulative distribution function. As the latter becomes awkward to handle in higher dimensions, density functions will be used exclusively in the following. The range of possible values of Y could also be discrete, in which case Y labels one of a finite number of possible events. Yet different types of Y (e.g. combined continuous-discrete) are often considered. In the following though, we will focus on continuous and multivariate verifications.

The exact shape of $p(y)$ should depend on information at hand about the unknown Y, be it expert knowledge, superstition or observations of quantities assumed to be related to Y. The question is how to fold the available information into a probabilistic forecast. Ideally, the probabilistic forecast $p(y)$ should be equal to the conditional probability of the verification Y, given the available information. However, in order to form this conditional probability, the relationship between the information and the verification Y has to be known. Even in the ideal situation where this is the case, $p(y)$ might be constrained by available computational resources. Therefore, probabilistic forecasting usually constitutes a modeling problem, similar to its deterministic counterpart.

The basic requirements for forming probabilistic forecasts from various information sources are very similar to those required for forming deterministic forecasts. The main building block is a suitable class of models, capable of producing probabilistic forecasts, as well as a mechanism to chose a "good" model, where it remains to be defined what "good" means in these circumstances. Although expert judgment is often indispensable for choosing models, objective criteria of forecast performance that can be evaluated numerically become necessary as soon as computational methods are employed in model design. In other words, quantitative measures of "goodness" are needed. This is reminiscent of deterministic forecasting, where statistical learning and numerical design of black-box models, usually in combination with expert knowledge, have become common in almost all fields where forecasts are required. In statistical learning, models are tuned by optimizing risk functionals [36]. While the model classes used in this chapter will be subject to section 3.3, the remainder of this section will focus on the evaluation of probabilistic forecasts, or in other words on quantitative measures of forecast performance.

To evaluate probabilistic forecasts, we will employ the mathematical concept of scores, see for example [12,16,31,34,68]. A score is a function $S(p(\cdot), Y)$, which takes

[1] Alternative names for Y could be "observation" or "target". In view of a particular example to be discussed in section 3.3.1, where a different kind of observation is used to construct the forecast, we use the term verification, which is also common in the weather forecasting community.

the forecast $p(\cdot)$ as its first and the verification Y as its second argument. The score effectively compares the forecast $p(\cdot)$ with the verification Y. As the forecast $p(\cdot)$ is a function, the score in general depends on the entire functional shape of $p(\cdot)$ and hence is more accurately described as an operator on the forecast. An example of a score is the *Ignorance* [16, 31, 34]

$$S(p(\cdot), Y) := -\log(p(Y)) \tag{3.1}$$

that will be used in the following to evaluate the performance of different forecasting methods. Another example is the *Proper Linear Score* [16, 31] (to our knowledge first discussed in [12] for the binary case)

$$S(p(\cdot), Y) := \int p(y)^2 \mathrm{d}y - 2p(Y) \tag{3.2}$$

which depends on the shape of the entire function $p(\cdot)$ (through the integral in (3.2)), while the Ignorance depends only on the value of $p(\cdot)$ at Y, that is, the verification that eventually obtains. This property of the Ignorance is called *locality*.

In the definition of the Ignorance and the Proper Linear Score, it is understood that a smaller score indicates a better forecast. The reason for this convention, which might not be consistent with the usage of the word "score" as used in ordinary parlance, will become clear later. With this convention in mind, the rationale behind the Ignorance is easily discerned. The Ignorance is small as soon as the forecast $p(\cdot)$ assigns a high value to the verification Y. A similar argument applies to the second term $-2p(Y)$ of the Proper Linear Score. The first (presumably less intuitive) term of the Proper Linear Score is necessary to render the score *proper*. Propriety of scores is a criterion that is indispensable to obtain consistent results, as we will argue now. Any score (be it a probabilistic or a deterministic one) should yield an optimum *expected* value if the forecast is the desired one. In the case of (probabilistic) scores, we require that if Y is drawn from $p(\cdot)$, then $p(\cdot)$ should in fact be the unique minimizer of the expected score, that is, any other forecast $q(\cdot)$ should get a larger (i.e. worse) score than the expected score of $p(\cdot)$. This statement can be written as

$$\int S(q(\cdot), y) p(y) \mathrm{d}y \geq \int S(p(\cdot), y) p(y) \mathrm{d}y. \tag{3.3}$$

We speak of a *proper score* if this inequality indeed holds for any $p(\cdot)$ and $q(\cdot)$, while we call a score *strictly proper* if there is a strict inequality in (3.3) unless $p(\cdot) = q(\cdot)$. Equation (3.3) can be rewritten as

$$\int \left[S(q(\cdot), y) - S(p(\cdot), y) \right] p(y) \mathrm{d}y \geq 0, \tag{3.4}$$

with equality only if $p(\cdot) = q(\cdot)$. This form makes explicit that every strictly proper score defines a measure of discrepancy or distance between $p(\cdot)$ and $q(\cdot)$, given by

the left hand side of (3.4). The fact that distances are usually positive motivates the convention made earlier that scores be negatively oriented. Propriety is a mathematical property of the score itself. It can be shown that both the Ignorance and the Proper Linear Score are indeed strictly proper. In case of the Ignorance, it is easily seen that the left hand side of (3.4) is in fact the Kullback-Leibler divergence, which is well-known to be positive definite. In case of the Proper Linear Score, the left hand side of (3.4) is the mean squared difference between $p(\cdot)$ and $q(\cdot)$. The name "Proper Linear Score" is motivated by the fact that the intuitively appealing *Naïve Linear Score* $-p(Y)$ is not proper itself but can be rendered proper by adding the first term in (3.2) [16]. For a detailed discussion of scores and the rationale of propriety see [14, 16, 31].

When evaluating forecast *systems*, one is not only concerned with a single density function $p(\cdot)$ but rather with an entire archive of probability density functions $p_n(\cdot)$ and corresponding verifications Y_n. This archive can be employed to estimate the performance of the forecast system. Similar to statistical learning, where the empirical risk is used to value the forecast system, we define the *empirical score* (with respect to a proper scoring rule S) as

$$S_N := \frac{1}{N} \sum_{n=1}^{N} S(p_n(\cdot), Y_n). \tag{3.5}$$

The empirical score values the performance of the forecast system over all forecast-verification pairs $(p_n(\cdot), Y_n)$ in the archive. It should be noted that when viewed as a function of p, the empirical Ignorance score is equal to the negative log-likelihood.

A few remarks on the connections between the presented scoring approach and the widely applied mean squared error will conclude this section. Various prediction algorithms employed in deterministic forecasting (a.k.a. forecasting nonlinear time series) first explicitly or implicitly construct an estimate of the conditional probability of the verification given the observation, but then keep only the mean value, throwing away all additional information. Evaluating such a deterministic forecast with the mean squared error amounts to using a not strictly proper score. Furthermore, even the mean value of the correct conditional density is a forecast with often rather undesirable properties. This can be seen as follows. If again $p_n(\cdot), n = 1 \ldots N$ is a series of probabilistic forecasts, a deterministic forecast is obtained by considering the mean

$$m_n := \int y \, p_n(y) \mathrm{d}y \tag{3.6}$$

of the forecast distribution.

This deterministic forecast can be evaluated by the ordinary *mean squared error*

$$S_N := \frac{1}{N} \sum_{n=1}^{N} (Y_n - m_n)^2. \tag{3.7}$$

Substituting from (3.6) for m_n in (3.7), it turns out that the mean squared error can be interpreted as a score with the score function

$$S(p(\cdot), Y) = (Y - \int y\, p(y) \mathrm{d}y)^2 . \tag{3.8}$$

Using the well known minimum property of the mean it can be shown that the mean squared error is a proper score (i.e. the relation 3.3 holds). It is however not strictly proper, as any two p, q with the same mean render relation (3.3) an equality. In other words, the mean squared error does not take into account any other aspects of the forecasts than the mean. This is appropriate if the mean is the only aspect we are interested in. Outside the linear context though this is rarely, if ever, the case. If for example $p_n(\cdot)$ is a distribution of possible states on the attractor of a dynamical system, the mean m_n is not necessarily a possible state on this attractor. Even worse, the sequence $m_n, n = 1 \ldots N$ almost never forms an orbit either of the system itself or any other similar system. More generally, the mean m_n is usually not a "typical" or "likely" value of $p_n(\cdot)$ in any sense.

3.3 Probabilistic Forecasting Schemes

This section provides a brief outline of several probabilistic forecasting methods. The discussed methods vary in terms of their complexity, computational demands, pre-knowledge about the underlying process, and other requirements. Nontrivial trade offs between performance and, for example, computational costs are not untypical for realistic situations. From a practical point of view, the required resources and implementational constraints of a particular method are therefore as important as its performance. Hence, a comparison of state estimation methods that focused on the performance alone would be incomplete.

As a first probabilistic forecasting system we discuss the invariant measure. The invariant measure can be considered as a rather cheap and low skill forecast system. It is built entirely upon average dynamical behavior of the system and does not take into account any on-line observations. Further details of this approach are explained in section 3.3.2. In section 3.3.3, a class of forecasting methods is briefly presented which is based on global and local density estimators approximating joint densities $p(y, x)$ in delay embedding space. Finally, as a sort of ideal probabilistic forecast system, the *optimal nonlinear filter* [15] is considered. The optimal nonlinear filter comprises dynamical equations for the conditional probability of the underlying state, given the time series of past and present observations. For this approach, the equations of state of the system need to be known. Building the optimal nonlinear filter though is known to be a formidable problem, suffering from the notorious curse of dimensionality. An approximative solution is provided by the *interacting particle filter*, of which several variants and flavors have been proposed. This approach is

briefly explained in section 3.3.4, with the technical details being deferred to the appendix.

It is probably worth mentioning at this point that estimators of the joint density $p(\mathbf{y}, \mathbf{x})$ of input vectors $\mathbf{x}$ and outputs $\mathbf{y}$ automatically provide estimates of the conditional density by means of the Bayes formula

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}, \mathbf{x})/c(\mathbf{x}) \qquad \text{with} \qquad c(\mathbf{x}) = \int p(\mathbf{y}, \mathbf{x}) \mathrm{d}\mathbf{y}, \tag{3.9}$$

a fact used by several forecasting schemes to be presented. As discussed in the introduction, the forecasting systems are to be employed to estimate the unknown state of a dynamical system from noisy observations. How we intend to use density estimators for this purpose is outlined in the next subsection. The details of the individual forecasting systems are subject to the following subsections.

3.3.1 Application of density estimators to state estimation

Let $\mathbf{Y}_n \in \mathbb{R}^d, n \in \mathbb{N}$ be the orbit of a dynamical system

$$\mathbf{Y}_{n+1} = F(\mathbf{Y}_n),$$

where the initial condition of Y_n is unknown and distributed according to the probability measure P_0[2]. The *observation process* Z_n is defined as

$$Z_n = \mathbf{c}\mathbf{Y}_n + \sigma R_n,$$

where $\mathbf{c}$ is a linear mapping from $\mathbb{R}^d$ to $\mathbb{R}$, σ a positive constant and R_n a process of independent and identically distributed random variables having density $\gamma(r)$ with unit variance.

The problem of state estimation (or data assimilation) is to obtain an estimate of the current state $\mathbf{Y}_n$, given only past and current observations, that is, observations $Z_{1:n} := \{Z_k, k = 1, \ldots, n\}$. Before CWMs (or any other black-box model) can be applied to this problem, suitable input vectors $\mathbf{X}_n \in \mathbb{R}^D$ have to be constructed from the scalar observations Z_n first. Again, as described in section 1.5.1, we use a delay embedding

$$\mathbf{X}_n := \left(Z_n, Z_{n-\delta}, \ldots, Z_{n-(D-1)\delta}\right) \tag{3.10}$$

with suitable dimension D and delay δ. Now suppose we have an archive of realizations $T := \{(\mathbf{x}_n, \mathbf{y}_n), n = 1 \ldots N\}$ of the delay vectors of observations $\mathbf{X}_n$ and the corresponding known states $\mathbf{Y}_n$. Using the $\mathbf{x}_n$ as inputs and the $\mathbf{y}_n$ as outputs, any density estimator can be applied to find an approximation of the joint

[2]Since our aim is to provide probabilistic estimates of the states $\mathbf{Y}_n$, they play the role of the verification. Hence using the letter $\mathbf{Y}$ for the states of the dynamical system is consistent with the notation used earlier. Also, in this chapter, the capitalized characters X, Y, Z denote random variables, and are set in bold if multi-dimensional.

probability $p(\mathbf{y}, \mathbf{x})$ and subsequently of the conditional probability $p(\mathbf{y}|\mathbf{x})$ by (3.9). Taking into account that the inputs $\mathbf{x}$ are delay vectors of observations, the algorithm is effectively a means to approximate the conditional probability density $p\left(\mathbf{Y}_n = \mathbf{y}_n|(Z_n, Z_{n-\delta}, \ldots, Z_{n-(D-1)\delta}) = \mathbf{x}_n\right)$.

3.3.2 The Invariant Measure

A simple and intuitively reasonable probabilistic forecasting strategy is to assign to any possible outcome its observed relative frequency, which obviously requires a large archive of samples of the variable in question in order to compute observed frequencies. Ideally, we should use the (unconditional) probability of $\mathbf{Y}_n$ for this strategy, or the invariant measure in case $\mathbf{Y}_n$ is in fact the orbit of a dynamical system. Employing the invariant measure as a probability forecast amounts to what is known in meteorology as "climatological rules". It should be noted that the system still has to meet certain requirements in order for this strategy to be applicable. For example, the system should have some (albeit weak) stationarity assumptions.

Since in the following the invariant measure will be employed as a performance benchmark for other forecasting systems that come in the form of probability density functions, it is convenient to represent the (approximation to the) invariant measure in the form of a probability density as well. This density was written as $c(x)$ in (3.9), and we will keep this notation, in order to distinguish it from a generic probability density function. Various techniques of considerable sophistication have been proposed to approximate densities from a sample of points, of which kernel estimators seem to be the best developed ones. An excellent introduction to this subject is [73]. The *kernel density estimate* (KDE) employed in the present work has the form

$$c(\mathbf{y}) = \frac{1}{N} \sum \frac{1}{\sigma_i^d} K\left(\frac{\mathbf{y} - \mathbf{y}_i}{\sigma_i}\right) \tag{3.11}$$

where K is the *kernel function*, $\mathbf{y}_i, i = 1 \ldots N$ is the archive of verifications and σ_i the *bandwidth*. We use exclusively Gaussian kernels in this chapter, that is

$$K(\mathbf{y}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\|\mathbf{y}\|^2\right). \tag{3.12}$$

This choice renders the kernel estimate $c(\mathbf{y})$ a positive and normalized density function. Choosing a good bandwidth σ_i is the tricky bit. We used the ansatz

$$\sigma_i = s \cdot \delta(y_i) \tag{3.13}$$

where $\delta(y_i)$ is the distance between $\mathbf{y}_i$ and its k'th nearest neighbor in the archive, with $k = \sqrt{N}$. The motivation for this choice is that all kernels should cover roughly the same amount of sample points. On the other hand, some drawbacks of the nearest-neighbor-estimate, discussed in [73], are thus avoided.

The only free parameter is the factor s, which was chosen by minimizing the empirical Ignorance score of $c(\mathbf{y})$ over the archive. It needs to be taken into account though that the same archive was used to build the density $c(\mathbf{y})$ in the first place. In order to avoid overfitting effects, leave-one-out cross validation was used to estimate the score. More specifically, we formed the densities

$$c_{-i}(\mathbf{y}) := \frac{1}{N-1} \sum_{k \neq i} \frac{1}{\sigma_k^d} K\left(\frac{\mathbf{y} - \mathbf{y}_k}{\sigma_k}\right) \tag{3.14}$$

and estimated the score by

$$S_N := \frac{1}{N} \sum_{i=1}^{N} -\log(c_{-i}(\mathbf{Y}_i)) \tag{3.15}$$

which was then minimized as a function of s. It is easy to see that if $c(\mathbf{y})$ were used in (3.15) rather than $c_{-i}(\mathbf{y})$, the minimum would be $-\infty$, occurring for $s = 0$. An implementation of this particular variant of kernel density estimation, which is somewhat of a cross-bred of concepts proposed in [73], is available for MATLAB under the Lesser GNU Public License [39].

3.3.3 Global and local density estimator

The kernel density estimator described in section 3.3.1 in connection with the invariant measure can be used just as well to obtain estimates of the joint density $p(\mathbf{y}, \mathbf{x})$, where again $\mathbf{x}$ is a vector of suitably delay-embedded observations. Unlike cluster weighted models, the kernel density estimator does not assume any functional dependence between the observations and the state of the dynamical system. Therefore, it can be interpreted as a somehow "brute force" approach to state estimation. The corresponding conditional density $p(\mathbf{y}|\mathbf{x})$ can be used as a probabilistic forecasting system for estimating the state of the dynamical system. In contrast to this global model, which depends on the entire archive of observations and corresponding state vectors, kernel estimators can also be used for constructing more local estimates around any possible realization of the delay vector of observations.

These densities are "local" in the sense that they are only assumed to be valid in some small neighborhood of the delay vector of observations and the corresponding unknown true state. In other words, each density is tailored to one specific delay vector of observations. This also implies that this model ansatz is a "lazy learner", since it delays any computation until it is queried with an actual delay vector of observations.

Given a set of pairs of scalar observations and corresponding state vectors, we first organize the scalar observations in delay vectors $\mathbf{x}_n$, as defined in section 3.3.1, Eq. (3.10). Given a new delay vector of observations $\tilde{\mathbf{x}}$, we search for the k nearest neighbors $\mathbf{x}_{nn(i)}, i = 1, \ldots, k$ of $\tilde{\mathbf{x}}$ among the archived delay vectors of observations.

To these delay vectors correspond archived verifications $\mathbf{y}_{nn(i)}$, to which the kernel estimator technique (as described in the previous section) is applied, providing a local density estimate $p(\mathbf{y}|\tilde{\mathbf{x}})$ around $\tilde{\mathbf{x}}$. Again, given the particular choice of inputs, this is effectively an estimate of the density $p\left(\mathbf{Y}_n = \mathbf{y}_n|(Z_n, Z_{n-\delta}, \ldots, Z_{n-(D-1)\delta}) = \mathbf{x}_n\right)$.

