Probabilistic Characterization and Synthesis of Complex Driven Systems

by

Bernd Schoner

Diplom-Ingenieur
Rheinisch-Westphälische Technische Hochschule Aachen - Germany (1996)

Ingénieur des Arts et Manufactures
Ecole Centrale de Paris - France (1996)

Submitted to the Program in Media, Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2000

Author .....
Program in Media, Arts and Sciences,
School of Architecture and Planning
August 4, 2000

Certified by .....
Neil A. Gershenfeld
Associate Professor of Media Technology
Thesis Supervisor

Accepted by .....
Stephen A. Benton
Chairman, Department Committee on Graduate Students

Probabilistic Characterization and Synthesis of Complex Driven Systems

by

Bernd Schoner

Submitted to the Program in Media, Arts and Sciences,
School of Architecture and Planning
on August 4, 2000, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Real-world systems that have characteristic input-output patterns but don't provide access to their internal states are as numerous as they are difficult to model. This dissertation introduces a modeling language for estimating and emulating the behavior of such systems given time series data. As a benchmark test, a digital violin is designed from observing the performance of an instrument.

Cluster-weighted modeling (CWM), a mixture density estimator around local models, is presented as a framework for function approximation and for the prediction and characterization of nonlinear time series. The general model architecture and estimation algorithm are presented and extended to system characterization tools such as estimator uncertainty, predictor uncertainty and the correlation dimension of the data set. Furthermore a real-time implementation, a Hidden-Markov architecture, and function approximation under constraints are derived within the framework.

CWM is then applied in the context of different problems and data sets, leading to architectures such as cluster-weighted classification, cluster-weighted estimation, and cluster-weighted sampling. Each application relies on a specific data representation, specific pre and post-processing algorithms, and a specific hybrid of CWM.

The third part of this thesis introduces data-driven modeling of acoustic instruments, a novel technique for audio synthesis. CWM is applied along with new sensor technology and various audio representations to estimate models of violin-family instruments. The approach is demonstrated by synthesizing highly accurate violin sounds given off-line input data as well as cello sounds given real-time input data from a cello player.

Thesis Supervisor: Neil A. Gershenfeld
Title: Associate Professor of Media Technology

Doctoral Dissertation Committee

Thesis Supervisor

Neil Gershenfeld
Associate Professor of Media Technology
MIT Media Laboratory

Thesis Reader

John Harbison
Institute Professor of Music
Massachusetts Institute of Technology

Thesis Reader

Terrence Sejnowski
Professor - The Salk Institute
Investigator - Howard Hughes Medical Institute
La Jolla, CA

Acknowledgments

I would like to thank

my advisor Neil Gershenfeld for giving me a place in his group and pushing me along all these years despite my pessimistic German attitude. Neil taught me many lessons including patience, the sense that science takes time, and appreciation for the making of physical objects. I'm deeply indebted to his educational mission.

my readers John Harbison for sharing his sensibility and knowledge about violins and Terry Sejnowski for contributing his technical and experimental expertise. My committee couldn't have been more versatile.

my collaborators Charles Cooper, for bringing his wisdom, manifested in lots of white hair, to the project as well as doing some really important hardware design, and Chris Douglas, Tuomas Lukka, and Edward Boyden for being young and brilliant.

my helpful early colleagues Eric Metois, Josh Smith, and Vadim Gerasimov, who got me started doing real work after years of theory.

Joe Paradiso for designing the initial violin bow which let me have a head start.

my patient violinists Julia Ogrydziak, Sandy Choi, Teresa Marrin, Romy Shioda, and Diana Young, whose musical talent I shamelessly abused by making them play my unreliable violin interfaces.

Rehmi Post my first, only and best officemate ever.

my fellow grad students Pehr Anderson, Henry Chong, Ben Denckla, Elektrizitäts - künstlerin Kelly Dobson, Rich Fletcher, Yael Maguire, Teresa Marrin, Eric Metois, Femi Omojola, Ravi Pappu, Rehmi Post, Matt Reynolds, Joey Richards, Peter Russo, Josh Smith, Jason Taylor, Christopher Verplaetse, Ben Vigoda, for being wonderfully helpful colleagues.

Mary Farbood for reading and correcting this document and for much support with the final thesis and defense preparation including a recording session of *Stars and Stripes Forever*.

my editors and colleagues Tristan Jehan and Diana Young.

my chamber music teachers David Deveau, John Harbison, and Marcus Thompson and my fellow musicians including Annie Chen, Nina Chen, Elaine Chew, Mary Farbood, Teresa Marrin, Jaemin Rhee, and Jason Wong, for many unforgettable moments at MIT.

Rafaela Arriaga, Emmanuel Barroux, Susza Bereznai, Dominik Bless, Thomas Bollein - inger, Wolfram Brinker, Götz Frömming, Stefan Götz, Armin Haindl, Anne Harley, Hans-Michael Hauser, Holger Kist, Stephan Koehler, Klaus Langhammer, Nicolas Lemoine, Jochen Linck, Dirk Lumma, Dominik Meiering, Jean-Stéphane Mesonnier, Thomas Rasch, Maximilian Riesenhuber, Noah and Seth Riskin, Julia Schmitt, Volker Schott, Thomas Weber, Stefan Welsch, and Anna Yook, for being my friends.

the Flying Karamazov Brothers, Howard, Kristina, Mark, Paul and Rod.

my father Konrad and mother Adelinde for giving me unconditional, loving support; my sister Ellen and brother Viktor; my grandmothers Marie and Amalie for their understanding; my aunt Hanna for her care and beautiful sculpture work and all my other family members.

Thank you all.