3.3.4 Interacting Particle Filters

The problem of estimating the state of a nonlinear system from noisy observations in a causal manner, that is by not allowing future observations to enter the current state estimation, is well known to have an optimal solution, often referred to as the optimal nonlinear filter. The optimal filter can be described as follows (see [40] and also [44] for continuous time filtering). Given states $\mathbf{Y}_n \in \mathbb{R}^d$ of a dynamical system (or more generally a Markov process) with observations $Z_n \in \mathbb{R}, n \in \mathbb{N}$, as defined in section 3.3.1, the problem of filtering is to compute

$$\pi_n(A) := \text{Prob}(\mathbf{Y}_n \in A|Z_{1:n}),$$

for arbitrary sets A, that is, the probability of finding $\mathbf{Y}_n \in A$ given the history of previous (and current) observations $Z_1 \ldots Z_n$ (which we abbreviated as $Z_{1:n}$). The quantity π_n is commonly referred to as the *filtering process*. Here we assume that the filtering process has a density (with respect to Lebesgue measure), which will be denoted by $\pi_n(\mathbf{y})$.

There are various different ways to represent the filtering process. For an overview see [40]. The representation used here resembles a dynamical system or evolution equation for the filtering process. Let

$$g(\mathbf{y}, Z_n) := \frac{1}{\sigma}\gamma\left(\frac{Z_n - \mathbf{c}\,\mathbf{y}}{\sigma}\right),$$

and define the measure

$$\pi_n^+(A) := \pi_n(F^{-1}(A)) = P(\mathbf{Y}_{n+1} \in A|Z_{1:n}). \tag{3.16}$$

Then the following holds:

$$\pi_{n+1}(A) = \frac{\int_A g(\mathbf{y}, Z_{n+1})\mathrm{d}\pi_n^+(\mathbf{y})}{\int_{\mathbb{R}^d} g(\mathbf{y}, Z_{n+1})\mathrm{d}\pi_n^+(\mathbf{y})} \tag{3.17}$$

The equations (3.16) and (3.17) should be understood as follows: To compute $\pi_{n+1}(A)$, sample from the distribution π_n and then apply the mapping F to the samples, resulting in a sample from π_n^+. Then weight each sample with the function $g(\mathbf{y}, Z_{n+1})$ (which brings the new observation Z_{n+1} into play), and finally "count" the weighted mass of all samples that fall into A. This gives the numerator in equation (3.17). The denominator is for normalization.

In order for (3.16) and (3.17) to be useful in a practical application, it is of course necessary to represent $\pi_n(\mathbf{y})$ by a suitable finite dimensional parametrization. If the dynamics are linear and the errors are normal, the Kalman filter [45] provides such a parametrization. If F is nonlinear though, the filtering process $\pi_n(\mathbf{y})$ typically does not admit a finite dimensional parametrization [15, 47]. Therefore, approximations are essential for applications. The approximative filter has to be, of course, finite dimensional and as optimal as possible. A large variety of approximation methods have been conceived, one of them being the interacting particle filter (IPF) [35], which will be employed as a benchmark.

Numerous variants and versions of it have since been conceived. Here we propose a variant with a *relaxation window*, which yields particles that are on (or in close proximity to) the attractor. The details of this approach are explained in appendix A. It is important to note that the interacting particle filter requires knowledge of the dynamical equations of the underlying system.

3.4 Numerical Simulations

In this section, the performance of the forecasting schemes presented in section 3.3 is investigated and compared in the context of several state estimation and prediction problems. The performance is quantified using the Ignorance score (3.1). Numerically generated data sets from two dynamical systems are considered, namely the Hénon system and the Lorenz system. Furthermore, we investigate data from a chaotic bronze spring experiment and compute probabilistic predictions of its oscillations.

3.4.1 Hénon System

Our first example is the Hénon system

$$\begin{aligned} y_{n+1}^{(1)} &= 1 - 1.4(y_n^{(1)})^2 + y_n^{(2)} \\ y_{n+1}^{(2)} &= 0.3 y_n^{(1)} \end{aligned} \tag{3.18}$$

with observations given by

$$z_n = \mathbf{c}\mathbf{y}_n + \sigma r_n, \tag{3.19}$$

where $\mathbf{c} = (1, 1)$, and r_n are independent random numbers with standard normal distribution. The actual amount of measurement noise is determined by σ.

Figure 3.1 shows a two dimensional delay embedding of the noise corrupted observations $\mathbf{x}_n$ with 30dB signal to noise ratio (left panel) and 10dB signal to noise ratio (right panel). The signal to noise ratio (SNR), when measured in dB, is given as $10 \log_{10}(V_z/\sigma^2)$, where V_z is the variance of the entire signal $\{z_n\}$, and σ^2 is the variance of the noise. The empirical Ignorance for the discussed forecasting

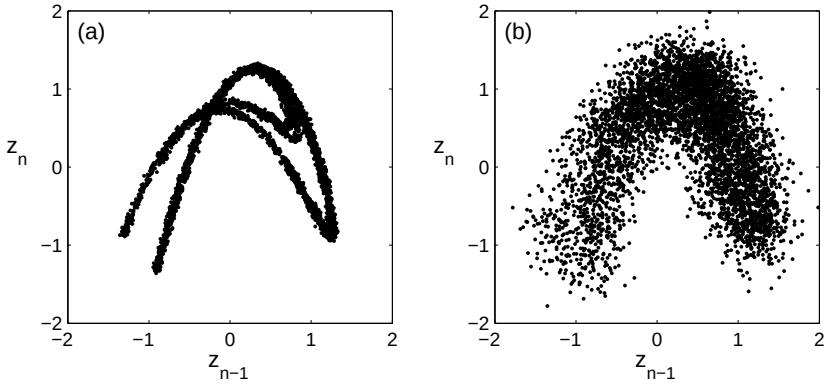


Figure 3.1: Two dimensional delay embedding of the noisy observations (Eq. 3.19) from the Hénon system (Eq. 3.18) with (a) SNR 30dB and (b) SNR 10dB.

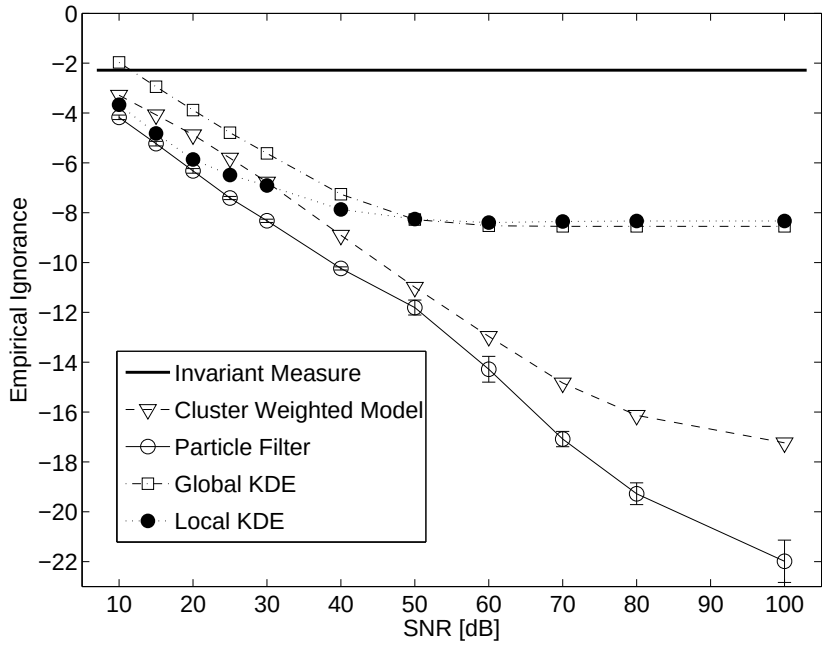


Figure 3.2: Empirical Ignorance (Eq. 3.5) vs. Signal-to-Noise Ratio compute with time series from the Hénon system (Eq. 3.18). The invariant measure, the local and global KDE, and the CWM are based on a three dimensional delay embedding with delay one. The CWM consisted of 100 clusters all equipped with (local) linear models.

schemes as a function of SNR is shown in figure 3.2. The models were trained on 5000 points of data from the Hénon system and evaluated on an equal number of points. As expected, for high noise levels (low SNR) the particle filter shows the best performance, with the local KDE coming second, CWMs are third and global KDE fourth. The forecasting performance of the invariant measure does not depend on either the observations or the noise level and is therefore represented by a horizontal line. At high noise levels (SNR $<$ 40dB) the performance as a function of SNR is surprisingly linear for all models except for the local KDE, which starts to saturate for SNR $>$ 20dB, converging to the same limit as the global KDE. The CWM forecasts show good performance for SNR $\leq$ 30 dB and outperform all other black-box model approaches for SNR $>$ 30 dB, with only the particle filter producing better results. While the difference between the CWMs and the particle filter is relatively small for SNR $\leq$ 60 dB, it gets increasingly larger as the noise further decreases. However, since the particle filter has knowledge of the underlying system equations, this result is to be expected. When computing statistical confidence bars for the performance, it has to be taken into account that the summands $-\log(p_n(\mathbf{Y}_n))$ in (3.5) are usually highly correlated. Therefore the standard $\frac{\sigma}{\sqrt{N}}$–estimator of the variance of the empirical average (3.5) is likely to be too low. To get a more realistic estimate of the performance variations, we divided the data into segments of 100 instances each, resulting in 50 segments. An average Ignorance S_b was calculated for each segments $b = 1 \ldots 50$ separately. The width of the confidence bars was then set to $\frac{\sigma}{\sqrt{50}}$, where σ here is the standard deviation of the S_b. In figure 3.2, confidence bars are shown only for the performance of the particle filter, since for the other algorithms, they turned out to be the size of the symbols and therefore have been omitted.

The discussion of the Hénon example is finished with a few technical remarks:

1. The empirical Ignorance can show extremely large variations when applied to imperfect probabilistic models. This is caused by "forecast busts", meaning that the forecast probability density is vanishingly small at the observation which in turn causes $-\log p_n(y_n)$ to be very large. A few such busts can completely dominate the empirical score. To avoid this effect, the forecast probability of the particle filter was mixed with the invariant measure $c(y)$, that is,
$$q_n(\mathbf{y}) := \alpha p_n(\mathbf{y}) + (1-\alpha)c(\mathbf{y}) \tag{3.20}$$
was used instead of p_n alone. The mixing parameter α was determined during the training phase. The resulting probability assigned to a verification $\mathbf{Y}$ is now never smaller than $(1-\alpha)c(\mathbf{Y})$. In the following, the forecast performance is stated in relation to the performance of the invariant measure as a reference, that is, the (mean of the) *difference* in performance between $p_n(\mathbf{y})$ and $c(\mathbf{y})$ is reported. This difference can be written as
$$S_N[p] - S_N[q] := \frac{1}{N}\sum -\log\left(\frac{q_n(\mathbf{Y}_n)}{c(\mathbf{Y}_n)}\right). \tag{3.21}$$

Replacing $q_n(y)$ from Equation 3.20 we get for every summand

$$\begin{aligned}\frac{q_n(\mathbf{Y}_n)}{c(\mathbf{Y}_n)} &= \frac{\alpha p_n(\mathbf{Y}_n) + (1-\alpha)c(\mathbf{Y}_n)}{c(\mathbf{Y}_n)} \\ &= \alpha\frac{p_n(\mathbf{Y}_n)}{c(\mathbf{Y}_n)} + (1-\alpha) \\ &\geq (1-\alpha),\end{aligned}$$

from which we can conclude

$$-\log\left(q_n(\mathbf{Y}_n)\right) \leq -\log\left(c(\mathbf{Y}_n)\right) - \log(1-\alpha).$$

Hence the empirical Ignorance of a forecast combined with the invariant measure relative to the invariant measure alone is never larger (i.e. worse) than $-\log(1-\alpha)$.

Blending with the invariant measure was found not to be necessary for other models.

2. The values of $-\log p_n(\mathbf{Y}_n)$ can be plotted versus $\mathbf{Y}_n$, which results in a plot of the Ignorance over the Hénon attractor. There seems to be no simple relationship between $\mathbf{Y}_n$ and $-\log p_n(\mathbf{Y}_n)$. In other words, one cannot speak of regions in state space where state estimation is either particularly easy or particularly difficult. The embedding dimension in our case was three, that is, p_n is a function of (z_n, z_{n-1}, z_{n-2}). It is therefore not surprising that $p_n(\mathbf{Y}_n)$ is subject to very strong fluctuations. We speculate that these fluctuations decrease with larger embedding dimension. However, what was discussed already in section 3.3.1 applies here, namely that the probability of the underlying state $p_n(\mathbf{Y}_n)$ given the complete history $Z_{1:n}$ cannot be reduced to a probability given a fixed window of observation. This is a mathematical consequence of the setup of the numerical experiment, and a marked difference to the deterministic case (i.e. in the absence of measurement noise or $\sigma = 0$), where $\mathbf{y}_n$ is a (two leaved) function of (z_n, z_{n-1}).

3.4.2 Lorenz System

Our second example is the well known Lorenz model

$$\begin{aligned}\dot{y}_1 &= 10(y_2 - y_1), \\ \dot{y}_2 &= 28y_1 - y_2 - y_1y_3, \\ \dot{y}_3 &= y_1y_2 - \frac{8}{3}y_3,\end{aligned} \tag{3.22}$$

with

$$z_n = y_1(n \cdot \tau) + \sigma r_n \tag{3.23}$$

and sampling time $\tau = 0.05$. The state to be reconstructed in this case is $\mathbf{y}_n := (y_1(n\tau), y_2(n\tau), y_3(n\tau))$.

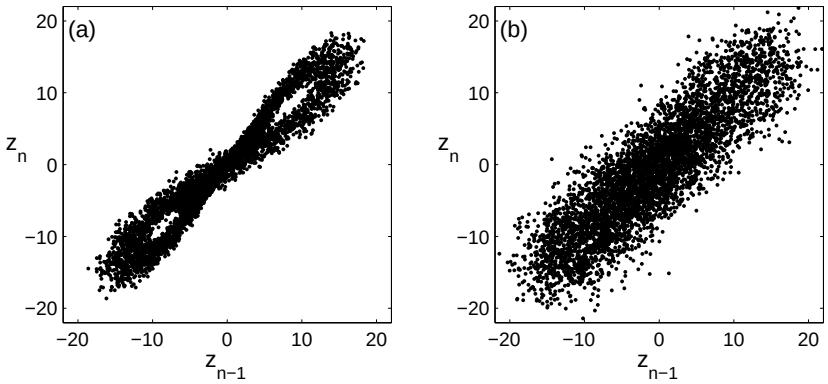


Figure 3.3: Two dimensional delay embedding of the noisy observations (Eq. 3.23) from the Lorenz system (Eq. 3.22) with (a) SNR 30dB and (b) SNR 10dB.

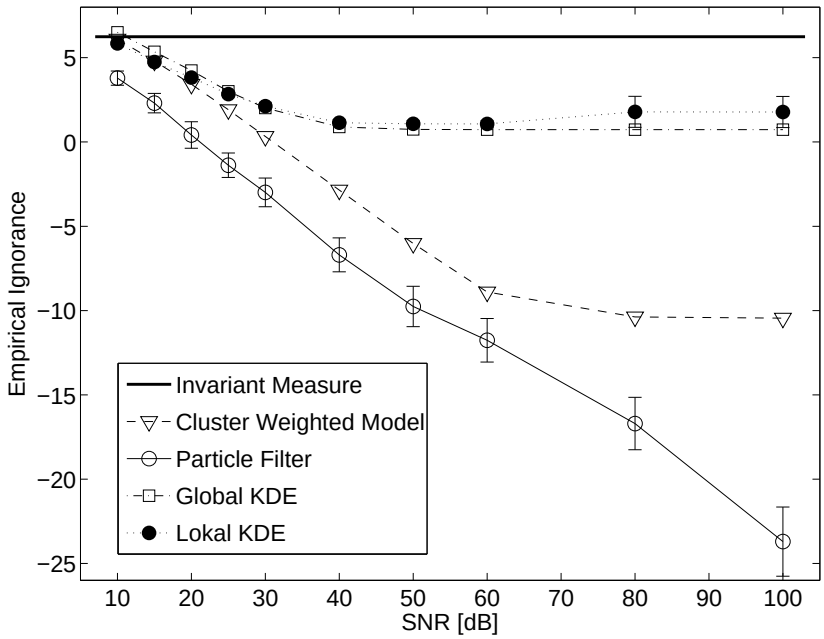


Figure 3.4: Empirical Ignorance (Eq. 3.5) vs. Signal-to-Noise Ratio compute with time series from the Lorenz system (Eq. 3.22). The embedding dimensions for the local and global KDE, and the CWM equaled 4, 3, and 7, respectively, all with delay $\delta = 1$. The CWM's comprised 60 clusters, all equipped with (local) quadratic models.

A two dimensional delay embedding of the observations can be seen in figure 3.3. The performance results shown in figure 3.4 are qualitatively similar to those for the Hénon system. As expected, the particle filter shows the best performance for

all noise amplitudes, with the difference between the CWMs and the particle filter getting significantly larger for very high SNR values above 60 dB. The Ignorance of the local and the global KDE saturate for SNR $>$ 40 dB, and the unavoidable modeling errors of the CWM result in saturation for SNR $>$ 60 dB. Confidence bars were calculated in a similar manner as discussed for the Hénon system. Again, the bars are only shown for the performance of the particle filter and the local KDE, since for the other algorithms, they turned out to be the size of the symbols and therefore have been omitted. We have not investigated the influence of the sampling time τ on the results. For the local and global KDE as well as the CWM, our cross validation procedure suggested a delay of $\delta = 1$. As was discussed at the end of section 3.3.1, this indicates that a shorter sampling interval might improve the results.

3.4.3 Bronze Ribbon Experiment

Our final example is concerned with data from an experimental chaotic oscillator. A detailed description of the experiment can be found in [22, 37, 71]. The experiment consists of a cantilevered horizontal bronze ribbon with a small magnet attached to the freely oscillating tip (see figure 3.5). The tip is subjected to an inhomogeneous magnetic field. Two coils, placed adjacent to the tip, are supplied with an AC voltage, thus driving the system. The voltage produced by a wire strain gauge attached to the beam is used as a measurement signal. The driving voltage is taken as $U(t) = U_0 \sin(2\pi t/T) + p$, and for suitable parameters U_0, T, and p, the bronze spring exhibits chaotic motion with an attractor of correlation dimension $\cong D_2 = 2.75$ [71].

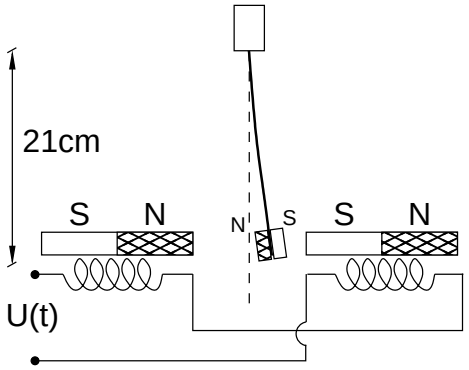


Figure 3.5: The bronze spring experiment (top view). A cantilevered horizontal bronze ribbon with a small magnet attached to the freely oscillating tip is subjected to an inhomogeneous magnetic field. The two coils are supplied with an AC voltage, driving the system. The deflection was measured by means of a wire strain gauge, attached to the beam close to the chucked end.

The variable to be forecast in this experiment was set to be the value of the time series 10 steps in the future. For the purpose of a detailed comparison of the

discussed algorithms, the data was corrupted with artificial noise. Independent and normally distributed random variables were used for this purpose, with mean zero and relative intensity of 10, 15, 20, 25, 30, 40, 50, 60, 80, and 100dB, as for the other two experiments. Note that in this case, the targets and features originate from the same source (albeit with time lags in between), while in the previous experiments, targets and features represented different variables. As a consequence, not only the properties of the features, but also of the targets change with changing noise strength, and in particular the performance of the invariant measure is different for different SNR, albeit only slightly, as it turns out.

The features were formed through delay-embedding, as described in section 3.3.1. The optimal embedding dimension, determined through cross validation, turned out to be four, at a time lag of five steps. In the present example, the performance of the global and local KDE and CWMs were compared, along with the unconditional density, in terms of Ignorance. The particle filter was not considered, as its application would have required a model of the system.

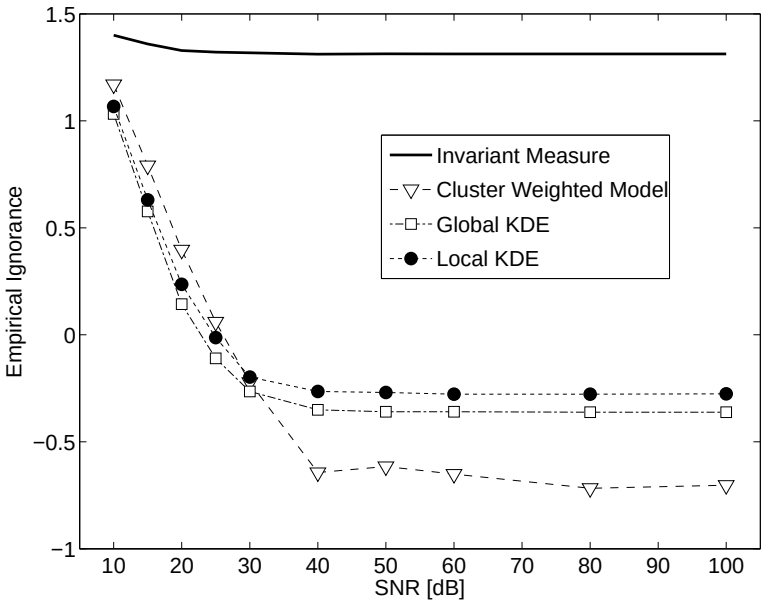


Figure 3.6: Empirical Ignorance (Eq. 3.5) vs. Signal-to-Noise Ratio for the bronze spring experiment. For all methods, the embedding dimension was 4 at a delay of 5 time steps. The CWMs comprised 120 clusters, all equipped with (local) quadratic models.

The performance of the four tested approaches is displayed in figure 3.6. We see a picture qualitatively similar to the previous examples. The invariant density performs worst, as expected. Both KDE approaches as well as the CWM perform similar until about 30dB, beyond which the CWM becomes significantly better than both KDE approaches. We again speculate that this is due to the CWM's more sophisticated modeling of the deterministic part of the relationship between features

and targets. The variability in the performance was very small, hence confidence bars could be omitted.

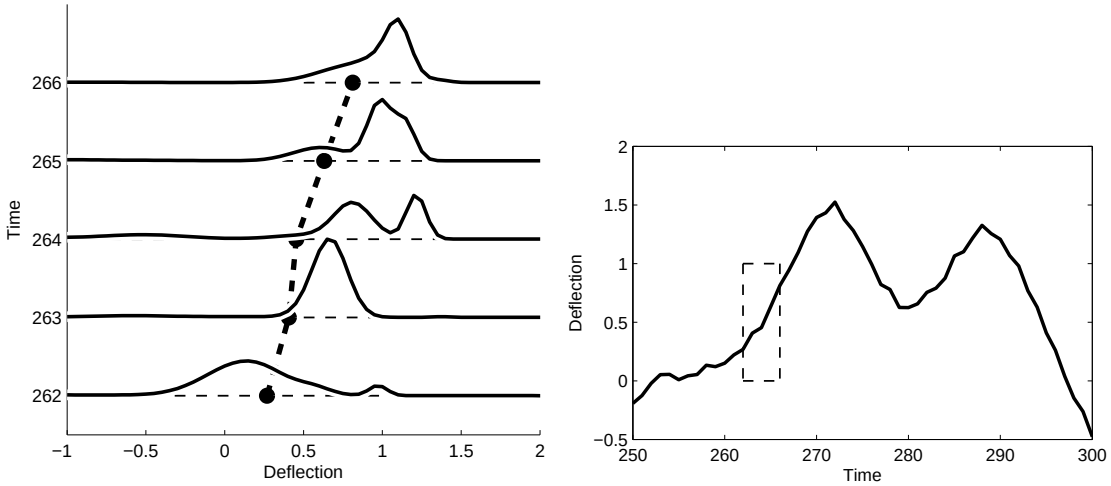


Figure 3.7: The left plot shows forecast densities for a few time instants for the bronze spring. The density becomes bimodal for $t = 264$. The plot on the right shows a short piece of the time series from the bronze spring. The dashed box marks the time window for which the estimated densities are displayed in the left panel.

For illustrational purposes, the estimated densities have been plotted for a few time instances (figure 3.7, upper panel; SNR=30dB). The lower panel of figure 3.7 shows a short piece of the time series from the bronze spring. The dashed box marks the time window for which the estimated densities are displayed in the upper panel. This piece of the time series is interesting, since the forecast densities seem to indicate that the system passes close to a saddle. Making predictions in such situations is inherently hard since small errors in the current state estimate are rapidly amplified near the saddle point, due to the presence of unstable directions. The forecast densities shown in figure 3.7, upper panel, reflect this fact. The density becomes bimodal for $t = 264$, as it is not certain at this point how the system will emerge from the proximity of the saddle. A few instances later, further information has been gathered, and the forecast density is unimodal again.

3.5 Conclusion

On the basis of our numerical simulations, we conclude that Cluster Weighted Models are well suited for obtaining probabilistic forecasts. Although CWMs were outperformed by the local KDE approach for low Signal-to-Noise Ratios (though only barely in the case of the Lorenz system), CWMs clearly perform best among the black-box approaches for medium to high SNR. Roughly speaking, we can locate

two different regimes in our examples: while for low SNR the purely stochastic model approaches based on the KDE perform well, they are unable to model the distribution for higher SNR, hence the performance saturates on a constant value. This is due to the fact that there is only a limited amount of data points available, and with decreasing SNR the number of points becomes insufficient to model the increasingly sharp probability distribution. Especially the local KDE approach, which performed very well for low SNR's in our examples, needs very small neighborhoods to model the resulting distribution of the dynamical system for higher SNR. Beginning with an SNR of about 40dB, this neighborhood reduces to five points (which is the lower limit in our algorithm), and the resulting model even performs slightly worse than the global KDE, which was the overall weakest black-box approach in our tests.

CWMs do not suffer from this effect, since they employ more sophisticated models for the functional relationship between input and output data. Although this functional relationship is obscured by noise in the case of low SNR, CWMs show their strength when the noise amplitude decreases. This can be clearly seen in our first two examples, where the performance of the CWMs saturates not until very high SNR and the data becomes practically noiseless (SNR $\geq$ 80dB). The results from the bronze ribbon experiment show the same behavior, albeit less pronounced, with the CWM already saturating for lower SNR values. For both the Hénon and Lorenz system, only the particle filter with its knowledge of the underlying system equations is able to outperform the CWMs. The particle filter was not applied to the bronze spring experiment, due to a lack of a proper understanding of how shortcomings of a dynamic model affect the performance or feasibility of the particle filter. Subject to this problem being resolved, we speculate that the instruments discussed, in conjunction with the IPF (or other nonlinear filter variants) can provide a framework for the probabilistic assessment of dynamical models. This however requires further investigations.

As a result, CWMs perform very well for state estimation of a deterministic process, ranging from very noisy to practically noiseless data. Only if there are sufficient data points available and the SNR is known to be fairly low, purely stochastic approaches like the local KDE are able to perform better. Additionally, CWMs are quite robust in their dependence on the model parameters. Although in our implementation the number of clusters is not data driven but has to be specified manually before training, the precise number is not as crucial as one might think, as long as it is not chosen too low. While a high number of clusters will lead to a large number of parameters to be estimated, an overfitting of the model can usually be prevented in the simplest case by an early stopping of the EM algorithm through means like cross validation. More sophisticated procedures for regularization will be discussed in the next chapter.

The numerical investigations show that the probability densities produced by CWMs show good performance when evaluated as full probabilistic forecasts. In particular, it can be concluded that CWMs provide useful information beyond just the expectation value.

Chapter 4

Regularization of Cluster Weighted Models

The problem of CWM regularization lies in the different kinds of parameters which are estimated during the EM algorithm. We can roughly divide those parameters, which were discussed in section 2.6, into three different types:

- *Cluster size and shape:* Positions $\boldsymbol{\mu}$, Variances in input space $\boldsymbol{\sigma}$ (or even the full covariance matrix $\mathbf{C}$), and in output space Σ. They define the area of confidence for each cluster.
- *Cluster weights* w_m, with $\sum w_m = 1$, the fraction of data each cluster explains.
- *Local function parameters* $\boldsymbol{\beta}$, the coefficients of the local, linear parametrized functions.

Based on the experiences in section 3, we discuss some means for the regularization of Cluster Weighted Models. First, we shortly review the different type of parameters which can be considered for regularization. Then we will deal with *early stopping*, before we turn to the more sophisticated methods which try to regularize the clusters and weights.

4.1 Early stopping

As described in the conclusion of section 3, it was observed that it is important to stop the EM algorithm before it reaches a regime where it begins to overfit on the given data set, and we used cross-validation to determine the onset of overfitting. This approach is well known under the name *early stopping*, and is for instance used in the training of neural networks [63], which also often have to deal with the problem of estimating many parameters with only scarce data sets.

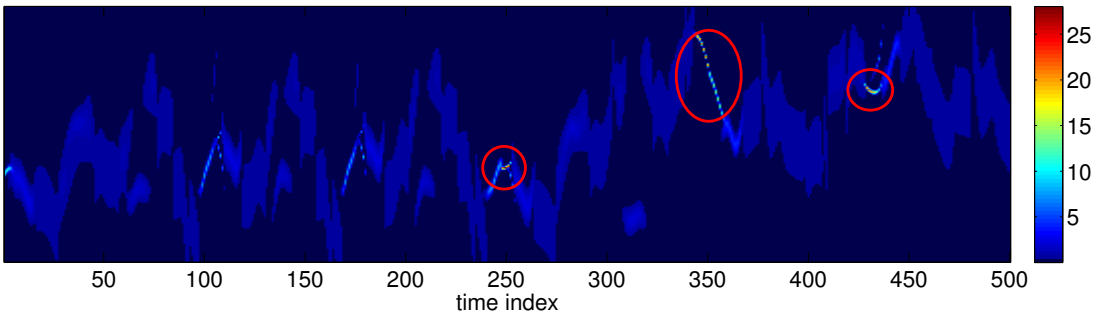


Figure 4.1: Example for speckling (marked regions) in the resulting distribution due to missing regularization (prediction of the Chua oscillator, sampled with 15Hz, predicted 50 steps into the future). The color for probability zero had to be further darkened, since with a linear scaled colormap most of the distribution would otherwise not be visible.

In the case of CWMs, early stopping is not employed to avoid a divergence of the parameters; as shown in section 2.6.2, the EM algorithm does converge, but given the large number of parameters to be estimated, especially in combination with the often very limited amount of data available in practice, the converged parameters are usually not general, meaning that the algorithm overfitted on the given data set. An additional, typical effect of overfitting are emerging "speckles" in the probability distribution, meaning small, separated regions with very high probability. This problem also occurs in other fields using the EM algorithm, for example in the reconstruction of Positron Emission Tomography (PET) images [50, 74], where one has to deal with very scarce data sets to keep the radiation exposure of the patient as low as possible. An example for such a speckled distribution can be seen in figure 4.1, which resulted from a direct prediction of 50 steps of the Chua oscillator (sampled with 15Hz). One can see the few regions with very high probability, making the rest of the distribution almost invisible in the image, due to the linear scaling of the colormap.

Using early stopping, one can halt the algorithm before such speckles emerge, thereby deliberately hindering the algorithm to converge, which in effect makes this a regularization method. However, it has the main drawback that it does not have a well-defined mathematical basis, which makes its usage slightly awkward. More importantly, all the parameters that were summarized in the previous section are practically handled the same way, but not only is their influence on the model very different, it is also very likely that they converge with different rates. While one could think of stopping the optimization of some parameters earlier than others, we would then need a different stopping criterion for every set of parameters, and it is not clear how those could be obtained.

We will therefore take a look at the different parameters and see how those could be regularized with respect to their influence on the model. This does not mean that

we can completely discard early stopping — in fact, stopping the EM algorithm is often still necessary, depending on the number of clusters — but it will become less important when additional regularization methods are employed.

4.2 Local function parameters

As described in section 2.6, the function parameters $\boldsymbol{\beta}$ are linear coefficients for the local functions (2.41), which are used in output terms (2.39). They define the functional approximation in the vicinity of a certain cluster c_m. In each EM iteration, their calculation requires a matrix inversion (see section 2.6.3, Eq. (2.65)), a problem very similar to the parameter estimation for nearest neighbor models with linear or higher order (section 2.5.1, Eq. (2.11)). We already discussed several possible regularization methods in section 2.5.5, and we can employ basically the same methods (ridge regression and principal component regression) here. Ridge regression has the advantage that we can use different regularization parameters on each component, but in case of CWMs, we already have a fine grained control of the component's influence through the input domain (2.36). Besides, one should be wary of raising the number of parameters even further; the basic principal component threshold regression (PCTR) only has one parameter (the threshold) and has shown to be a powerful method for nearest neighbor models, so it shall be used here, too. While in nearest neighbor models it was used for the calculation of a pseudo-inverse, it can be used just as well for invertible matrices, where the pseudo-inverse coincides with the regular inverse.

As described in section 2.5.5, PCTR works by introducing a lower bound $\sigma_{\min}$ on the singular values σ_i of the matrix which is to be inverted. All singular values which lie below this lower bound are simply dropped, thereby effectively discarding the dimensions with low variance, which are usually spurious dimensions, an artifact resulting from (measurement) noise in the data.

To show the effect of this regularization method, we shall investigate the modeling of the Chua oscillator (2.69); the aim is to directly predict the state reached after 50 steps. We use 2000 points, sampled with 15Hz, to train a cluster weighted model with 20 clusters and quadratic cluster functions. For calculating the model's performance, we use 3000 test points (in practice, one would usually not use more test than training points, but for demonstrating the effect of regularization, it is important to have a good estimation of the model error). For quantifying the model error, we use the Ignorance score described in section 3.2 (using the mean squared error would produce a qualitatively similar picture, but much less pronounced, since the ignorance score penalizes an overfitted model much more than the MSE).

Figure 4.2 shows the development of the Ignorance during EM optimization, with the abscissa showing the number of iterations. For a very strong regularization with a value of $\sigma_{\min} = 1$, effectively removing most of the model's variance, the

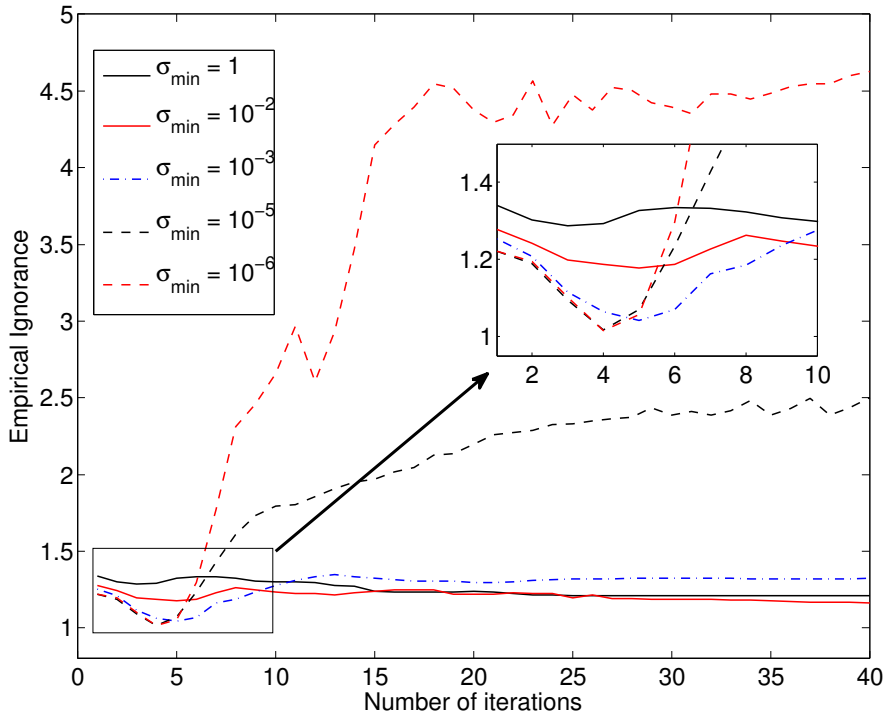


Figure 4.2: Development of the Ignorance during EM optimization for different PCTR regularization values. The larger the value for σ_{min}, the stronger the effect of the regularization. The zoomed region shows that a minimum score is already reached after 4–5 iterations.