Contents

1 Introduction **10**
1.1 The problem 10
1.2 Outline 13

I Cluster-Weighted Modeling **15**

2 Background and related work **17**
2.1 Graphical models 17
2.1.1 Notation and theory 18
2.1.2 Inference 21
2.1.3 Example I: Gaussian mixture models 24
2.1.4 Example II: Hidden Markov models 26
2.2 Nonlinear dynamic systems 28
2.2.1 State space reconstruction 29
2.2.2 Input-output embedding 31
2.2.3 Characterization of nonlinear dynamic systems 32

3 The cluster-weighted modeling framework **34**
3.1 Model architecture 34
3.2 Model estimation 39
3.3 Model evaluation and system characterization 44
3.4 Online implementation 46
3.4.1 Online EM updates 46
3.4.2 Cluster-weighted Kalman-filter updates for the local models 47
3.5 A cluster-weighted input-output hidden Markov model 49
3.6 Function approximation under constraints 53

II Synthesis Architectures and Applications **60**

4 Classification **62**
4.1 Classification and decision theory 62
4.1.1 Linear and nonlinear classifiers 64
4.1.2 Feature extraction 66

4.2 Cluster-weighted classification . . . 69
4.2.1 Simplifications . . . 70
4.3 Applications . . . 72
4.3.1 Stimulus detection from MEG brain data . . . 72
4.3.2 Consumer fraud characterization . . . 74
4.3.3 Electronic noses . . . 76

5 Prediction and Estimation **79**
5.1 Background . . . 79
5.1.1 Estimation Theory . . . 79
5.1.2 Linear models . . . 82
5.1.3 Linear Regression . . . 83
5.1.4 Generalized linear and polynomial models . . . 84
5.2 Cluster-weighted estimation . . . 86
5.3 Applications . . . 87
5.3.1 Predicting physical systems . . . 87
5.3.2 Prediction of the M3-competition time series . . . 88

6 Linear predictive coding **93**
6.1 Introduction and related work . . . 93
6.2 Cluster-weighted linear predictive coding . . . 95
6.3 Application: An excitation-filter model of the violin . . . 97

7 Frequency domain estimation **99**
7.1 Frequency domain representations of signals and systems . . . 99
7.1.1 Volterra expansion . . . 101
7.2 Cluster-weighted complex estimation . . . 103
7.3 Application: nonlinear device characterization . . . 106

8 Sampling **109**
8.1 Algorithmic pieces and related work . . . 109
8.2 Cluster-weighted sampling . . . 112

III Data-Driven Modeling of Musical Instruments **117**

9 Related work in musical synthesis **119**
9.1 Synthesis algorithms . . . 119
9.1.1 Physical modeling . . . 119
9.1.2 Sinusoidal analysis/synthesis . . . 123
9.1.3 Wavetable synthesis . . . 126
9.1.4 More synthesis techniques... . . . 127
9.2 Connectionism and musical applications . . . 129
9.3 Synthesis evaluation . . . 130

10 The digital stradivarius **133**
10.1 General concept and chronology . . . 133
10.1.1 Background . . . 133
10.1.2 The goal . . . 134
10.1.3 Defining the measurements . . . 136
10.1.4 Embedding and musical synthesis . . . 137
10.1.5 A working hello-world: the arm demo . . . 138
10.1.6 A cellist playing a single-string violin . . . 141
10.1.7 The final approach . . . 143
10.2 Data-collection . . . 144
10.3 Data-analysis . . . 146
10.4 Inference model . . . 149
10.4.1 Representation . . . 149
10.4.2 Feature selection . . . 153
10.5 Experimental Results . . . 155
10.5.1 Discussion . . . 155
10.6 Artistic extensions and applications . . . 157
10.6.1 A universal violin controller interface . . . 158
10.6.2 Marching Cello . . . 159
10.6.3 A new violin MIDI controller . . . 159
10.6.4 Flexible musical instruments: CD-player versus Strad . . . 160

11 Conclusions **162**
11.1 Contributions . . . 162
11.2 Future Work . . . 164

A Three views of the EM algorithm **165**
A.1 Gradient approach . . . 165
A.2 Information theoretic approach . . . 167
A.3 Differential-geometric approach . . . 168
A.3.1 The curved space of exponential distributions . . . 168
A.3.2 EM and em algorithm . . . 170

B Sensor technology and hardware design **173**
B.1 Sensor technology . . . 173
B.1.1 Sensing the bow . . . 173
B.1.2 Sensing fingers on the fingerboard . . . 175
B.1.3 Recording violin audio signals . . . 175
B.2 Hardware packaging . . . 176
B.2.1 Violin sensor board . . . 176
B.2.2 Instrumenting violin and cello . . . 177
B.2.3 Instrumenting the Marching Cello . . . 177
B.3 Sensor calibration . . . 177

C Code **183**
C.1 Cluster-weighted modeling implementations 183
C.1.1 Matlab interface 183
C.1.2 C interface 188
C.2 Sinusoidal synthesis 190
C.3 Wavetable synthesis 191

Chapter 1

Introduction

> Die Grammophonplatte, der musikalische Gedanke, die Notenschrift, die Schallwellen, stehen alle in jener abbildenden internen Beziehung zu einander, die zwischen Sprache und Welt besteht.
> Ihnen allen ist der logische Bau gemeinsam.
> (Wie im Märchen die zwei Jünglinge, ihre zwei Pferde und ihre Lilien. Sie sind alle in gewissem Sinne Eins.)
> L. Wittgenstein, TRACTATUS LOGICO-PHILOSOPHICUS.[1]

1.1 The problem

This thesis is about the emulation, prediction, and characterization of real-world dynamic systems on digital machines. It is about teaching machines how to simulate and predict physical objects and phenomena given some observation of their state.

Although computers have become many orders of magnitude faster since the idea was first conceived that they could understand and emulate the world in a human-like fashion, computers appear to be as dumb today as they were then. They know practically nothing about what the physical world is like, about how to make an intelligent decision, and about how people like to be treated. What's worse, computers have no notion of how to acquire these insights. Unless we explicitly tell them every single detail about a phenomenon, a process, or human behavior, they remain ignorant.

The aim of this thesis is to provide the physical and logical tools that enable computers to acquire and process data from observed physical systems. It is motivated by the understanding that the ignorance of computers regarding representations of the world and their inability to learn from observation isn't due to a lack of computational resources. Instead, what's missing are devices that give machines access to the right kinds of information re-

[1] 4.014: The gramophone record, the musical thought, the score, the waves of sound, all stand to one another in that pictorial internal relation, which holds between language and the world.
To all of them the logical structure is common.
(Like the two youths, their two horses and their lilies in the story. They are all in a certain sense one.)
L. Wittgenstein, *TRACTATUS*.

garding the objects and processes around them [Min85]. Comparing machine performance to human intelligence is unfair as long as humans can see, hear, touch, and smell, while machines are only fed some context-free numbers. In addition, algorithms are needed that handle the given data efficiently, transparently, and in the right context. The black box approach of artificial neural networks is appealing but too crude when it comes to building a flexible methodology that, for example, reuses earlier experience. In this dissertation, we will present work on sensing instrumentation for some specific applications but will focus on the logical and numerical means to predict or synthesize systems in the digital medium.