Ignorance remains almost constant at a score of roughly 1.3. On the other end we have $\sigma_{\text{min}} = 10^{-6}$, which regularizes the model only slightly and results in a lower minimum score slightly above 1. However, this score is already reached after 4 iterations, and one can clearly see how the model overfits almost immediately afterwards, quickly reaching a score of 3 after only 10 iterations.

For $\sigma_{\text{min}} = 10^{-3}$, we get a minimum Ignorance almost as low (see zoomed section on the right-hand side), but without the same amount of overfitting, thus making it a good compromise between the two extremes. It still suffers from the fact that the lowest score is reached very early (after only 5 iterations). This is problematic since it is likely that other parameters, like the cluster positions and sizes, have slower convergence rates and might profit from a larger number of iterations. We will therefore take a look at regularizing the cluster's sizes in the following section.

4.3 Cluster variances (size)

Another set of parameters describes the size and shape of a cluster: the center $\boldsymbol{\mu}$ and its covariance matrix (or in the simplified form just the variances in input and output

space $\boldsymbol{\sigma}, \Sigma$). By looking at figure 4.1, one can assume that without regularization the output variances simply become too small, thereby concentrating all their weight on a very small region. In fact, when looking at the resulting output variances shown in figure 4.3, one can see that they stretch over several orders of magnitude. A regularization method for the variances should therefore penalize small variances and make sure that they are roughly in the same order of magnitude to avoid the emergence of speckles in the distribution. Regarding the cluster positions, we will not apply any specific regularization method for those, since they are not nearly as critical to overfitting as the variances.

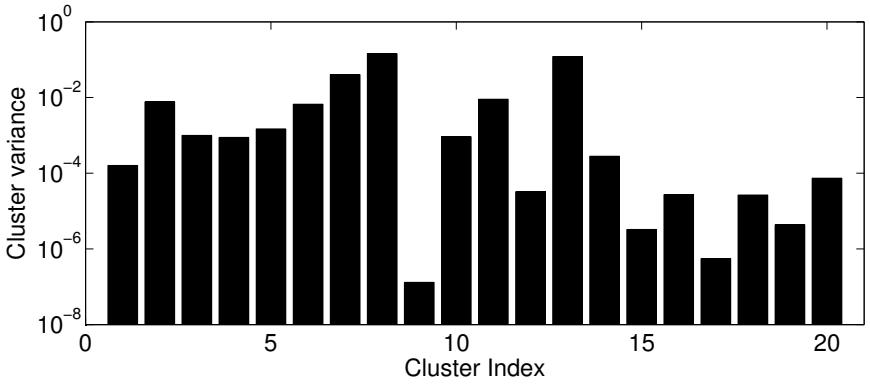


Figure 4.3: Output variances for the Chua example from figure 4.1. Without any regularization applied, the output variances spread over several orders of magnitude.

We will use a method which not only prohibits the clusters getting to small, but which also allows to enforce a certain similarity in the volume covered by the different clusters. It was developed for clustering algorithms using a mixture of Gaussians and for Fuzzy Clustering [11], and will be adapted in the following for CWMs.

To define the size of a cluster, we start with the determinant of its covariance matrix $\mathbf{C}_m$, which is the clusters squared (hyper-) volume. With d being the dimensionality of the data, we can define the *isotropic radius* and the *isotropic variance* through

$$\varsigma_m^2 = \sqrt[d]{|\mathbf{C}_m|} \quad \text{and} \quad \varsigma_m = \sqrt{\varsigma_m^2} = \sqrt[2d]{|\mathbf{C}_m|} \text{ , resp.} \tag{4.1}$$

For regularizing the cluster sizes, we can now introduce a rescaling of the clusters to ensure that they do not shrink too quickly, and to enforce at least a tendency to a relation between the different cluster sizes. In the extreme case, all clusters would have the same isotropic radius, variance or (hyper)-volume. For this, we introduce the exponent a of ς_m, with $a = 1$ and $a = 2$ leading to an equivalent isotropic radius or variance, resp., or with $a = d$ to an equivalent (hyper-) volume. After each EM iteration, we rescale the cluster's covariance matrix so that it has an isotropic

variance according to the following adaptation rule:

$$\varsigma_m^{\text{new}} = \sqrt[a]{s \cdot \frac{\sum_{k=1}^{M} \varsigma_k^a}{\sum_{k=1}^{M} (\varsigma_k^a + b)} \cdot (\varsigma_m^a + b)} = \sqrt[a]{s \cdot \frac{\sum_{k=1}^{M} \varsigma_k^a}{c\, b + \sum_{k=1}^{M} \varsigma_k^a} \cdot (\varsigma_m^a + b)}\,. \tag{4.2}$$

Next to the exponent a, we have two parameters to control the regularization: the parameter $b \in \mathbb{R}^+$ defines the amount by which the isotropic variable should be increased; for $b \to \infty$, the cluster sizes will be completely equalized, whereas for $b = 0$ no adaption will be done. It is also possible to set $b = 0$ as long as the spread between smallest and largest cluster size is below some critical value (see [11] for details).

Since the clusters are also normalized, the sum of the sizes will be preserved. However, to slow down the shrinkage of the clusters, we can use the scaling parameter s, which with $s > 1$ will scale the sum of the sizes up after each iteration. By choosing $s < 1$, we can also accelerate the size shrinkage, but this is usually not desirable. Since we have a variance in input as well as in output space (see eqs. (2.37) and (2.39)), we can rescale both variances by introducing the corresponding scaling parameters s_x and s_y.

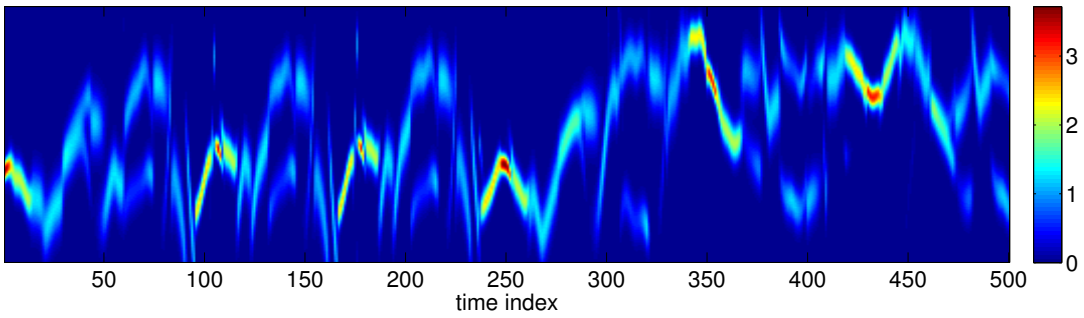


Figure 4.4: Distribution for Chua time series from 4.1 with size regularization. The parameters used were $a = 1, b = 5$ and $s_{x,y} = 1$ (i.e., no scaling).

In a similar manner, it is possible to regularize not only according to size, but also the shape of the clusters, i.e., avoiding a large spread of the variances in the different dimensions. However, we found that this spread is in fact an often desirable property, since it allows the clusters to adapt to the form of the data distribution. Only in the case of very few clusters, it might make sense to enforce a tendency towards a circular form.

Applying this method to the already mentioned Chua system, we can see in figure 4.4 that the distribution is now less speckled. While there are still areas with

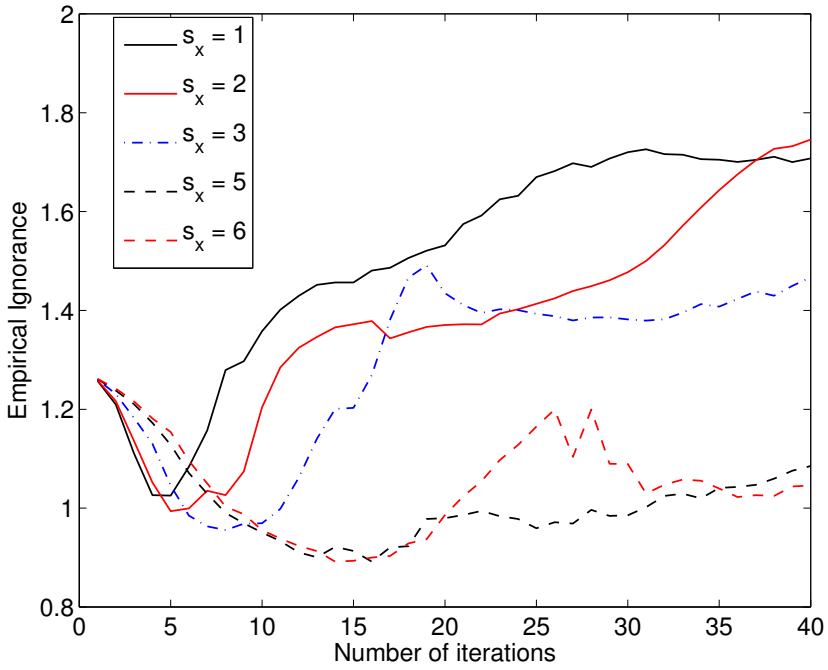


Figure 4.5: Development of the Ignorance over the EM iterations for different values of the scaling parameter s_x (with $s_y = 1$). One can see how larger scaling effectively slows down the EM optimization, also yielding a lower Ignorance.

high probability, the spread between the probability values is reduced, yielding a smoother distribution.

Figure 4.5 shows the effect of the scaling parameter s_x of the input clusters on the Ignorance score, with a cutoff value $\sigma_{\min} = 10^{-4}$ for the PCTR. One can see how larger values for s_x lead to lower scores at about 0.9, and with 15 iterations this minimum score is reached later in the optimization process. The rescaling of the clusters slows down the convergence, giving the clusters the necessary time to reach better parameters than without rescaling. However, a rescaling of the output clusters using $s_y > 1$ does not have a further positive effect on the ignorance score, which is why we will keep the value at $s_y = 1$ for the following examples.

Figure 4.6 shows the effect of the parameter b in equation (4.2), i.e., the size added to the isotropic radius or variance after each EM iteration. In this case, we used $a = 1$ (isotropic radius), $s_x = 4$ and $s_y = 1$ for the scaling parameters, and again $\sigma_{\min} = 10^{-4}$ for PCTR. One can see how enlarging this parameter slows down the optimization process even further. Here, the lowest score is reached after 26 iterations, and with a value of 0.81 it is lower than in the previous examples. One can see that even a very small value of $b = 0.15$ already has a pretty significant effect on the Ignorance score, with the lowest values being reached for $b = 1$ and $b = 3$. Larger values for b do not decrease the score further.

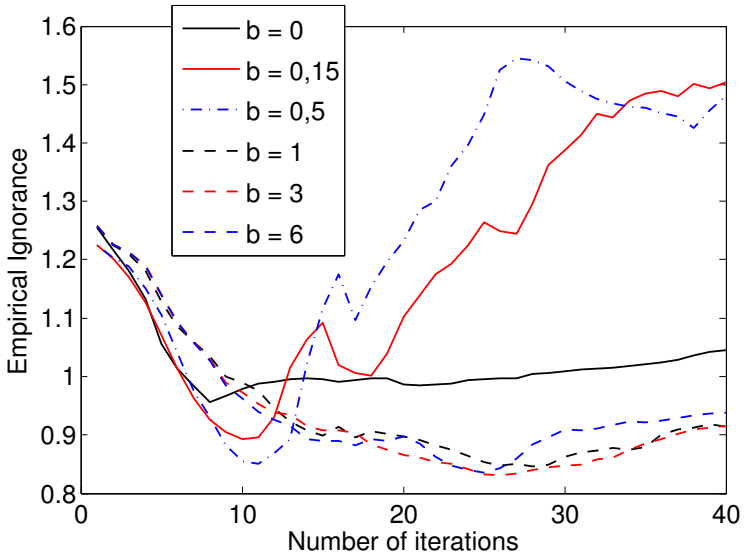


Figure 4.6: Development of the Ignorance over the EM iterations for different values of the size regularization parameter b, with $a = 1, s_x = 4, s_y = 1$ and $\sigma_{\text{min}} = 10^{-4}$.

Therefore, we decreased the overall score from 1 to 0.81 by combining the rescaling of the clusters (through parameter s_x) with the added size and renormalization (through parameter b). It is interesting to note that applying the added size b without scaling does not reduce the error in this manner.

4.4 Cluster weights

Another important set of parameters are the weights associated with the clusters. Similar to the variance distribution shown in figure 4.3, the spread between the weights can become quite large without regularization, though not over several orders of magnitude as seen for the variances (see figure 4.7). Still, the speckling of the distribution could stem from a combination of a small variance with high cluster weights, hence the modeling might benefit from weight regularization as well.

We basically use the same method we already used for the variances [11]:

$$w_m^{\text{new}} = \frac{\sum_{m=1}^{M} w_m}{\sum_{m=1}^{M}(w_m + b_w)} \cdot (w_m + b_w) = \frac{\sum_{m=1}^{M} w_m}{c\, b_w + \sum_{m=1}^{M} w_m} \cdot (w_m + b_w)\,. \tag{4.3}$$

Again, the regularization parameter b_w specifies the weight increase, and for $b_w \to \infty$ we get a uniform distribution. Since the weights w_m are probabilities, we must ensure that $\sum w_m = 1$ after regularization, so no additional scaling of the parameter can be done.

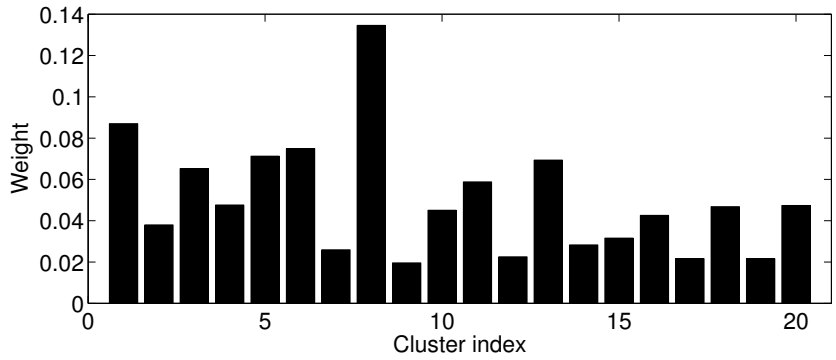


Figure 4.7: Cluster weights w_m for the Chua example from figure 4.1.

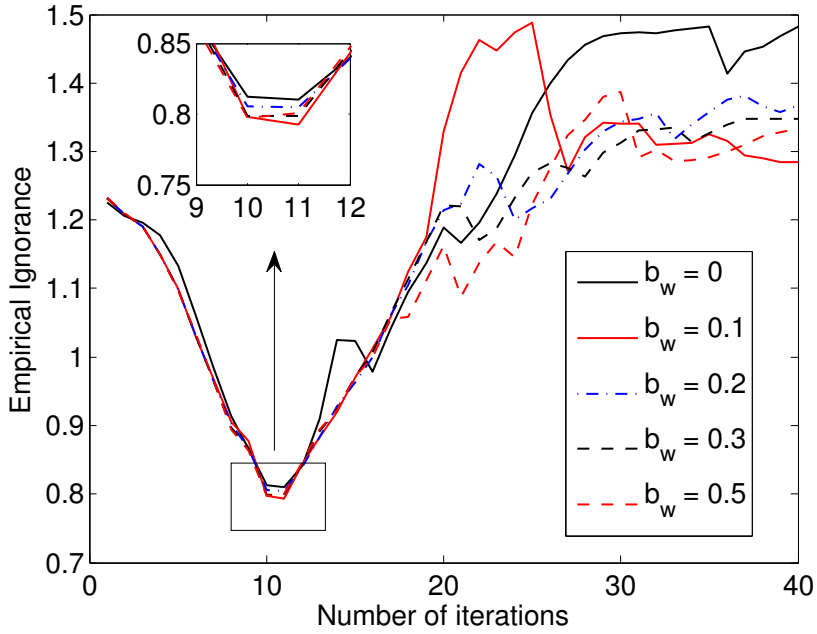


Figure 4.8: Development of the Ignorance score over the EM iterations with different weight regularization parameter b_w. The zoomed region contains the area with minimum score, showing the difference being very small.

It turns out though that weight regularization does not have the effect on the model's output one would hope. Figure 4.8 shows the development of the Ignorance score over the EM iterations, with b_w ranging from 0 to 0.5, with the other regularization parameters being $s_x = 3$, $b = 0.5$ and $\sigma_{\min} = 10^{-4}$. While the the lowest score is reached for $b_w = 0.1$, as can be seen in the zoomed region, the difference is minuscule. While the overfitting after roughly 20 iterations is a bit

reduced through the weight regularization, this alone hardly justifies an additional parameter. It seems that the PCTR and size regularization already have an influence on the weight distribution, so that an additional, explicit regularization does not yield a further reduction of the model error.

Chapter 5

Active Learning

So far we dealt with the construction of good models based on some data set which was simply given beforehand. In this section, we will discuss the problem of how such a data set can be obtained efficiently in the first place. This question is motivated by the fact that in practice, data is usually obtained through some kind of *experiment*, which is often a costly process, be it in terms of time or money. Therefore, in such a situation, it is of great importance to choose query points carefully as to not waste resources.

In general, the aim of creating an experimental design is to get a good model with as few data points as possible, which is known under the term *Response Surface Modeling* (RSM). However, there often exist additional objectives for which the constructed model should be used:

- *Screening*, which aims to find those input components with the greatest impact on the output variable.
- *Optimization*, i.e., finding the point in input space which maximizes or minimizes the target output.
- *Robustness and Stability*, meaning that little deviations in the input variables should not lead to drastic changes in the target output.

The problem of creating such an experimental design is of course not new, and there is a whole branch of statistics dedicated to this topic, known as *Optimal Experiments* [25], or more general as *Design of Experiments* (in the following abbreviated by *DoE*), which will be discussed in the first part of this chapter. DoE is well understood for *linear* models and Gaussian errors, and here it turns out that the data set can be constructed beforehand, meaning that it is completely independent of the actual output of the experiment. After a short review of the linear case, we will then deal with nonlinear models, leading to *Active Learning*, where in contrast to DoE, the current model output is integrated into the data set construction.

5.1 Design of experiments (passive learning)

In the following, an *experiment* is defined as procedure to obtain a *target* output variable y, dependent on some controllable *factors* $x^{(1)}, \ldots, x^{(d)}$. Given a fixed number N, which defines the number of times the experiment can be performed, we say that these are *optimal* when we gain a maximum amount of information from the resulting data set

$$\mathbf{\Omega} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}\,, \tag{5.1}$$

where $\mathbf{x} = (x^{(1)}, \ldots, x^{(d)}) \in \mathbb{R}^d$. This directly leads us to the question how we should quantify information, and how we can derive optimality criteria from this quantity.

The classical theory of DoE deals with linear parametrized models, meaning models of the form

$$f(\mathbf{x}) = \sum_{i=1}^{M} \beta_i \cdot g_i(\mathbf{x}) \tag{5.2}$$

with $g_i(x)$ being some suitable *basis functions* (e.g. monomes or radial basis functions) and $\beta_i \in \mathbb{R}$ being the linear coefficients of the model. With $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$ and $\mathbf{G}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_M(\mathbf{x}))$, we can write this as

$$f(\mathbf{x}) = \mathbf{G}(\mathbf{x}) \cdot \boldsymbol{\beta}\,. \tag{5.3}$$

Given the above data set (5.1) with $N > M$, meaning we have more data points than basis functions (and preferably much more), we can estimate these parameters through a least squares approximation by minimizing

$$\sum_{i=1}^{N} |y_i - f(\mathbf{x}_i)|^2 = \|\mathbf{Y} - \mathbf{G}(\mathbf{X}) \cdot \boldsymbol{\beta}\|_2^2\,, \tag{5.4}$$

with $\mathbf{Y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ and

$$\mathbf{G}(\mathbf{X}) = \begin{pmatrix} g_1(\mathbf{x}_1) & \ldots & g_M(\mathbf{x}_1) \\ g_1(\mathbf{x}_2) & \ldots & g_M(\mathbf{x}_2) \\ \vdots & \ddots & \vdots \\ g_1(\mathbf{x}_N) & \ldots & g_M(\mathbf{x}_N) \end{pmatrix} \tag{5.5}$$

being the so called *design matrix*. The parameter vector minimizing (5.4) is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{G}(\mathbf{X})^{\mathrm{T}}\mathbf{G}(\mathbf{X}))^{-1}\,\mathbf{G}(\mathbf{X})^{\mathrm{T}} \cdot \mathbf{Y} = \mathbf{G}(\mathbf{X})^{\dagger} \cdot \mathbf{Y} \tag{5.6}$$

where $\mathbf{G}(\mathbf{X})^{\dagger}$ denotes the *pseudo inverse* of $\mathbf{G}(\mathbf{X})$, as already described in section 2.5.1. This parameter vector $\hat{\boldsymbol{\beta}}$ is called the *least square estimate* and is a maximum-likelihood estimator, meaning that the probability density that the observations $\mathbf{\Omega}$ were generated by a parameter vector $\boldsymbol{\beta}$ has a maximum for $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}$.

Additionally, it is an *unbiased* estimator such that $\mathrm{E}[\,\hat{\boldsymbol{\beta}}\,] = \boldsymbol{\beta}_f$ for any true parameter vector $\boldsymbol{\beta}_f$.

If we now look at the difference between the estimated and the true (unknown) parameter vector $\boldsymbol{\beta}_f$, we get

$$\begin{aligned} \hat{\boldsymbol{\beta}} - \boldsymbol{\beta}_f &= \mathbf{G}(\mathbf{X})^\dagger \cdot \mathbf{Y} - \mathbf{G}(\mathbf{X})^\dagger \cdot \mathbf{G}(\mathbf{X})\boldsymbol{\beta}_f \\ &= \mathbf{G}(\mathbf{X})^\dagger \cdot \left(\mathbf{Y} - \mathbf{G}(\mathbf{X})\boldsymbol{\beta}_f\right) . \end{aligned} \tag{5.7}$$

We now have to make two important assumptions:

- The true, unknown mapping from $\mathbf{X}$ to Y can be described by the linear parametrized model, and
- the measurement error is Gaussian and i.i.d.

Given these two assumptions, $\mathbf{Y} - G(\mathbf{X})\boldsymbol{\beta}_f$ is distributed according to $N(0, \sigma^2)$ and a covariance matrix

$$\sigma^2 \cdot \mathbf{G}(\mathbf{X})^\dagger \cdot \left[\mathbf{G}(\mathbf{X})^\dagger\right]^{\mathrm{T}} = \sigma^2 \cdot \left(\mathbf{G}(\mathbf{X})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X})\right)^{-1} . \tag{5.8}$$

From this result, we can see that our aim should be to minimize the matrix given by $\left(\mathbf{G}(\mathbf{X})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X})\right)^{-1}$. However, there is no analytical procedure to uniquely minimize the entries of a matrix, hence we need some other kind of optimality criterion based on (5.8).

One possibility is to minimize the determinant, leading to a *D-optimal* experimental design $\mathbf{X}_{\text{d-opt}}$, satisfying

$$\begin{aligned} \det\left[\left(\mathbf{G}(\mathbf{X}_{\text{d-opt}})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X}_{\text{d-opt}})\right)^{-1}\right] &= \prod_{i=1}^{M} \lambda_i = \text{Min} \\ \Rightarrow \quad \det\left(\mathbf{G}(\mathbf{X}_{\text{d-opt}})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X}_{\text{d-opt}})\right) &= \text{Max} . \end{aligned} \tag{5.9}$$

Generally speaking, this criterion minimizes the confidence ellipsoid of the parameters $\boldsymbol{\beta}$. This however is just one possible criterion alongside several others for an optimal experimental design. For example, *A-optimal* designs minimize the trace

$$\text{trace}\left[\left(\mathbf{G}(\mathbf{X}_{\text{a-opt}})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X}_{\text{a-opt}})\right)^{-1}\right] = \text{Max} , \tag{5.10}$$

while *E-optimal* designs minimize the largest eigenvalue of $\left(\mathbf{G}(\mathbf{X})^{\mathrm{T}} \cdot \mathbf{G}(\mathbf{X})\right)^{-1}$. Among the other optimality criteria, two further ones shall be mentioned since they directly depend on the actual model error, which for one specific point in input space $\mathbf{x} \in M \subset \mathbb{R}^d$ is given by $e(\mathbf{x}) = \mathbf{G}(\mathbf{x})(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}_f)$. Given the above two assumptions that our model is sufficient for describing the process with a Gaussian measurement error, this model error is also a random variable $N(0, \sigma^2)$ with covariance matrix

$$\boldsymbol{\Sigma} = \sigma^2 \cdot \mathbf{G}(\mathbf{x})^{\mathrm{T}} \left(\mathbf{G}(\mathbf{X})^{\mathrm{T}}\mathbf{G}(\mathbf{X})\right)^{-1} \mathbf{G}(\mathbf{x}) . \tag{5.11}$$

From this, we can derive two other optimality criteria, namely *G-optimal* designs, which minimize

$$\max \left\{ \mathbf{G}(\mathbf{x})^{\mathrm{T}} \left(\mathbf{G}(\mathbf{X})^{\mathrm{T}} \mathbf{G}(\mathbf{X}) \right)^{-1} \mathbf{G}(\mathbf{x}) \,|\, \mathbf{x} \in M \subset \mathbb{R}^s \right\} , \tag{5.12}$$

and *I-optimal* designs, which minimize

$$\int_M \mathbf{G}(\mathbf{x})^{\mathrm{T}} \left(\mathbf{G}(\mathbf{X})^{\mathrm{T}} \mathbf{G}(\mathbf{X}) \right)^{-1} \mathbf{G}(\mathbf{x}) \, p(d\mathbf{x}) \tag{5.13}$$

with respect to a density p on M. Both designs are intuitively appealing since they look at the actual model error and not just at the parameter vector $\boldsymbol{\beta}$, but since we have to regard every possible point in input space, those designs are much more complex to calculate. Additionally, it turns out that at least G-optimal designs are strongly related to D-optimal ones, even being equivalent under certain conditions [78]. However, there is no exact analytical solution for calculating D-optimal designs; instead, one usually uses a computational, iterative search algorithm from one of the various available statistics software.

5.2 Active Learning strategies

In the classical Design of Experiments paradigm, the experimental design does not change during the experiment, but is fixed beforehand. Therefore, if we would like to apply a learning algorithm to model the mapping between the factors and the target variable, this algorithm is completely *passive* in its selection of the training examples. The learning algorithm is simply confronted with an existing training data set, including the experimental outputs. As described in the previous section, given a linear process with Gaussian measurement errors, and using a model which can exactly describe this process, this is no limitation. As soon as we deal with nonlinear processes however, it makes sense to use a strategy which chooses new query points during the modeling process, so one can react the actual experimental outcomes. This strategy is called *Active Learning*, or sometimes also known as *Query Learning* [2].

In an Active Learning scenario, the learning or modeling algorithm can decide for itself for which input data it would like to see the experimental output; the design is not fixed before the modeling begins, but is constructed while the experiment is performed.

Of course, one has to start with some kind of given data set to be able to construct a first preliminary model. This initial data set is usually very small and contains either random points from the input space or points chosen according to some space filling criterion (a grid over the input space, for instance). In addition to the modeling procedure, one now also needs some kind of *query algorithm*, which is able to choose one or several further points which it deems informative and for which it would like

to see the experimental output[1]. The new point, together with its corresponding output, is incorporated into the training data set and the process is repeated until some convergence criterion is met, be it simply a maximum number of iterations or some measure based on the model's training error or variance.

One can think of various criteria for the query algorithm to decide which point(s) should be chosen. Especially in the beginning, when there are not enough data points to construct an acceptable model, one can resort to the two most basic strategies:

- *Random* picking of points from the data set, and
- *space filling*, which means picking the point with the largest distance to all other points in the data set (this of course implies that the input space is bounded).

For some models types and regression problems, it is possible to calculate the expected model variance when incorporating a new data point, which is a statistically optimal solution to the Active Learning problem [19]. However, for many model types, calculating a closed form solution for the expected variance is often intractable, so there are many other strategies which work by optimizing a different, non-optimal query criterion [65].

While there are many different strategies for Active Learning, they usually stem from two main approaches:

- *Model variance*, used as a measure of model *uncertainty*. The idea is to include points where the model has a high uncertainty, since we can expect that these will have a higher gain of information than those points where the uncertainty (and hence the variance) is low. This strategy is also known under the name *uncertainty sampling* [48]. There are several variations to this strategy; as already mentioned, it is possible for some model types to calculate the *expected* model variance under inclusion of a new data point [19], but those statistics usually have similar leading terms like the directly calculated model variance, and therefore lead to similar results.
- *Query by committee* (QbC), which uses an *ensemble* of several models $g_i(\mathbf{x}), i = 1, \ldots, K$, from which one can calculate the ensemble's variance

$$\mathrm{var}_{\mathrm{ens}} = \frac{1}{K-1} \sum_{i=1}^{K} (g_i(\mathbf{x}) - \bar{g}(\mathbf{x}))^2 , \tag{5.14}$$

  with $\bar{g}(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} g_i(\mathbf{x})$ being the ensemble mean.

  The notion behind QbC is that a large ensemble variance should correlate with a large uncertainty [72]. Hence, it is reasonable to include the query

[1] In the Machine Learning community, this is usually paraphrased as "presenting the query point to an *oracle*", which is able to *label* any point from the input space without error.

point for which $\mathrm{var}_{\mathrm{ens}}$ is maximal. QbC is especially useful where calculating the variance for a single model is intractable or even impossible. Of course, a requirement for QbC is that the models $g_i(\mathbf{x})$ are not identical, either by choosing completely different model types, or by using different training data sets. If the modeling algorithm has a dependence on the initial conditions of the model parameters (as is the case for Cluster Weighted Models), one can also create a model ensemble by varying those. Although creating a whole ensemble of models increases the computational cost, they are often shown to be superior than one single model, since in the ideal case they have uncorrelated errors and can, in a manner of speaking, "correct" each other. The reader is referred to [36, 46, 80] for further discussions on model ensembles.

5.3 Using Cluster Weighted Models for Active Learning

As described in section 3.2, a model which produces a probability distribution as output can provide the user with much more information than just the mean. The first thing coming to mind is of course the variance in output space, which in the case of Cluster Weighted Models can be analytically derived by calculating the conditional expectation of the output variance [70] and is given through

$$< \Sigma \,|\, \mathbf{x} >= \frac{\sum_{m=1}^{M} \left[\Sigma_m + f(\mathbf{x}, \boldsymbol{\beta}_m)^2\right] \, p_m(\mathbf{x}) \, w_m}{\sum_{m=1}^{M} p_m(\mathbf{x}) \, w_m} - < y \,|\, \mathbf{x} >^2 \ . \tag{5.15}$$

Therefore, it is not strictly necessary to use a model ensemble to obtain a measure of uncertainty for the model output, although we will see that especially in the beginning, when the data set is still small, using an ensemble is still advisable for a better estimation of the variance. The first description of the algorithm will be done with a single model, to keep the notation reasonably simple.

In the following, our main application for Active Learning will be the optimization of some unknown cost function. That means, our learning procedure should primarily explore areas where the function might have an extremal value of interest (i.e., a minimum or maximum). Based on the variance (5.15) for CWMs, we can formulate the following Active Learning algorithm:

Given: Initial data set $\boldsymbol{\Omega}^{(0)} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_K, y_K)\}$, consisting of K pairs of input vectors $\mathbf{x} \in \mathbb{R}^d$ and corresponding scalar outputs $y \in \mathbb{R}$. The number of data pairs K is chosen so that it is possible to get a model that is numerically stable. In the following, the variable j denotes the current iteration, beginning with $j = 0$.

1. Train model CWM($\boldsymbol{\Psi}$) with the current training set $\boldsymbol{\Omega}^{(j)}$. The vector $\boldsymbol{\Psi}$ denotes the model parameters which have to be trained; in the case of CWMs, this parameter vector consists of

$$\boldsymbol{\Psi} = [\{\boldsymbol{\mu}_i\}, \{\boldsymbol{\sigma}_i\}, \{w_i\}, \{\boldsymbol{\beta}_i\}] \ \text{ with } i = 1, \ldots, M \,,$$

where we used the same notation as in chapter 2.6. The training of the models yields a parameter vector $\mathbf{\Psi}^{(j)}$.

2. Using the model CWM($\mathbf{\Psi}^{(j)}$) obtained in the previous step, calculate the conditional expectation $\hat{y}$ and variance $\hat{\Sigma}$ over the given input space. For an input space with low dimensionality (≤ 3), this can usually be achieved by evaluating the model on a grid over the input space (the problem of higher dimensional spaces will be discussed in chapter 7).

3. Choose new points for training according to the following criteria:

   - *Space filling*: As already mentioned in section 5.2, this chooses point $\mathbf{x}_{\text{sf}}$ with the longest distance to all other points in $\mathbf{\Omega}^{(j)}$:

     $$\mathbf{x}_{\text{sf}} = \arg\max_{\tilde{\mathbf{x}}} \|\tilde{\mathbf{x}} - \mathbf{x}_i\|_2, \quad \forall \mathbf{x}_i \in \mathbf{\Omega}^{(j)} .$$

     Again, for low dimensional spaces we can get an approximation of $\mathbf{x}_{\text{sf}}$ through evaluation over a grid. For high dimensional spaces, it is usually best to simply resort to a *random* point in input space.

   - *High variance*: Choose point with maximum model variance. This mostly targets areas in input space where there are already training points, but not enough to produce a robust model output.

   - *Minimum or maximum values*: Especially when one is interested in extrema of the modeled function or system, one aims to identify regions of maximum or minimum values. Therefore, it makes sense to investigate those regions in the model output more carefully. Next to the obvious current global minimum or maximum of the model output, it is also advisable to include a current local minimum/maximum, since it might turn out to actually be a global one. Local extrema can be obtained by using a steepest descent/ascent algorithm with a random starting point.

4. Add points according to the previous criteria, forming the new training set $\mathbf{\Omega}^{(j+1)}$, and return to step (1). End the algorithm when a stopping criterion is reached, like the model variance being below some value, or by setting a limit on the number of iterations or the number of points in the training set.

This algorithm defines the basic Active Learning procedure, and can be modified in various ways, also dependent on the iteration j, since with growing j the model error decreases and criteria like *space filling* become less important.

Using a model ensemble for Active Learning

Especially in the beginning of the algorithm, when there are only very few points available for training the model, one is faced with the problem that the resulting

model is highly dependent on the initial distribution of the clusters. As already done in section 2.9, where an ensemble of CWMs was used for coping with instabilities when doing freely iterated predictions, we shall also use an ensemble here. The idea is that we can use the mean of this ensemble to get a more reliable model, since in the ideal case the different ensemble member will make uncorrelated errors which will cancel each other out.

Of course, a precondition when building model ensembles is that the individual models are different. This has to be guaranteed either by feeding different data into the models, or by having a stochastic component in the modeling algorithm [46]. As already noted above, the stochastic element in CWMs is the initial distribution of the clusters, thus we can use the same input data for all models and still obtain different individual model outputs.

5.4 Numerical Example

For showing how the algorithm works, we shall use numerically generated data from a function with one global minimum at the origin, which is surrounded by local minima, shown in figure 5.1. Since the global minimum is only slightly below the surrounding local ones, detecting the global minimum is a difficult problem.

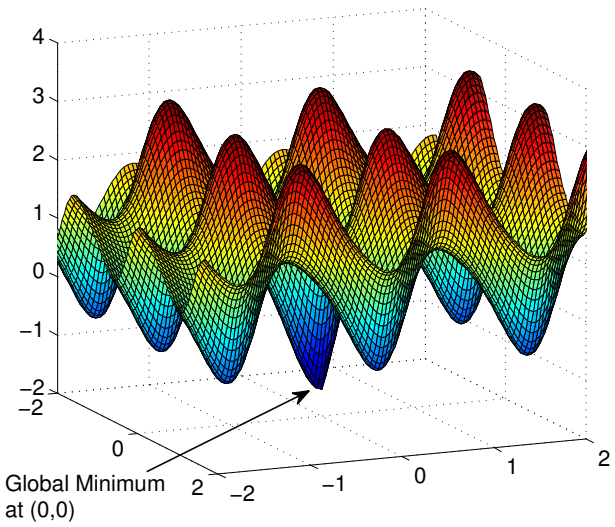


Figure 5.1: Function with one global minimum at the origin, surrounded by local minima.

We start out with a training set forming a 7×7 grid over the bounded input space $[-1\ 4]^2$ (we deliberately do not use a symmetric input space around the origin, where the global minimum resides). With these points, we train five initial CWMs, each with another initial distribution for the cluster positions (but same variances, to

make sure that the clusters roughly overlap the training points). Since we only have a scarce point set, we cannot use any points for testing the model output; instead, we limit the variance of the CWMs by using only 10 clusters and 10 iterations. However, these parameters will be raised during the learning process, since with a growing data set we can make the models more complex.