The list of worthwhile systems to model and forecast is about as long as the list of unsuccessful attempts to do so. This is astounding given that there is a body of results and algorithmic tools that let us characterize and predict dynamic systems. Linear systems theory, for example, provides an exhaustive methodology for any system that can be described by linear equations. The theory is well established, has generated a multitude of useful results, and is taught and used in virtually every scientific discipline. Moreover, although it provides different tools for different ends, these tools are wonderfully described in a common language.

Where the assumption of linearity breaks, dynamic systems theory loses its homogeneity. The lack of a uniform theory for nonlinear systems is partly due to the definition of the field in the negative. The term *non*-linear does not define a class of problems but rather a class which is *not* something else, i.e. linear [Bos99]. Naturally this is a difficult starting point. However, the theoretical insights into how much we can know about a nonlinear system are still remarkable. The reconstruction (embedding) theorem provides the theoretical means to handle highly nonlinear behavior of arbitrary physical systems with hidden dynamics [Tak81]. It proves that a system's state space can be mapped into a diffeomorphic space, constructed from any observable of the system, and that the system can be characterized with respect to dimensionality and dynamic behavior in this reconstructed space. The reconstruction theorem also detects low-dimensional structure in a high-dimensional data space, modeling the effective degrees of freedom of a system rather than its mechanical degrees of freedom. In the case of a physical device such as the violin, this means that we do not need to represent every single fiber of wood, rather we can model the larger picture of input signals, effective internal states and output signal [GW93]. The reconstruction theorem is useful as an analysis tool, but, unfortunately, not very good as a prediction tool for complex systems. Models become easily unstable given a complicated state space or an arbitrary prediction horizon. Driven systems should be easier to predict than autonomous systems; however, the embedding dimension of a driven system is significantly larger since both input and output observables need to be adequately represented [Cas92]. The presence of noise in practically any real-world system further complicates the embedding job. Due to these problems, we end up with a fairly small number of examples where embedding has been applied successfully to predict a signal (sections 2.2, 5.3.1, and 10.1.4).

In addition, there are theoretical results that, in all generality, are concerned with the fundamental limits of predicting systems and signals. One of the most prominent of such results is the *halting problem*, formulated by Turing in the 1930's. It states that there is no way to tell from a given computer program's output whether the program will even-

tually come to a halt [Tur36]. Since a Turing machine can be implemented in the form of a dynamic system, there cannot be a universal algorithm that predicts the output of an arbitrary system. The prediction of the system's output would obviously include the prediction of its halt [WG93]. Clearly this insight has important implications, but, fortunately, does not interfere with our goals. We do not intend to predict arbitrary systems, merely systems that behave reasonably in a certain sense. A violin will not stop resonating as long as a competent violinist continues to play. Our intuition about the physical device helps us conclude that there are no hidden variables that would cause unexpected difficulties. Most systems in the natural world seem to be somewhat predictable in the sense that we do not expect *strange* behavior.

Regarding the estimation and approximation of a data set, the *Kramer-Rao* bound is a lower bound on the variance of any estimator used.[2] Regardless of how well we design an estimator, it will be limited by the uncertainty given by this bound [WWS96, Ger99a]. Although this information-theoretic result is an interesting limit, its relevance for the practical estimation of complex nonlinear systems is small. Typically, the real problem is to measure an optimal set of data rather than design the estimator. Unfortunately, the bound on the variance does not explain which variables to measure in the first place.

A third powerful concept is the *source entropy* of a dynamic system. It is defined as the asymptotic increase of information in a time series [CT91, Ger99a] and hence quantifies how much we can and cannot know about a system looking into the future. The source entropy is a troublesome quantity. The number of measurements needed for its estimation increases exponentially with the number of system dimensions. Even more limiting, entropy always assumes a specific representation of the data. As there is no model-independent measure of entropy, we need a precise idea about what the model should look like before we can compute its source entropy. The real problem therefore is to design a good model *a priori*.

There appears to be a gap between what theory claims to know and what is actually possible to model and predict in the real world. This thesis aims at bringing the theoretical limits and practical system characterization closer together. An algorithmic framework will be presented that unifies the prediction and characterization of arbitrary systems within a single coherent probabilistic approach. The goal is to create a modeling language that is general enough to describe the majority of nonlinear problems yet still specific enough to be useful in practical applications. We'll attack a variety of real-world data sets reusing techniques and theories that have been established, refined, and matured in past practice. We will overcome the limitations of these techniques by embedding them in a more general framework. In most applications, linear systems theory will be extended to nonlinear tools that keep the transparency of the original techniques but are not restricted by the assumption of linearity.

The main test application for our work is the design and implementation of a digital violin. We will show how to derive a predictive model of an acoustic instrument from observed data. A violin combines linear and nonlinear processing as well as probabilistic and deterministic dynamics; it is high-dimensional and difficult to play, yet controllable, as is proven every day in concert halls all over the world. The physical and logical im-

[2] under the condition that this estimator exists.

plementation of a digital instrument requires resources and knowledge that range from sophisticated mathematical algorithms to new sensing technology, and even a basic understanding of the art of violin making. Our ambition is grounded in multiple disciplines. From a physicist's point of view, we want to know if we can numerically emulate a system as complex as a violin. From an engineer's point of view, we would like to implement a digital instrument that operates in real time, sounds great, and is packaged nicely. From the musician's point of view, we would like to create an object that not only reproduces a given acoustic instrument, but also gives us access to its internal states which can be used to enhance its sonic space.

The creation of a digital copy of a violin serves as a paradigm for the philosophy of the *Things That Think* research consortium and the Media Lab in general. Namely, a traditional object, in this case a musical instrument, about which people have strong opinions, is digitally emulated and then functionally enhanced. In order for this new object to be accepted by the user and the audience, its historic, social, and cultural identities need to be respected and its original functionality needs to be preserved. In addition, a new balance between old and new technologies, between the beauty and efficiency of a violin and the power of today's computers, and between the intimacy of a chamber music concert and the richness of a multimedia event need to be found.

1.2 Outline

In the first part of this thesis, the algorithmic framework cluster-weighted modeling (CWM) is introduced. CWM is a probabilistic modeling tool for the characterization and prediction of systems of arbitrary dynamic character. The framework is based on density estimation around Gaussian kernels that contain simple local models describing the system dynamics of a data subspace. In one extreme case, the framework collapses into a simple model with linear coefficients, while in the opposite extreme, the model allows for embedding and forecasting data that may be non-Gaussian, discontinuous, and chaotic. In between these two extremes, CWM covers a multitude of models, each of which is characterized by a specific local input-output relationship and state representation. By embedding past practice and mature techniques in the general nonlinear framework, we create globally nonlinear models with transparent local structures.

The second part deals with the implementation details of CWM for a variety of problems. Guided by real-world data and applications, different local models are presented, pre and post-processing algorithms are discussed, and experimental results are given. Cluster-weighted classification is applied to the classification of credit card fraud data, bacterial measurements from electronic noses, and stimulated MEG brain data. Cluster-weighted estimation is applied to low-dimensional physical systems and data from a set of economic time series (section 5.3.2). Cluster-weighted spectral synthesis is applied to nonlinear microwave device characterization. Cluster-weighted linear predictive coding and cluster-weighted sampling are introduced as tools for digital sound synthesis.

In the third part, we explain how to build a digital violin. Data-driven musical synthesis is presented and compared to other synthesis methods such as physical modeling or global sampling. The modeling process can be divided into a series of steps. In the data-collection step, the synchronized audio and sensor data of a violinist playing the

sensor violin are recorded. In the data analysis step, low-level physical information, such as the short-term spectral decomposition and the instantaneous volume, and high-level information, such as the instantaneous pitch of the audio, are extracted from the data set. In the actual modeling step, the nonlinear mapping between input and output is approximated. This latter step involves finding the optimal data representation, the optimal feature vector, and a nonlinear search for the allocation of model parameters given the model architecture. In the synthesis step, signals are reconstructed given new off-line or real-time input from a violinist or cellist. A real-time system is presented which feels and sounds like a violin or cello, although it is in reality a mute sensor interface driving a computer model. The models are extended to cross-synthesis between different members of the violin family.

In the appendix, the custom sensing hardware and other supporting material are documented.

Part I

Cluster-Weighted Modeling

> Daß alle unsere Erkenntnis mit der Erfahrung anfange, daran ist gar kein Zweifel [...]. Wenn aber gleich alle unsere Erkenntnis mit der Erfahrung anhebt, so entspringt sie darum doch nicht eben alle aus der Erfahrung. Denn es könnte wohl sein, daß selbst unsere Erfahrungserkenntnis ein Zusammengesetztes aus dem sei was wir durch Eindrücke empfangen , und dem, was unser eigenes Erkenntnisvermögen [...] aus sich selbst hergibt [...].
> Man nennt solche *Erkenntnisse a priori*, und unterscheidet sie von den *empirischen*, die ihre Quellen *a posteriori*, nämlich in der Erfahrung, haben.
> I. Kant, *Kritik der reinen Vernunft*, B1,2[3]

The descriptive power and algorithmic beauty of graphical probabilistic networks is widely appreciated in the machine-learning community. The strong idea behind this methodology is the concept of conditional probabilities and Bayes' theorem. Immanuel Kant (1724-1804) never got to know Thomas Bayes (1702-1761), and he certainly didn't have Bayes' famous theorem in mind when thinking and writing about the origin of knowledge. However, the above quote from *Critique of Pure Reason* expresses a very similar idea in that it postulates that there needs to be some *a priori* intellectual skill or knowledge in order to acquire new knowledge from observation.

Bayesian networks follow the general scheme of Bayesian learning and knowledge representation. Unfortunately, the generality and flexibility of these networks are just about matched by their difficulty of use. Unless the architectures are constrained appropriately and are tailored for particular applications, they are of little use in a practical modeling situation. Gaussian mixture models, a subclass of graphical models, resolve some of these deficiencies. We present cluster-weighted modeling (CWM), a Gaussian mixture architecture that combines model flexibility with fast model design and ease of use.

CWM is also used in the context of nonlinear dynamic systems. For a long time, the field of nonlinear dynamics has been suffering from the lack of a unified theory, a common language, and a coherent mathematical toolkit. Prediction methods and characterization tools exist, but they are fragmented and rely on different assumptions. CWM unifies some of this methodology in a single framework.

[3] Experience is without doubt the first product that our understanding brings forth [...]. Nevertheless it is far from the only field to which our understanding can be restricted [...]. It tells us, to be sure, what is, but never that it must necessarily be thus and not otherwise.

Now such universal cognitions, which at the same time have the character of inner necessity, must be clear and certain for themselves, independently of experience; hence one calls them *a priori* cognitions: whereas that which is merely borrowed from experience is, as it is put, cognized only *a posteriori*, or empirically.

I. Kant. *Critique of Pure Reason.* Translated by P. Guyer and A.W. Wood.

Chapter 2

Background and related work

2.1 Graphical models

Graphical representations of probabilistic networks conveniently illustrate conditional dependence and independence in a network (fig. 2-1). Graphical models can be considered low-level descriptions of a network architecture since they provide a detailed decomposition of a complex network of nodes. At the same time, they are high-level descriptions of a network because the graphical representation abstracts from the parameterization of a specific distribution. Graphical Models provide the methodology to design new and complex architectures for complex problems with unusual data dependencies. On the other hand, they can be used to decompose and understand popular architectures that have been used for a long time, such as feed-forward neural networks, Markov models and mixture models. Moreover, graphical networks can be used at a meta-level to analyze and optimize training algorithms for these network architectures [Bun94].