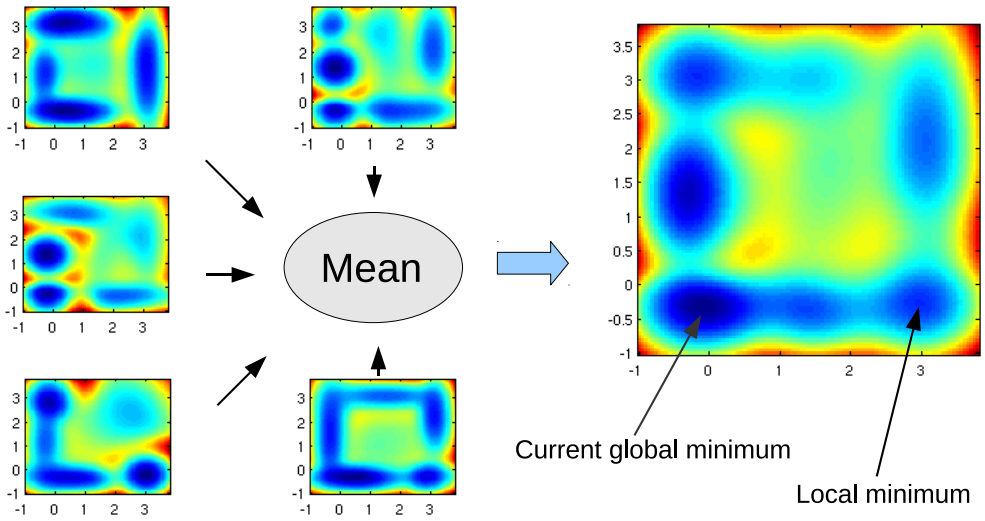


Figure 5.2: Model output of the five individual CWMs, forming a model ensemble. On the right panel, the mean of those five models is shown, which is then used for the Active Learning algorithm.

In figure 5.2, we can see an example after the first iteration of the Active Learning algorithm using an ensemble of CWMs. On the left-hand side, one can see the results from the five individual models. Due to the small training data set and the different initial cluster positions, those five models vary greatly in their output. Averaging those five models yields the output shown on the right-hand side. The global minimum is at (0,0) in the lower left, but currently the model fails to detect it correctly — while it shows a minimum in that region, its position is skewed to the lower left, due to the position of the initial grid points (seen in figure 5.4(a)).

In figure 5.3, the same is done for the variance: the left-hand side showing the five individual model variances, and the average on the right-hand side. One can see that the regions of highest variance mostly lie between the detected minimal regions.

As described in section 5.3, we choose new points for training according to variance and global/local minima; every third iteration, we search the current global minimum, otherwise we start with a random point in input space and use a steepest descent algorithm to find a minimum, which will in this case likely not be the global one. This way, we get two new points for each iteration. Space filling or random points were not used, since we used an initial grid for training, which already covered

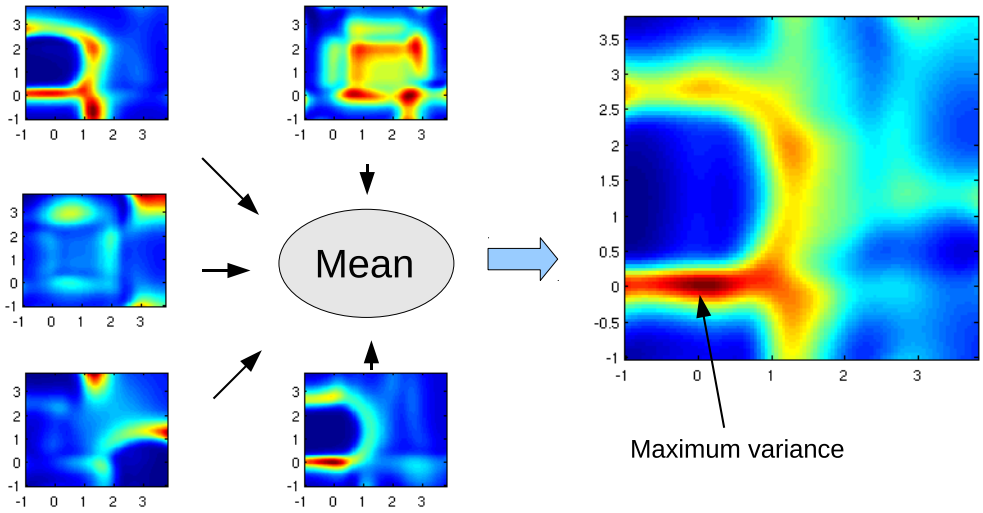
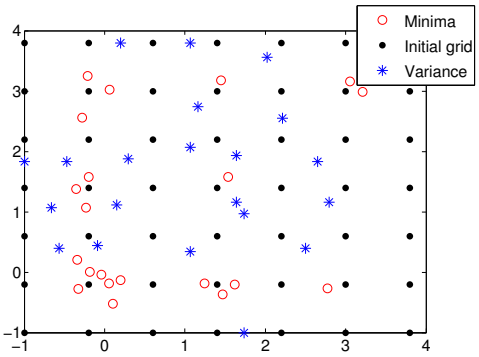
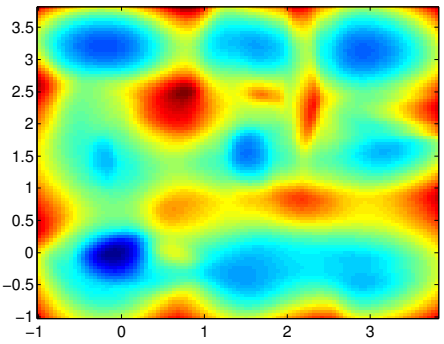


Figure 5.3: Model variance for each ensemble member, and again averaged to form the ensemble's output.



(a) Distribution of training points, showing the initial grid (black), points chosen with the variance (blue stars) and the minima criterion (red circles).



(b) Model output of CWM ensemble with the point distribution shown in left plot.

Figure 5.4: State of modeling after 21 iterations of the Active Learning algorithm.

the input space pretty well.

In figure 5.4 we can see the state of modeling after 21 iterations of the Active Learning algorithm. In figure 5.4(a) we see the point distribution, marked with different symbols denoting the criterion by which they were chosen. One can clearly see how points are clustering around the global minimum at the origin, which is also reflected in the model output shown in figure 5.4(b). It is important to note that the primary aim here is to quickly identify the basic features of the underlying

function, mainly where its minimum regions are and which one is in fact the global minimum. The actual model error might in fact be higher than when using simpler point selection strategies like random points or space filling, but they are less likely to cover the important features.

Chapter 6

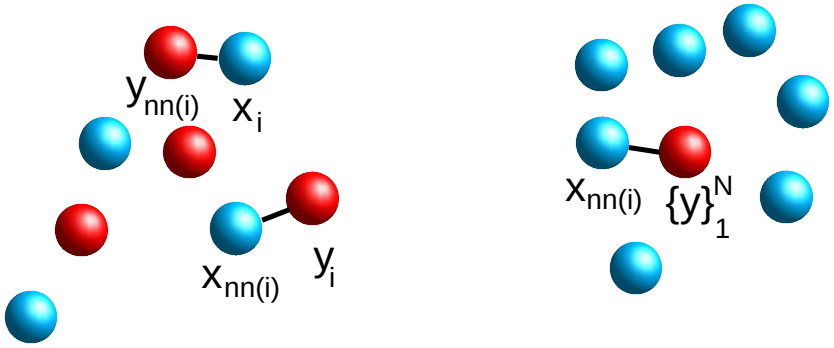
Training models on larger time scales

For chaotic systems, we have so far restricted ourselves to prediction on short time scales, limited by the prediction horizon of the observed system, after which a good prediction is no longer possible ("good prediction" meaning a NMSE < 1, i.e., better than a trivial estimation using the time series' mean). However, even when a model yields very good prediction results on short time scales, it can exhibit a vastly different behavior from the observed system on longer time scales; in fact, one often faces the problem that the model is not even stable when freely iterated, as was shown in section 2.9 for friction modeling using CWMs.

Due to their sensitivity on the initial conditions, it is an inherent property of chaotic systems that they cannot be predicted for arbitrary time lengths. Still, one often would like to have a model that *qualitatively* exhibits the same behavior as the observed system, meaning that their attractors should show the same essential features. This motivates the following procedure, which employs a cost function for comparing two attractors in phase space.

6.1 Comparing attractors in phase space

The basic idea is to compare two attractors in the same embedding space, thus we need a *measure of similarity* of two compact sets in phase space. This problem is of course not new — there are various approaches in the literature to tackle it, like the Kolmogorov-Smirnov test [64], which is used for testing if two data sets were drawn from the same probability distribution, but although there exist extensions of the test to two and three dimensions, it is usually very difficult to employ it for higher dimensional data. The often used Kullback-Leibler divergence [36] can be used for multi-dimensional data, but goes to infinity when the two distributions become disjoint. Furthermore, we need a procedure that should be robust against noise and should also be independent of the sampling time with which the data was recorded, as long as the attractor can still be unfolded with the lowest possible delay $d = 1$.



(a) Best case scenario: For each point in A (blue) there is a unique nearest neighbor in B (red).

(b) Attractor B (red) is a fixed point, so that each point in B has the same nearest neighbor in A (blue).

Figure 6.1: Illustrative example of the nearest neighbor method to obtain a measure of similarity for two point sets.

The main idea is to use a purely *local* procedure: Given two attractors A and B in the form of point sets $\{\mathbf{x}_i\}_{i=1}^N$ and $\{\mathbf{y}_i\}_{i=1}^N$, resp., we can for each point $\mathbf{x}_i \in A$ search its nearest neighbor $\mathbf{y}_{nn(i)} \in B$, and we can sum up the distances between those pairs:

$$d_{xy} = \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{y}_{nn(i)}\| \,. \tag{6.1}$$

Conversely, we can calculate the distance from B to A through

$$d_{yx} = \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{x}_{nn(i)}\| \,. \tag{6.2}$$

If the two point sets were completely identical, we would have $\mathbf{x}_i = \mathbf{y}_{nn(i)}$ for each $i \in \{1, \ldots, N\}$ and thus $d_{xy} = d_{yx} = 0$. With different attractors, we would have in the ideal case a unique nearest neighbor for each point, for which it follows that $d_{xy} = d_{yx} > 0$, hence one of the two distances would already suffice; this situation is shown in figure 6.1(a). Since this will usually not be the case, the two distances will differ and we have to calculate both distances, leading to the primary cost function

$$p = d_{xy} + d_{yx} \,. \tag{6.3}$$

6.1.1 Cost functions for comparing attractors

Figure 6.1(b) shows an illustrative example where the just introduced primary cost p is misleading. Here, the points from attractor A are spread, but attractor B is a

fixed point, meaning that the set $\{y_i\}_{i=1}^N$ consists of identical points $y_i = y_{\text{fp}}$ for all $i \in \{1, \dots N\}$. Since all points from B have the same nearest neighbor, the resulting distance d_{yx} in (6.3) will be small, although the two attractors are vastly different. One possible solution for this problem is to also incorporate the distribution of the nearest neighbors indices $nn(i)$.

For this, we look at the two extreme cases:

- In the ideal case, each point from one attractor has a unique nearest neighbor in the other point set, leading to a uniform distribution of the nearest neighbor indices $nn(i)$.
- In the worst case, all points in one attractor have the same nearest neighbor, leading to a delta-like distribution of the indices. Figure 6.1(b) is just showing one possible example for this case, but it can also happen for two attractors which are completely disjoint, for instance.

One well-known measure which can be used to quantify the complexity of a distribution p is the Shannon entropy

$$S(p) = -\sum_{i=1}^{N} p_i \cdot \log(p_i)\,. \tag{6.4}$$

The value of $S(p)$ is maximal for p being uniform and minimal for a delta distribution, hence with setting p as the distribution of the nearest neighbor indices, our new cost function reads

$$p = d_{xy} + d_{yx} + C \cdot \frac{S_{\max} - S(p)}{S_{\max}}, \quad \text{with } S_{\max} = \ln N\,. \tag{6.5}$$

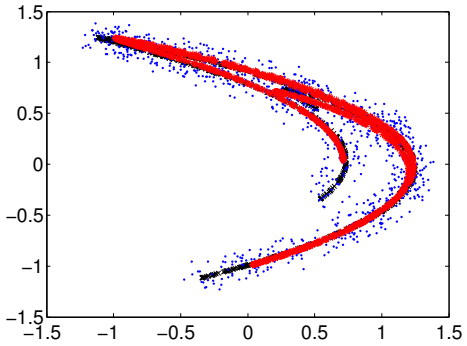
Through the constant $C \in \mathbb{R}^+$ we can control the influence of the distribution's entropy to the cost function.

Instead of using the entropy in the cost function, we can also *enforce* a uniform distribution of the nearest neighbor indices. We can do this by removing a point from the attractor set as soon as it was used in the distance calculation. The major drawback of this procedure is that we cannot use an efficient algorithm for searching nearest neighbors like ATRIA [55], since those rely on a search tree built in a preprocessing stage (as described in section 2.5). Removing a point from such a search tree is much more costly than simply doing a brute force search in the first place.

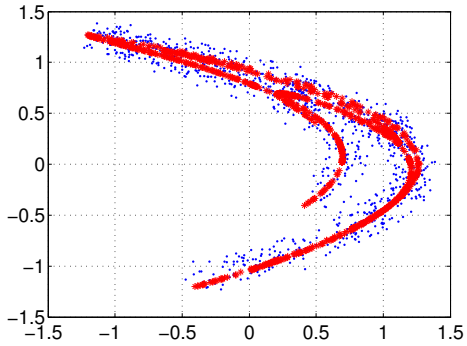
To diminish the computational cost of the procedure, one can implement a "blocked" version of the algorithm: With user-chosen block size B, first build a search tree with the current point set, then search the nearest neighbors for B points in the attractor, and remove those B points from the point set. The larger B, the more the distribution of the found nearest neighbors can deviate from the uniform distribution.

6.2 Example: Post-optimization of a polynomial model through attractor comparison

Given is a time series from the Hénon map (3.18), corrupted with white noise. In figure 6.2(a), the original, noise-free Hénon attractor is shown in black. The noisy time series, embedded in two dimensions, is shown with blue dots, yielding a "blurred" version of the true attractor.



(a) With least squares fit, the model's attractor (red) deviates from the original attractor (black) due to noise in the data (blue dots), especially in the outer regions.



(b) After optimization through attractor comparison, the model's attractor (red) now reaches the outer areas.

Figure 6.2: Post-optimization of a polynomial model for fitting a noisy Hénon time series.

Based on the reconstructed (noisy) attractor from the time series, we fit a polynomial model of degree 3, using a simple least-squares estimator for the model coefficients. The resulting attractor from this model is shown in red; one can see that while the model should technically be able to perfectly model the Hénon map (which is a polynomial of degree 2), the noise in the time series introduces a bias, leading to erroneous coefficients from the least squares estimator. The model's attractor differs significantly from the noiseless one; most prominently, it fails to reach the outer areas of the original attractor.

We now employ a genetic algorithm (see section 2.5.6), targeted at minimizing the cost function (6.5), to further optimize the model's coefficients, thereby aiming to maximize the overlap between the reconstructed noisy attractor and the model's attractor. The resulting model after optimization is shown in red in figure 6.2(b). We can see that it is now reaching the outer regions. It still fails to reconstruct the finer details of the original attractor, but given the amount of noise present in the data, this is to be expected.

6.3 Estimating ODE coefficients through attractor comparison

If one has a time series and also basic knowledge of the underlying dynamics in the form of a system of ODEs, one is confronted with the problem of estimating those ODE parameters which generated the given time series. Since this is a pretty common task, one can choose among various different methods for this purpose, with the most popular being:

- *Multiple Shooting*: Iteratively adjust parameters and initial conditions on short trajectory segments [5].
- *Synchronization*: Drive model with time series and vary parameters so that synchronization error is minimized [61, 62].
- *Optimization*: Also based on synchronization, but reformulating it as a tracking problem in an optimal control framework, thereby minimizing the required coupling and allowing the usage of powerful algorithms for constrained optimization [1, 20].
- *Nonlinear Filtering*: Approximate temporal evolution of probability density functions in state space [15].

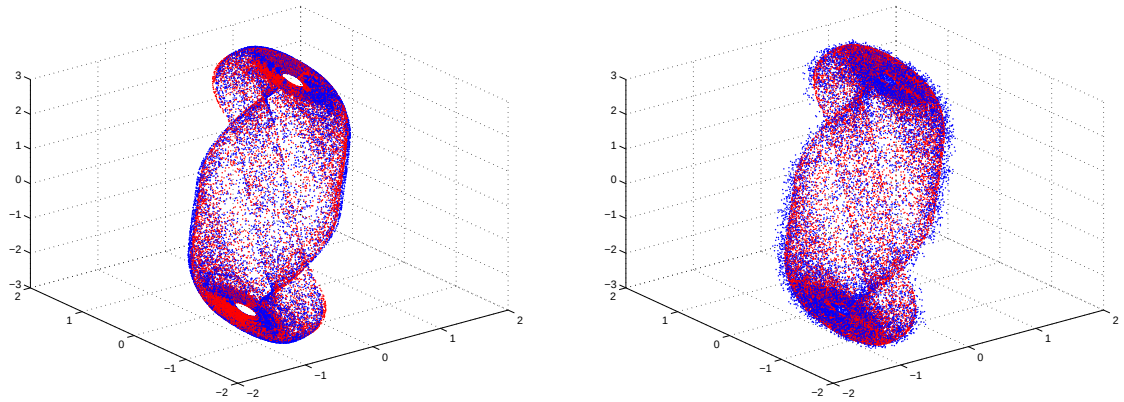
While all those methods have been applied successfully for the identification of ODE parameters of nonlinear dynamical systems, problems often occur when the time series is corrupted by measurement noise. Additionally, since those methods explicitly rely on the time series itself, large sampling times complicate the parameter estimation since the time series is not sufficiently smooth. Since our method only compares attractors, it is not dependent on a smooth time series, as long as the sampling time still allows an unfolding of the attractor in the reconstruction space.

The procedure is the same as in the previous section: We have an observed time series and reconstruct the attractor through a delay embedding. We then optimize the parameters of an ODE (or system of ODEs) through a genetic algorithm, minimizing the cost function (6.5).