Graphical models or their subclasses have many names, all of which point out a property or function. They are referred to as *independence networks* because the graphical representation describes dependence and independence among random variables. They are called *Bayesian belief networks* since dependencies between variables are expressed in terms of conditional probability functions that have implicit or explicit prior beliefs built into them. They are also called *influence diagrams* since causal dependencies between variables are illustrated. *Influence* is meant in the probabilistic sense, but deterministic causal dependence is included as a special case. As a meta-class of models, graphical models are conceptually unbounded. They unify existing network architectures, for example classical ANNs, in a single theory [Nea96], but, at the same time, they provide new insights and extensions to conventional networks and open up new application domains [HW95, Bun96, Jor98a].

A graphical network is defined at two levels: firstly, the *structure* of the network, consisting of a number of nodes, connections, and independence relations, indicates prior assumptions about the problem. Secondly, the set of *parameters* describes the marginal and conditional probabilities of the network structure. Assumptions of independence make the graph transparent, easy to understand and, most importantly, allow for parameters to be estimated from data. The more independences there are, the fewer parameters there are and the easier it is to estimate statistically valid parameters from data.

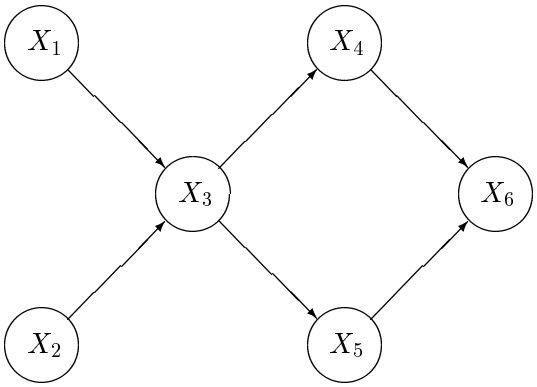


Figure 2-1: Directed acyclic graph [Jor98b].

Statistical machine learning deals with three fundamental problems related to a graphical model.

1. Find the optimal model structure for a problem or a set of observed data (*qualitative specification*).
2. Infer the optimal set of parameters, i.e. the correct distributions, given the structure and the set of observations (*quantitative specification*).
3. Given a structure and parameters, infer the probabilities of unobserved nodes given new observations.

(1) and (2) can be summarized as the *learning or estimation problem.* (3) happens at a later time and is referred to as the *inference* problem. It is important to define (1) in such a way that (2) to (3) are tractable and statistically valid. Often (3) is evaluated in the process of finding a solution for (1) and (2).

2.1.1 Notation and theory

Bayesian Networks and directed graphs

A probabilistic graph is specified in terms of a set of discrete or continuous random variables $S = \{X_1, X_2, ..., X_K\}$, represented in nodes, and in terms of conditional dependences between the variables, represented in edges between the nodes.

A Bayesian Network is a directed acyclic graph (DAG, fig. 2-1). The graphical representation indicates a conditional decomposition of the joint probability,

$$\begin{aligned} p(X_1, X_2, X_3, X_4, X_5, X_6 \mid M) \;=\; & p(X_1 \mid M)\; p(X_2 \mid M)\; p(X_3 \mid X_1, X_2, M) \\ & p(X_4 \mid X_3, M)\; p(X_5 \mid X_3, M)\; p(X_6 \mid X_4, X_5) \end{aligned} \tag{2.1}$$

where M is the available prior knowledge about the problem. The probability of each variable X_i is conditioned on its parents, where parents are all the variables that have a

directed edge connected to X_i. The general form of the expansion is

$$p(S \mid M) = \prod_{X \in S} p(X \mid \text{parents}(X), M) \quad . \tag{2.2}$$

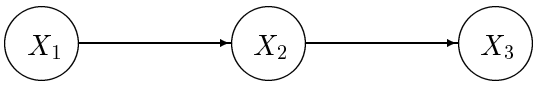
Different types of independence can be identified. Two random variables X_1 and X_2 are conditionally independent with respect to a third variable X_3 if

$$p(X_1, X_2 | X_3) = p(X_1 | X_3)\ p(X_2 | X_3) \quad . \tag{2.3}$$

Two random variables are marginally independent if

$$p(X_1, X_2) = p(X_1)\ p(X_2) \quad . \tag{2.4}$$

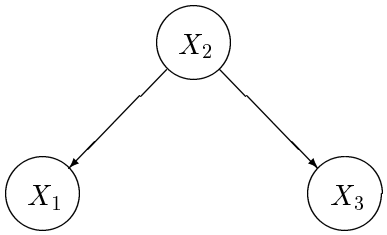
Neither of the two independence properties implies the other. In the configuration



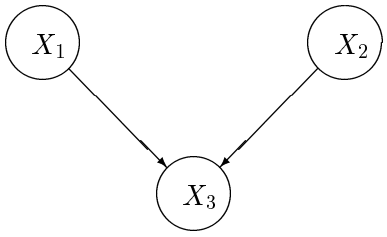
X_1 and X_3 are marginally dependent, but conditionally independent

$$\begin{aligned} p(X_1, X_3) &\neq p(X_1)\ p(X_3) \\ p(X_1, X_3 | X_2) &= p(X_1 | X_2)\ p(X_3 | X_2) \end{aligned} \tag{2.5}$$

The interpretation of this relationship is that when X_2 is known, any further information about X_1 does not change the distribution of X_3. Equ. 2.5 also holds true for the configuration



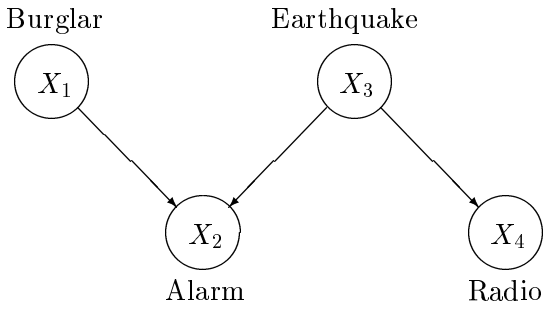
In this case X_1 and X_3 are said *d-separated* by X_2 [Pea87]. In



the situation is inverted. X_1 and X_2 are marginally independent, but conditionally dependent

$$\begin{aligned} p(X_1, X_2) &= p(X_1)\ p(X_2) \\ p(X_1, X_2|X_3) &= p(X_1|X_3)\ p(X_2|X_1, X_3) \neq p(X_1|X_3)\ p(X_2|X_3) \end{aligned} \tag{2.6}$$