6.3.1 Example: Chua oscillator and Rössler system

First, we take numerically generated data from the Chua oscillator (2.69), both with and without added noise. The goal here is to estimate all five parameters $\{\alpha, \beta, \gamma, m_0, m_1\}$ from the ODE system.

As training data, we generate 20000 points from the ODE system, and in addition generate a data set with the same points but corrupted with white noise, resulting in a SNR of 25dB. In figure 6.3, the training data is plotted with blue dots, with (a)



(a) Original data (blue) and fitted ODE system (red) in the absence of noise.

(b) Orignal data (blue), corrupted by noise (SNR 25dB), and fitted ODE system (red).

Figure 6.3: Attractor comparison for the Chua oscillator.

the noiseless and (b) the noisy data set. For the genetic optimization, we start with an initial population of 30000 parameter values and then let the algorithm run over 50 iterations. The number of iterations was limited in this way to yield a reasonable runtime; on a current (but still single-core) desktop PC, the calculation takes about 15 minutes per iteration. In the end, we obtained the following parameter values:

	α	β	γ	m_0	m_1
exact values	9.25	14.29	0.016	-1.138	-0.722
fit (noiseless)	8.62	13.74	0.017	-1.17	-0.65
fit (SNR 25dB)	8.35	13.24	0.011	-1.50	-0.644

The resulting attractor from both models is plotted with red dots in figure 6.3. As expected, the fit from the noiseless data yields better results, but in both cases we get a good approximation of the original attractor.

This procedure can of course also be applied to experimental data. In figure 6.4, we see data recorded from an electronic Chua oscillator (blue dots). Using the same procedure as before, we fit the parameters of the Chua ODE (2.69) to this data, but in addition to five coefficients from the ODE, we now have to incorporate the sampling time dt as a sixth parameter, because we are using the dimensionless Chua equations. The resulting values are

α	β	γ	m_0	m_1	dt
11.86	14.38	0.0144	-1.134	-0.706	0.31

Since we are working with the dimensionless version of the Chua ODE, we cannot directly compare those values to the actual values from the circuit, but in figure 6.4 we can see that we get a good approximation of the attractor from the experimental data.

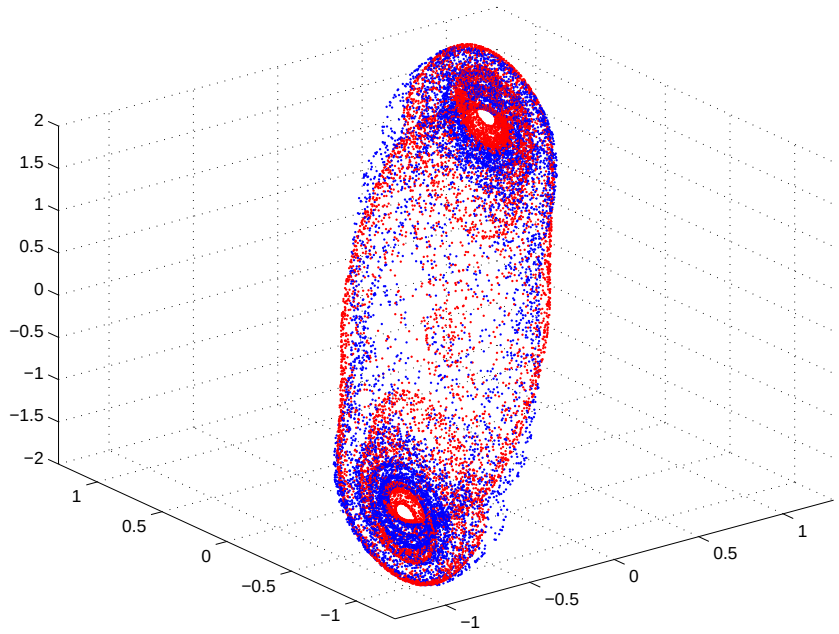


Figure 6.4: Recorded data from a Chua oscillator (blue) with fitted attractor (red).

So far, we had knowledge of the underlying equation which generated the actual data. In practice however, this is often not the case. We will therefore now use a very general ODE of the form

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= z \\ \dot{z} &= c_0 + c_1 x + c_2 y + c_3 z + c_4 xy + \ldots \\ &\quad \ldots + c_{19} \cdot z^3 \end{aligned} \tag{6.6}$$

with 20 parameters $\{c_0, \ldots, c_{20}\}$. The goal is to fit this ODE to data obtained from the Rössler system (2.67). Next to the 20 coefficients of the ODE, we again have to incorporate the sampling time dt. Additionally, we introduce a scaling parameter $\alpha \in \mathbb{R}$, which allows us to scale the resulting time series and therefore the resulting attractor.

The result of the genetic optimization of (6.6) can be seen in figure 6.5, with the blue dots being the original Rössler data set and the resulting model output shown in red. While we do not see a perfect fit, the resulting model attractor exhibits similar features as the original one. Again, the optimization had to be limited, since the calculations involved take quite long.

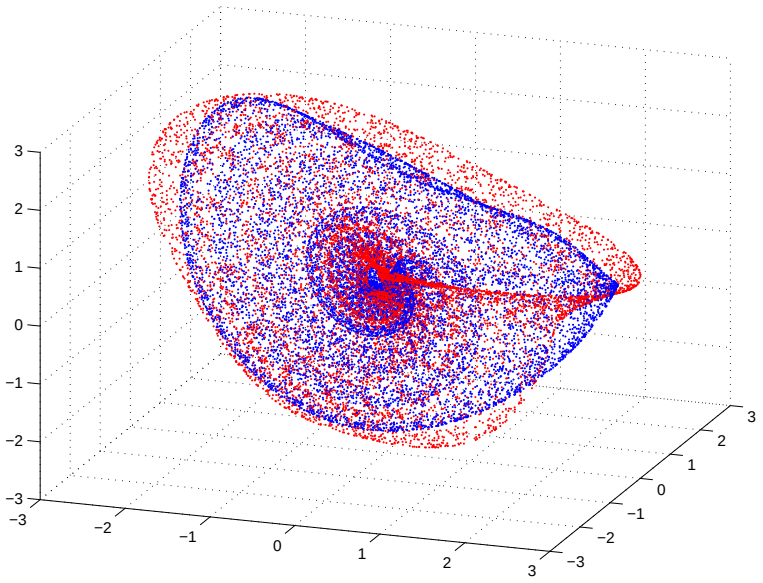


Figure 6.5: Fit of the Rössler attractor through a generic system of polynomial ODEs.

6.3.2 Example: Hindmarsh Rose system and experimental neuron data

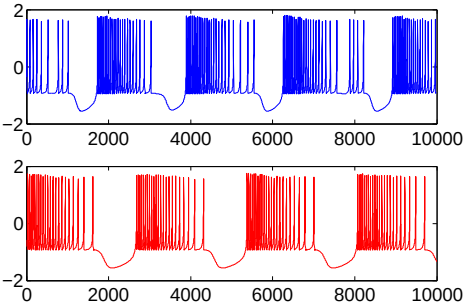
The same procedure shall now be applied to a more complicated problem, namely the Hindmarsh-Rose system, which was developed as a model of neuron activity, with the defining equations being

$$\begin{aligned} \dot{x} &= y + a \cdot x^2 - x^3 - z + I \,, \\ \dot{y} &= 1 - b \cdot x^2 - y \,, \\ \dot{z} &= \varepsilon \left[x - (z - z_0)/4 \right] \,. \end{aligned} \tag{6.7}$$

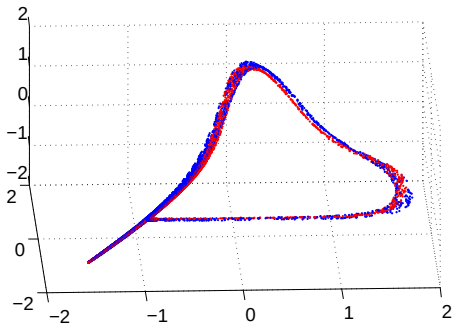
A typical time series is shown in figure 6.6(a). One can see that it consists of two different dynamics: a very fast one, consisting of very narrow spike sequences (so called *bursts*), which are separated by longer periods of a slow relaxation dynamic. The ratio between those two dynamics is defined through the equation for z in (6.7), with $\varepsilon \ll 1$. Dependent on the value for z_0, the system can exhibit different dynamics, including chaos for a small window between $z_0 \approx 3,159$ and $z_0 \approx 3,2$, interspersed with periodic windows [77].

The two different dynamics make this system quite difficult to model, and especially the estimation of the parameter ε is problematic. For modeling, we use a more general form of the Hindmarsh-Rose equations:

$$\begin{aligned} \dot{x} &= c_0 + c_1 x + c_2 x^2 - c_3 x^3 - y - z \\ \dot{y} &= c_4 y + c_5 x + c_6 x^2 \\ \dot{z} &= c_7 z + c_8 x \,. \end{aligned} \tag{6.8}$$



(a) Time series of the Hindmarsh-Rose system: upper plot shows the original training data, whereas the lower plot shows the model output from the optimized ODE.



(b) Original (blue) and model (red) attractor.

Figure 6.6: Fit of the Hindmarsh-Rose system.

The resulting model after optimization is shown in red in figures 6.6(a) (time series) and 6.6(b) (attractor). One can see that both time scales are successfully modeled, albeit not perfectly, since the model's relaxation time is slightly larger.

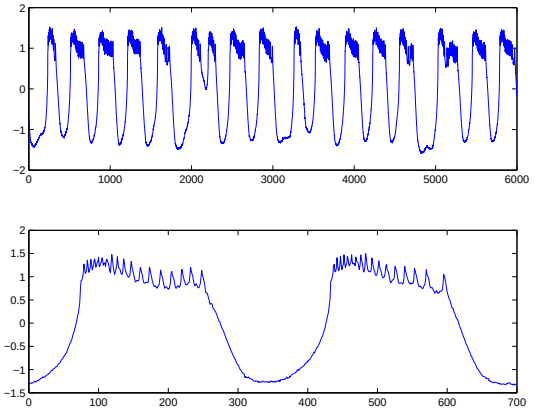
As in the previous section, we now want to apply the same model to measured data. The data set is a time series measured from an isolated neuron from the stomatogastric ganglion of a California spiny lobster [23]. The data set was kindly provided by the group of Henry D.I. Abarbanel from the University of California, San Diego.

The time series is shown in figure 6.7(a), with the upper panel showing the complete training data, and the lower panel containing a zoomed area to show the typical bursting sequences of the neuron. Reconstructing the attractor from this time series in three dimensions yields the plot shown in figure 6.7(b); its shape is rather blurred, suggesting a time series heavily corrupted by measurement noise.

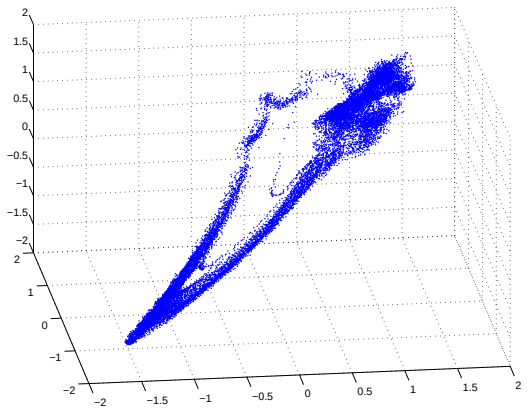
For modeling, we take the ODE equations (6.8) which we used for the Hindmarsh-Rose system, but we add further terms to raise the variance of the model, resulting in the following systems of ODEs:

$$\begin{aligned} \dot{x} &= c_0 + c_1 x + c_2 x^2 - c_3 x^3 - y - z + c4x^4 + c_5 x^5 + c_6 yx + c_7 zx \\ \dot{y} &= c_8 y + c_9 x + c_{10} x^2 + c_{11} x^3 + c_{12} x^4 \\ \dot{z} &= c_{13} z + c_{14} x + c_{15} x^2 + c_{16} x^3 \,. \end{aligned} \tag{6.9}$$

Next to the coefficients $\{c_0, \ldots, c_{16}\}$, we also have to estimate the sampling time dt, leading to a total of 18 parameters. With an initial population size of 80000 parameter sets and 60 iterations of the genetic optimization, we obtain the model shown in figure 6.8. The plot 6.8(a) shows the time series, with the upper panel again showing the experimental data and the lower panel showing the model's output.

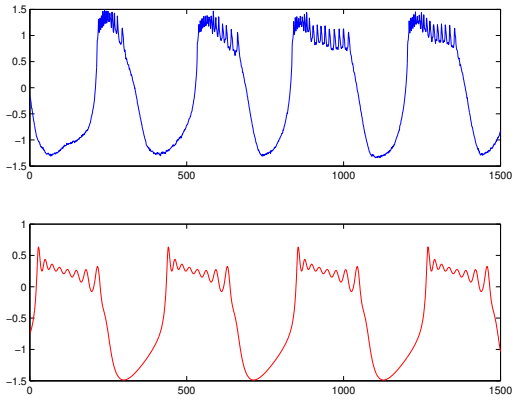


(a) Time series of the experimental neuron data (upper panel: complete data, lower panel: zoomed area).

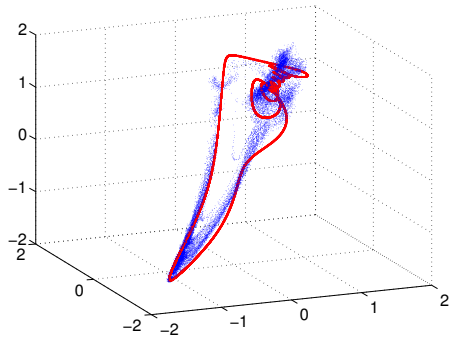


(b) Reconstructed attractor from the experimental neuron data.

Figure 6.7: Experimental neuron data.



(a) Zoomed region of original and model time series (upper and lower panel, resp.).



(b) Original (blue) and model attractor (red).

Figure 6.8: Fit of the experimental neuron data.

Figure 6.8(b) shows the reconstructed attractors of the data and the model in blue and red, resp., where the model attractor turns out to be a limit cycle and hence periodic, which the original system is clearly not. However, we can also see that the burst sequences have a similar structure, with the typical higher frequencies of the spikes at the beginning.

Hence, we can see that despite the large amounts of parameters to be estimated, combined with a very noisy time series as input data, we can still obtain parameters which yield a similar, albeit periodic dynamic.

Chapter 7

Summary and Outlook

The first modeling technique introduced in this thesis was based on nearest neighbors, showing very good results for the prediction of dynamical systems. They are simple to understand, easy to implement (since efficient implementations for nearest-neighbor searching are readily available), and quite robust. However, the data set remains an inseparable part of the model, and for getting best results, a time consuming training of the model is still necessary. In contrast to CWMs, they do not provide a probability distribution as output. While one can extend them to provide an estimate of a distribution, for instance by using sampled distributions, the core of Cluster Weighted Models is based on a probabilistic view of the modeling process, instead of just extending a purely deterministic approach.

While Cluster Weighted Modeling proves to be a valuable tool for black-box modeling of dynamical systems, they are currently seldom used for practical purposes, the reasons for this can only be speculated about. First, they are not straightforward to implement; we hope that we can somewhat alleviate that problem by releasing the code, which was developed during the work on this thesis, as a MATLAB toolbox for everyone's use. Secondly, and more importantly, their model performance is at first glance not superior to simpler modeling strategies, so that it is unlikely that the implementation effort will immediately return the investment in terms of a reduced model error.

It is not immediately clear how the output in form of a probability distribution should be used. Many will simply reduce that distribution to its expectation value, and it was shown that in this case CWMs perform in the same range as nearest-neighbor models, which is already a good result, considering that this can usually achieved by solely choosing a large enough number of clusters and a simple PCTR regularization, combined with cross validation. However, usage of the full probability distribution allows for a better understanding of the model's output, as was shown for the numerical Chua data, and can be used for procedures like Active Learning in a straightforward way.

CWMs are a global modeling technique, which over nearest-neighbor models has the advantage that they can be efficiently evaluated after training, and that the data

set is not needed anymore after the model parameters have been optimized. One problem of CWMs is the large number of parameters that have to be estimated: in addition to the cluster positions and sizes as well as their weights, we also need the coefficients of the linear parametrized local models. As a result, CWMs are prone to overfitting. It was shown that the risk of overfitting can be minimized by using different regularization techniques, tailored at the different types of parameters, in combination with cross validation and early stopping. In addition of avoiding overfitting, this often also results in a lower Ignorance score.

There is still one area though, where the purely local approach from nearest-neighbor models has a prime advantage: the training on the multi-step prediction error, which not only leads to lower errors on iterated predictions, but makes the model more stable against accumulated errors which inevitably occur. This was a problem for predicting the friction data, since here a freely iterated model was needed, and a single CWM turned out to be unstable. It was shown that using a CWM ensemble can deal with that problem, albeit at higher computational cost. During the work on CWMs, different approaches were tried for a direct multi-step prediction without the need for an ensemble, for example by first training a CWM on the single-step error, then a "re-training" of the model on a further step, but this also did not result in a stable model. However, this is a basic problem practically all global modeling techniques share, not only CWMs.

To compare different probabilistic model approaches, we argued that the mean squared error, operating only on the expectation value, is not sufficient for evaluating a model's performance in the probabilistic context. Instead, probabilistic scores should be used, in particular the Ignorance score. It was shown that CWMs perform very well in this context; they combine the purely stochastic approach with a deterministic model and hence perform well for both kinds of problems, as was demonstrated on numerical and experimental data with different signal-to-noise ratios.

The simple calculation of quantities derived from the distribution, like the model's variance, make CWMs well suited for Active Learning, where new data points are selected during training. Different criteria for point selection were shown, and demonstrated on an example for detecting a global minimum surrounded by local minima. As a future work, it would be interesting to see how those criteria behave for different modeling problems. Also, many criteria in Active Learning are not easily usable in high-dimensional input spaces. One particular problem is the calculation of extremal points of the CWM's model output and variance, which for higher-dimensional systems cannot be done anymore through the evaluation of the model on a grid. The special structure of CWMs with their underlying clustering might make it possible to find an efficient algorithm for this problem. Another possibility would be to use a procedure based on genetic algorithms as follows: A genetic algorithm is employed to generate new possible candidates for querying the experiment, but those are presented first to an already trained CWM ensemble. The ensemble's output and variance for those points is calculated, and then candidates are selected according to the criteria described in section 5.2. This should greatly reduce the number of

queries for the the actual experiment.