X_3 is called a head-to-head node. Combining the two fundamental cases we get the famous *earthquake* paradox, which illustrates the surprising interdependency of nonadjacent random variables through a series of arcs:



If a house owner gets paged, because the house alarm system went off, he naturally assumes that a burglar had tried to enter the house. However, if at the same time the radio announces that there has been an earthquake in the area, the probability that a burglar caused the alarm to go off goes down by a lot.

$$\begin{aligned} p(X_1, X_3) &= p(X_1)\ p(X_3) \\ p(X_1, X_3|X_2, X_4) &= p(X_1|X_2, X_4)\ p(X_3|X_1, X_2, X_4) \\ &\neq p(X_1|X_2, X_4)\ p(X_3|X_2, X_4) \end{aligned} \tag{2.7}$$

Undirected graphs

Undirected graphs consist of nodes and undirected edges between them. Nodes are assumed independent from any non-adjacent node and hence fulfill the first order Markov condition. Therefore undirected graphs are also referred to as *Markov fields* (fig. 2-2). This class of networks is well-suited for applications in image processing. Every pixel is represented by a node and is assumed to depend on neighbor pixels only [Bun94].

In order to expand the probability distribution over the nodes, we summarize subsets of nodes in *cliques*. A clique is a maximal subset of nodes that are fully connected, i.e. a subset of fully connected nodes that are not completely part of another fully connected subset. An example of a clique in fig. 2-2 is $C = \{X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2}\}$. The graph is interpreted in terms of the functions (potentials) over the cliques,

$$p(X) = \prod_{C_i} f_{C_i}(C_i) \tag{2.8}$$

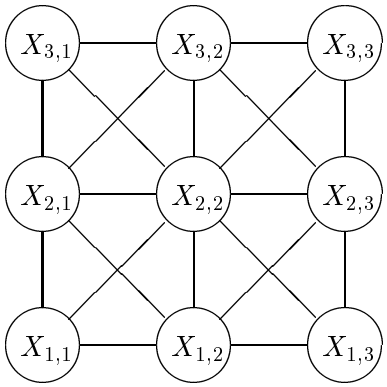


Figure 2-2: Markov field, UGM, [Bun94, P.164].

In our example (fig.2-2) the joint distribution decomposes into

$$\begin{aligned} p(S) &= f_1(X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2}) \cdot f_2(X_{1,2}, X_{1,3}, X_{2,2}, X_{2,3}) \cdot \\ & \quad f_1(X_{2,1}, X_{2,2}, X_{3,1}, X_{3,2}) \cdot f_2(X_{2,2}, X_{2,3}, X_{3,2}, X_{3,3}) \quad . \end{aligned} \tag{2.9}$$

From this representation we see that X_i is independent from all the variables with which it doesn't share a clique, i.e. all the variables that aren't an argument of the f_j of which X_i is an argument. The family of global distributions $p(S)$ and the family of product distributions $\prod_i f_i$ is identical.

2.1.2 Inference

The nodes of a probabilistic graph can be classified in visible units (observed variables V) and hidden units (hidden variables H). The visible units are represented by observed instantiated variables. They are also called *evidence nodes*. Inference is the operation that finds the conditional distribution of the hidden units given the evidence units [Jor98b, JGJS99],

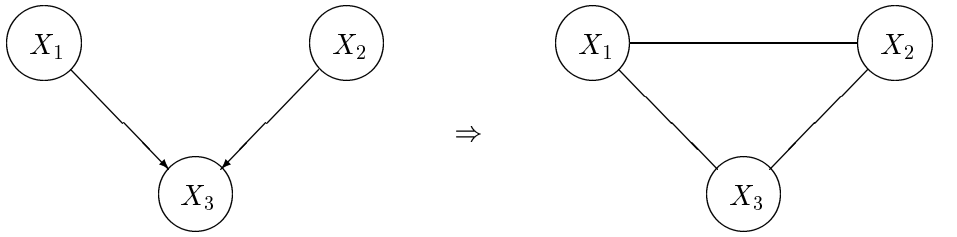
$$p(H|V) = \frac{p(H,V)}{\sum_H p(H,V)} \quad . \tag{2.10}$$

A general methodology for inference in directed and undirected graphs is the JLO algorithm, named after its inventors Jensen, Lauritzen and Olesen [JLO90, LS88]. The approach consists of three steps: moralization, triangulation and propagation. Moralization and triangulation are summarized in the construction step, in which a *junction tree* is constructed. The construction happens off-line and only once. In the propagation step, the inference problem is solved, propagating new evidence, that is observed data, through the network.

The JLO algorithm can be used to estimate arbitrarily complex graph structures. It contains well known inference algorithms such as the forward-backward algorithm for hidden Markov models or the Expectation-Maximization algorithm for Gaussian mixture models as special cases.

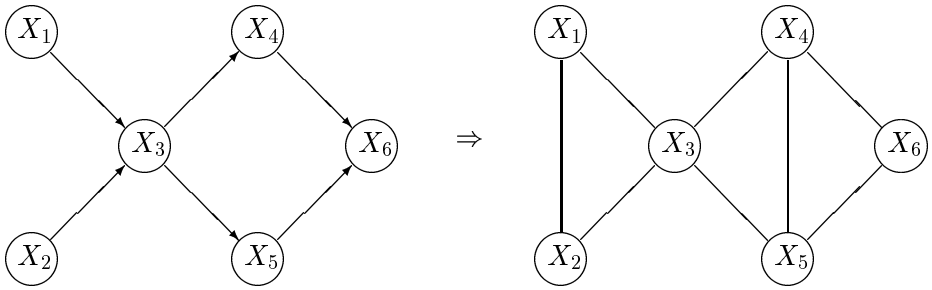
Moral graphs

We first convert the directed graph into an undirected graph. Since both directed and undirected graphs expand the global distribution in product form we can transform conditional probabilities into local potentials. In directed graphs a node and its parents are usually not fully connected. We marry the parents of each node to obtain a *moral graph* [Jor98b]:



$P(X_3|X_1, X_2)$ $\qquad$ $f(X_1, X_2, X_3) = P(X_3|X_1, X_2)$

For example, moralizing the Baysian Network from fig. 2-1 yields



and a global distribution that expands into

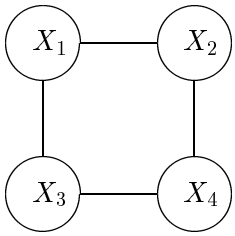
$$\begin{aligned} P(X_1, X_2, X_3, X_4, X_5, X_6) &= P(X_1)\ P(X_2)\ P(X_3|X_1, X_2)\ P(X_4|X_3) \\ &\quad P(X_5|X_3)\ P(X_6|X_4, X_5) \\ &= f_1(X_1, X_2, X_3)\ f_2(X_3, X_4, X_5)\ f_3(X_4, X_5, X_6) \quad , \end{aligned} \tag{2.11}$$

with

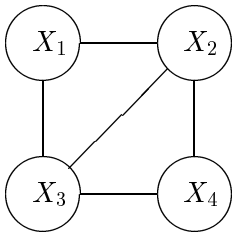
$$\begin{aligned} f_1(X_1, X_2, X_3) &= P(X_1)\ P(X_2)\ P(X_3|X_1, X_2) \\ f_2(X_3, X_4, X_5) &= P(X_4|X_3)\ P(X_5|X_3) \\ f_3(X_4, X_5, X_6) &= P(X_6|X_4, X_5) \quad . \end{aligned} \tag{2.12}$$

Triangulation

The triangulation step is needed to guarantee local and global consistency of a clique tree. A triangulated graph has no cycles of four or more nodes without a chord, where a chord is an edge between two non-neighboring nodes. We triangulate a tree by adding chords where necessary. For example,



is triangulated into



A triangulated clique tree has the so-called *running intersection property*: any node that is part of two cliques is also represented in any clique in between. The clique tree of a triangulated graph is called a *junction tree*. The edges joining two cliques are called separators.

Unlike moralization, triangulation of a network is not unique. Given a moral graph there are usually many different ways of turning it into a junction tree. In the limit, we can always fully connect all the nodes and obtain a perfectly valid triangulation of the tree. However, in the interest of efficient and valid model estimation, we want cliques to be as small as possible. The search for the optimal junction tree is NP-complete, yet, there are usually many acceptable solutions and good heuristics to find them.

Propagation

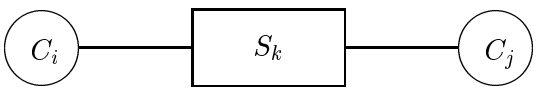
Given new evidence, distributions and potentials are marginalized and rescaled with respect to the observed data. This can be tricky when nodes are part of more than one clique since we need to preserve consistency among cliques. Our starting point is the representation of the joint probability in terms of a product of potential functions on the cliques,

$$p(U) = \prod_{C \in U} f_C(X_C) \quad . \tag{2.13}$$

We can extend this representation to include functions on the edges between cliques and obtain the *generalized potential representation*,

$$p(U) = \frac{\prod_C a_C(X_C)}{\prod_S b_S(X_S)} \quad . \tag{2.14}$$

The additional term $b_s(X_s)$ is used for message passing between cliques. Given the evidence X_k, the adjacent cliques C_i and C_j and the separator S_k in between,



we define

$$\begin{aligned} b^*_{S_k}(x_{S_k}) &= \sum_{C_i \setminus S_k} a_{C_i}(x_{C_i}) \\ a^*_{C_j}(x_{C_j}) &= a_{C_j}(x_{C_j})\lambda_{S_k}(x_{S_k}) \\ \lambda_{S_k}(x_{S_k}) &= \frac{b^*_{S_k}(x_{S_k})}{b_{S_k}(x_{S_k})} \end{aligned} \tag{2.15}$$

The full clique tree is updated in a *collection* and a *distribution* step. In the collection step the messages are passed to a root clique. In the distribution step they are sent back to the cliques along the same edges [JLO90, Oli00].

The complexity of the JLO algorithm is exponential in the size of the cliques. Fortunately cliques are small for most of the well-known architectures, such as HMM and Mixture models. Recent research has yielded approximate inference algorithms to estimate architectures that were intractable before [SJJ96, SJ96, SJ96, GJ96, JGJS99].

2.1.3 Example I: Gaussian mixture models

Mixture models, also called mixture of experts models, consist of K *expert systems* and an integrating network, sometimes referred to as the *gating network* (fig. 2-3) [Hay99]. Mixture models are usually used in supervised learning problems where different domains of the feature space have different output characteristics. Depending on the problem, a large number of simple experts, e.g. linear functions, may be used [JJ94], or alternatively, a few complex experts, e.g. Neural networks [WMS95]. Mixture models are closely related to *radial basis function* architectures [GJP95] and *locally weighted polynomial regression* [MSD97].

Gaussian mixture models implement the gating function in the form of a posterior likelihood of expert K with respect to a data point.

$$g_k(\mathbf{x}) = \frac{C_k e^{\mu(\mathbf{x})}}{\sum_{l1}^{K} C_l e^{\mu(\mathbf{x})}} \tag{2.16}$$

We consider a mixture distribution of K normal distributions $\mathcal{N}(m_k, \sigma_k^2)$ over the multivariate random variable x and the random variable z. z depends on x and takes on the values $\{1, 2, ..., K\}$. The joint distribution of (x, z) is

$$p(x, z) = \sum_{k=1}^{K} \delta_k(z) p_k \frac{1}{\sqrt{2\pi}} \sigma_k \exp\left\{ -\frac{(x - m_k)^2}{2\sigma_k^2} \right\} \tag{2.17}$$

The term $\delta_k(z)$ makes sure that only one of the distributions in the sum is turned on at a time. However, usually z is hidden yielding the marginal distribution of x

$$p(x, z) = \sum_{k=1}^{K} \frac{p_k}{\sqrt{2\pi}} \sigma_k \exp\left\{ -\frac{(x - m_k)^2}{2\sigma_k^2} \right\} \tag{2.18}$$

which is called the normal mixture distribution.