The method of comparing attractors in phase space was used to estimate coefficients for ordinary differential equations. On several different numerical and experimental data sets, it was shown that it is possible to obtain a model with qualitatively similar features. The main problem of this algorithm is its high computational cost, especially when the method is used were found points are removed from the data set, since then an efficient nearest-neighbor searcher cannot be used. The solution to this problem would be to find an efficient way of removing points from the searcher's data structure, for example from the search tree built in the pre-processing stage of the ATRIA algorithm. Still, it should be mentioned that this algorithm is still faster than techniques which rely on an actual density estimate of the two attractors, e.g. through a kernel density estimator.

Another interesting project would be to combine the attractor comparison with Cluster Weighted Models, also to stabilize their behavior for freely iterated prediction; however, the large number of parameters in CWMs greatly complicate this task. It would first be necessary to reduce CWMs to their essential clusters, maybe through a "pruning" algorithm which in a first step simply removes those clusters with a small weight. It would then be necessary to identify those parameters which have the most impact on the attractor shape, and then the optimization should concentrate on those.

Theoretically, Cluster Weighted Models allow mixing different model types, i.e., one can use linear models in some clusters and quadratic models in others. Also theoretically, the clusters should target those positions for which they are best suited. In practice however, this has been found not to be the case. The reason is that during EM optimization, clusters more or less smoothly move in input space, so that if a cluster with a linear model already occupies a certain space, it cannot easily be replaced by a cluster with another model, even if it would be better suited for that area. It might be beneficial to include a stochastic element into the modeling process, which either allows clusters to switch positions or to change their model type.

Appendix A

The Interacting Particle Filter

To define the IPF, consider

$$\pi_n^w(A) := \text{Prob}(Y_{n-w} \in A | X_{1:n}),$$

where w is referred to as the *relaxation window*. In this notation, the filtering process π_n and the quantity π_n^+ of section 3.3.4 are given as $\pi_n(A) = \pi_n^0(A)$ and $\pi_n^+(A) = \pi_n^{(-1)}(A)$, respectively.

Similar to (3.16) and (3.17), it holds that

$$\pi_n^{w-1}(A) = \pi_n^w(F^{-1}(A)), \tag{A.1}$$

$$\pi_{n+1}^w(A) = \frac{\int_A g(F^w(x), X_{n+1}) \mathrm{d}\pi_n^{w-1}(x)}{\int_{\mathbb{R}^d} g(F^w(x), X_{n+1}) \mathrm{d}\pi_n^{w-1}(x)}. \tag{A.2}$$

Here F^w stands for the w–fold composition of F. The idea of the IPF is to sample from $\pi_{n+1}^w(y)$, assuming that $\pi_n^w(y)$ has the form of a kernel estimate

$$\pi_n^w(y) \cong \frac{1}{M} \sum_{i=1}^{M} \frac{1}{\sigma_i^d} K\left(\frac{y - y_i}{\sigma_i}\right), \tag{A.3}$$

and subsequently to use these samples to approximate $\pi_{n+1}^w(y)$ again by a kernel estimate of the form (A.3). This is implemented by the following algorithm:

1. At any one time n, we have an ensemble of M particles $\{y_1 \dots y_M\}$. Choose one $i \in \{1 \dots M\}$ at random.

2. Drawing a standard d–dimensional normal variable r, we get a sample from $\pi_n^w(y)$ in the approximation (A.3) by setting

   $$\eta := y_i + \sigma_i r.$$

   A sample from $\pi_n^{w-l}(y)$ for any desired l is obtained by $F^l(\eta)$.

3. Now draw a random variable δ with uniform distribution on the unit interval. The sample η gets *accepted* if

$$\gamma\left(\frac{X_{n+1} - cF^{w+1}(\eta)}{\sigma}\right) \leq \delta \max_z \gamma(z),$$

otherwise it is *rejected*. Here, γ is the density of the measurement noise, as defined at the beginning of Subsec. 3.3.1. Repeat steps (i)–(iii) until M samples have been accepted. The accepted samples form a sample from $\pi_{n+1}^{w+1}(y)$.

4. To get a sample from $\pi_{n+1}^{w}(y)$, apply the dynamics once to the accepted samples. Then return to step 1, replacing n with $n+1$.

The original IPF had neither a relaxation window nor any regularization, that is, no perturbation r was added. Some particles will certainly get selected more than once. This would not be a problem if the system had a stochastic component itself. Since in our case the system is deterministic though, adding no dynamic noise would cause some particles to occupy the same point in state space after resampling. We would effectively end up with fewer and fewer particles until only one is left. The idea of regularization is extensively discussed in [57].

The relaxation window helps to get samples of π_n which lie on the attractor. According to step 2 of the algorithm, to get a sample from π_n (i.e. π_n^0) we have to apply the dynamics w times to the "working" sample η, causing it to fall onto the attractor. Using a relaxation window is, to our knowledge, a new idea. Note that the IPF with relaxation window is *not* the same as applying filtering to *strands of orbits* rather than single states. Although filtering strands of orbits is a probably useful idea, applying the IPF to strands is unlikely to work straight away. The reason is that every observation would be taken into account more than once (to be precise, l times if l is the length of the strands), which is likely to result in too narrow ensembles or overconfident forecasts.

Bibliography

[1] H. D. I. Abarbanel, D. R. Creveling, R. Farsian, and M. Kostuk. Dynamical state and parameter estimation. *SIAM Journal on Applied Dynamical Systems*, 8(4):1341–1381, 2009.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[3] J. Argyris, G. Faust, and M. Haase. *Die Erforschung des Chaos*. Vieweg-Verlag, Braunschweig, Wiesbaden, 1995.

[4] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[5] E. Baake, M. Baake, H. Bock, and K. Briggs. Fitting ordinary differential equations to chaotic data. *Physical Review A*, 45(8):5524–5529, 1992.

[6] V. Babovic and D. R. Fuhrman. Data assimilation and error prediction using local models. *D2K Technical Report 0401-2*, 2001.

[7] P. J. Barral, A. Hasmy, J. Jiménez, and A. Marcano. Nonlinear modeling technique for the analysis of DNA chains. *Physical Review E*, 61(2):1812–1815, 2000.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] S. A. Billings and K. L. Lee. A smoothing algorithm for nonlinear time series. *International Journal of Bifurcation and Chaos*, 14(3):1037–1051, 2002.

[10] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[11] C. Borgelt and R. Kruse. Shape and size regularization in expectation maximization and fuzzy clustering. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, pages 52–62. Springer Verlag Berlin/Heidelberg, 2004.

[12] G. W. Brier. Verification of forecasts expressed in terms of probabilities. *Monthly Weather Review*, 78(1), 1950.

[13] D. S. Broomhead and G. P. King. Extracting qualitative dynamics from experimental data. *Physica D*, 20, 1986.

[14] J. Bröcker. Decomposition of proper scores. *Max Planck Institut für Physik komplexer Systeme (Tech. Rep.)*, 2007.

[15] J. Bröcker, U. Parlitz, and M. Ogorzalek. Nonlinear noise reduction. *Proceedings of the IEEE*, 90(5):898–918, 2002.

[16] J. Bröcker and L. A. Smith. Scoring probabilistic forecasts: The importance of being proper. *Weather and Forecasting*, 22(2):382–388, 2007.

[17] K.-S. Chan and H. Tong. *Chaos, a statistical perspective*. Springer Verlag, New York, 2001.

[18] L. O. Chua, M. Komuro, and T. Matsumoto. The double scroll family. *IEEE Trans. Circuits and Systems-I*, 33:1072–1118, 1986.

[19] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[20] D. R. Creveling, P. E. Gill, and H. D. I. Abarbanel. State and parameter estimation in nonlinear systems as an optimal tracking problem. *Physics Letters A*, 372(15):2640–2644, 2008.

[21] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[22] U. Dressler, T. Ritz, A. Schenck zu Schweinsberg, R. Doerner, B. Hübinger, and W. Martienssen. Tracking unstable periodic orbits in a bronze ribbon experiment. *Physical Review E*, 51(3):1845–1848, 1995.

[23] R. C. Elson, A. I. Selverston, R. Huerta, N. F. Rulkov, M. I. Rabinovich, and H. D. I. Abarbanel. Synchronous behavior of two coupled biological neurons. *Physical Review Letters*, 81(25):5692–5695, 1998.

[24] J. D. Farmer and J. J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, 1987.

[25] V. V. Fedorov. *Theory of Optimal Experiments*. Academic Press, New York, 1972.

[26] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[27] K. Geist, U. Parlitz, and W. Lauterborn. Comparison of different methods for computing Lyapunov exponents. *Progress in Theoretical Physics*, 83(5):875–893, 1990.

[28] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[29] N. Gershenfeld. *The nature of mathematical modeling*. Cambridge University Press, New York, 1999.

[30] N. Gershenfeld, B. Schoner, and E. Metois. Cluster-weighted modeling for time series analysis. *Nature*, 397:329–332, 1999.

[31] T. Gneiting and A. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102:359–378, 2007.

[32] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA., 1989.

[33] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996.

[34] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society*, XIV(1):107–114, 1952.

[35] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/nongaussian bayesian state estimation. *IEEE Proceedings, Part F*, 140(2), 1993.

[36] T. Hastie, R. Tibshirani, and J. Friedmann. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2001.

[37] B. Hübinger, R. Doerner, W. Martienssen, M. Herdering, R. Pitka, and U. Dressler. Controlling chaos experimentally in systems exhibiting large effective lyapunov exponents. *Physical Review E*, 50(2):932–948, 1994.

[38] B. Hugueney, B. Bouchon-Meunier, and G. Hébrail. Fuzzy long term forecasting through machine learning and symbolic representations of time series. In B. Reusch, editor, *Computational Intelligence, Theory and Applications: International Conference 8th Fuzzy Days in Dortmund*, pages 109–123. Springer Verlag, Berlin, 2005.

[39] A. Ihler. Kernel density estimation toolbox for matlab. URL: http://ssg.mit.edu/ ihler/code/kde.shtml.

[40] A. H. Jazwinsky. *Stochastic Processes and Filtering Theory*. Academic Press, 1970.

[41] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, New York, 1986.

[42] K. Judd and A. Mees. Embedding as a modeling problem. *Physica D*, 120:273–286, 1998.

[43] K. Jänich. *Analysis für Physiker und Ingenieure*. Springer, Berlin, 1995.

[44] G. Kallianpur. *Stochastic Filtering Theory*. Springer Verlag, 1980.

[45] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Ser. D: J. Basic Eng.*, 82:35–45, 1960.

[46] G. Langer. *Modellierung von Parameterabhängigkeiten nichtlinearer dynamischer Systeme*. Dissertation, Georg-August Universität Göttingen, 2003.

[47] J. Levine and G. Pignie. Exact finite–dimensional filters for a class of nonlinear discrete–time systems. *Stochastics*, 18:97–132, 1986.

[48] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *roceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag, 1994.

[49] T. Lindén, F. García, A. Heikkinen, and S. Lehti. Optimizing neural network classifiers with ROOT on a rocks linux cluster. In *PARA'06: Proceedings of the 8th international conference on Applied parallel computing*, pages 1065–1073, Berlin, Heidelberg, 2007. Springer-Verlag.

[50] B. A. Mair, R. Murali, and J. M. M. Anderson. A minimum chi-squared method for indirect parameter estimation from poisson data. *Statistica Neerlandica*, 56(2):165–178, 2002.

[51] G. McLachland and D. Peel. *Finite Mixture Models*. John Wiley and Sons, New York, 2000.

[52] J. McNames. *Innovations in Local Modeling for Time Series Prediction*. Ph.D. thesis, Stanford University, 1999.

[53] J. McNames. A fast nearest neighbor algorithm based on a principal axes tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):964–976, 2001.

[54] J. McNames, J. A. K. Suykens, and J. Vandewalle. Winning entry of the K. U. Leuven time series prediction competition. *International Journal of Bifurcation and Chaos*, 9(8):1485–1500, 1999.

[55] C. Merkwirth, U. Parlitz, and W. Lauterborn. Fast nearest-neighbor searching for nonlinear signal processing. *Physical Review E*, 62(2):2089–2097, 2000.

[56] A. H. Murphy. Hedging and skill scores for probability forecasts. *Journal of Applied Meteorology*, 12, 1973.

[57] C. Musso, N. Oudjane, and F. Le Gland. Improving regularized particle filters. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 247–271. Springer Verlag, 2001.

[58] E. Ott. *Chaos in dynamical systems*. Cambridge University Press, 1993.

[59] U. Parlitz. Identification of true and spurious lyapunov exponents from time series. *International Journal of Bifurcation and Chaos*, 2:155–165, 1992.

[60] U. Parlitz, A. Hornstein, D. Engster, F. Al-Bender, V. Lampaert, T. Tjahjowidodo, S. D. Fassois, D. Rizos, C. X. Wong, K. Worden, and G. Manson. Identification of pre-sliding friction dynamics. *Chaos*, 14(2):420–430, 2004.

[61] U. Parlitz, L. Junge, and L. Kocarev. Synchronization-based parameter estimation from time series. *Physical Review E*, 54(6):6253–6259, 1996.

[62] M. Pecora and T. L. Carroll. Synchronization in chaotic systems. *Physical Review Letters*, 64(8):821–824, 1990.

[63] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761 – 767, 1998.

[64] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, Cambridge, 1992.

[65] N. Roy and A. Mccallum. Toward optimal active learning through sampling estimation of error reduction. In *Proc. 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, 2001.

[66] D. Ruelle. *Turbulence, strange attractors, and chaos*. World Scientific, 1996.

[67] T. Sauer, J. A. Yorke, and M. Casdagli. Embedology. *Journal of Statistical Physics*, 65(4):579–616, 1991.

[68] L. J. Savage. Elicitation of personal probabilities and expectation. *Journal of the American Statistical Association*, 66(336), 1971.

[69] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[70] B. Schoner and N. Gershenfeld. Cluster-weighted modeling: Probabilistic time series prediction, characterization and synthesis. In A. I. Mees, editor, *Nonlinear Dynamics and Statistics*, pages 365–385. Birkhäuser, Boston, 2001.

[71] A. Schenck zu Schweinsberg, T. Ritz, U. Dressler, B. Hübinger, R. Doerner, and W. Martienssen. Quasicontinuous control of a bronze ribbon experiment using time-delay coordinates. *Physical Review E*, 55(3):2145–2158, 1997.

[72] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, New York, NY, USA, 1992. ACM.

[73] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[74] B. W. Silverman, M. C. Jones, D. W. Nycha, and J. D. Wilson. A smoothed EM approach to indirect estimation problems, with particular reference to stereology and emission tomography. *Journal of the Royal Statistical Society. Series B (Methodological)*, 52(2):271–324, 1990.

[75] W.-H. Steeb. *Chaos und Quantenchaos in dynamischen Systemen*. BI Wissenschaftsverlag, Zürich, 1994.

[76] F. Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*. Springer Verlag, Berlin, 1981.

[77] X.-J. Wang. Genesis of bursting oscillations in the Hindmarsh-Rose model and homoclinicity to a chaotic saddle. *Physica*, D(62):263–274, 1996.

[78] W. K. Wong. On the equivalence of D and G-optimal designs in heteroscedastic models. *Statistic & Probability Letters*, 25(4):317–321, 1995.

[79] J. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[80] Z. H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

Danksagung

An erster Stelle möchte ich Prof. Parlitz herzlich danken, der diese Arbeit ermöglicht und betreut hat.

Ich danke Herrn Prof. Ronneberger für die Aufnahme in das Graduiertenkolleg Strömungsinstabilitäten und Turbulenz. Herrn Prof. Kress danke ich für die Aufnahme in das Graduiertenkolleg Identifikation in mathematischen Modellen; den Herren Prof. Schaback und Prof. Wendland danke ich für die dortige Betreuung.

Herrn Prof. Wörgötter danke ich für die Übernahme des Korreferats dieser Arbeit.

Viele Menschen im Dritten Physikalischen Institut haben durch ihre Unterstützung maßgeblichen Anteil an dieser Arbeit. Insbesondere zu nennen ist hier Prof. Lauterborn als langjähriger Leiter. Neben Unterstützung danke ich für die manchmal auch notwendige Ablenkung den Kollegen Jörg Dittmar, Daniel Schanz, Philipp Koch und Karsten Köhler. Jochen Bröcker danke ich für die Einweisung in die Kunst des Scorings und die gute Zusammenarbeit.

Lieben Dank gebührt meinen Eltern, meinem Bruder und besonders meiner Gefährtin Dana. Ohne sie alle wäre diese Arbeit nicht möglich gewesen. Ich danke Dana vor allem für die vielen schlaflosen Nächte der letzten Monate mit unserem Sohn. In Zukunft wird auch der Vater wieder Nachtschichten übernehmen.

Lebenslauf

Name	David Engster
Geburtsdatum	25.02.1975 in Göttingen
Eltern	Hermann und Doris Engster
Geschwister	Frank Engster, geb. 27.02.1972
Staatsangehörigkeit	Deutsch
Familienstand	In Partnerschaft lebend, ein Kind

1981–1985	Besuch der Grundschule Höltyschule, Göttingen
1985–1986	Besuch der Orientierungsstufe Weende-Nord,Göttingen
1986–1994	Besuch des Hainberg-Gymnasium, Göttingen
1994	Abschluss mit der Allgemeinen Hochschulreife
1994–1995	Zivildienst im Klinikum Göttingen und in der Beratungsstelle für Multiple Sklerose
1996	Diplomstudiengang Physik an der RWTH Aachen
1996–2002	Fortsetzung des Studiums an der Georg-August-Universität, Göttingen
06/2002	Erwerb des Hochschulgrades Diplom-Physiker
07/2002–03/2003	Stud. Hilfskraft am III. Physikalischen Institut zur Implementierung von Modellbildungs-Algorithmen
04/2003	Beginn der Promotion an der Universität Göttingen
04/2003–03/2005	Stipendiat des Graduiertenkollegs Strömungsinstabilitäten und Turbulenz
04/2005–03/2006	Stipendiat des Graduiertenkollegs Identifikation in mathematischen Modellen
04/2006–04/2010	Wissenschaftliche Hilfskraft bei der Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen (GWDG)
seit 07/2010	Wissenschaftlicher Mitarbeiter bei der GWDG

Göttingen, den 14.07.2010 _____

(David Engster)