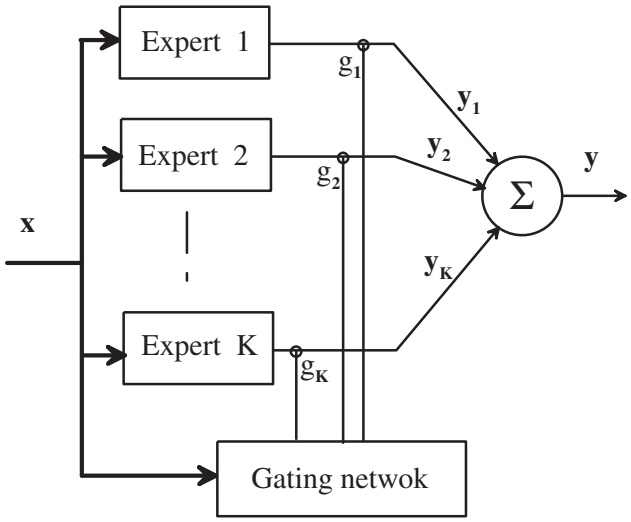
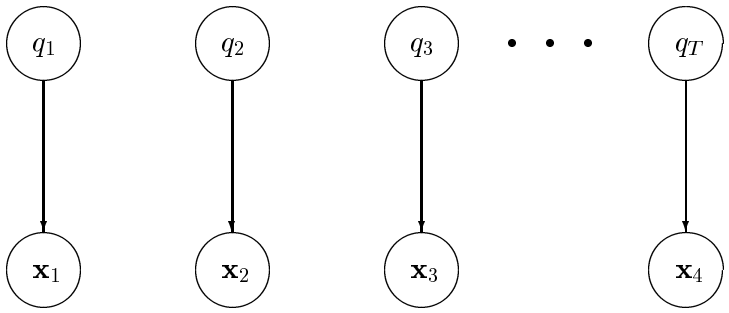


Figure 2-3: Gated (mixture) experts showing experts 1 through K as well as a gating network [Hay99, p.368].

The graphical representation of a Gaussian mixture is as simple as



The model estimation step is concerned with identifying the parameters p_k, σ_k and m_k. We first observe that the estimation of model parameters from data is easy, when all the states, including z, are observed. In this case we simply compute the relevant moments for kernel k with respect to the data $(\mathbf{x}, z|z = z_k)$,

$$\begin{aligned} \mathbf{m}_k &= \frac{1}{N_k} \sum_{\{\mathbf{x},z|z=z_k\}} \mathbf{x} \\ \sigma_k &= \frac{1}{N_k} \sum_{\{\mathbf{x},z|z=z_k\}} (\mathbf{x} - m_k)^2 \\ p_k &= \frac{N_k}{N} \end{aligned} \tag{2.19}$$

where N is the number of data points and N_k is the number of points for which $z = z_k$.

Since the z are usually unknown, we need to estimate the model parameters based on incomplete data. The implication of this is that we update the parameters in a soft rather than hard way. The counting procedure is replaced by an update that weights the contribution of all data points given a particular kernel. Secondly the one step ML estimation is replaced by an iterative search, the Expectation-Maximization (EM) algorithm. The EM-algorithm uses two distinct steps to optimize the likelihood of the data given the model.

1. In the E-step the model is assumed correct and the data distribution is reevaluated.
2. In the M-step the data distribution of hidden and observed states is assumed correct and the model parameters are updated, so that they maximize the likelihood of the data given the model.

Section 3.2 and Appendix A discuss the EM-algorithm in depth. Mixture models can become arbitrarily complex using hierarchical layers of gating [JJ94].

2.1.4 Example II: Hidden Markov models

A hidden Markov model (HMM) is a graphical mixture model with dynamics. HMMs model observables of a time series $\mathbf{x}_t$ assuming that the observables are dependent on internal states q_t (fig.2-4). The hidden states obey the first-order Markov property, i.e. q_τ is independent from any state q_t except $q_{\tau-1}$ and $q_{\tau+1}$. The observations $\mathbf{x}_\tau$ are conditionally independent from the states $q_{\tau-1}$ and $q_{\tau+1}$, given the state q_τ.

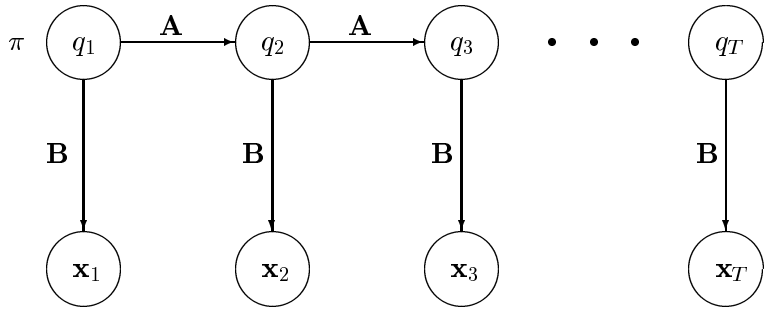


Figure 2-4: Hidden Markov model in graphical model notation. π is the initial state probability, $\mathbf{A}$ is the matrix of transition probabilities and $\mathbf{B}$ is the matrix of emission probabilities.

An HMM is usually characterized by the number of hidden states J, the number of time steps T, where T can be infinite, the initial state probabilities π, the transition probability matrix $\mathbf{A}$, and the matrix of emission probabilities $\mathbf{B}$.

$$\begin{aligned} \pi(q_i) &= p(q_i) \\ [\mathbf{A}]_{j,i} &= p(q_{j,t+1}|q_{i,t}) \\ [\mathbf{B}]_{j,i} &= p(\mathbf{x}_{j,t}|q_{i,t}) \end{aligned} \tag{2.20}$$

In this review we consider discrete emission probabilities and observations of discrete symbols $\mathbf{x}_t$. Section 3.5 we will introduce a continuous-valued HMM. We assume that there is some underlying state sequence that emits the observed variables and we assume the system is stationary, i.e. $\mathbf{A}$ and $\mathbf{B}$ are independent of time.

Given this general architecture three key problems can be identified [RJ86]:

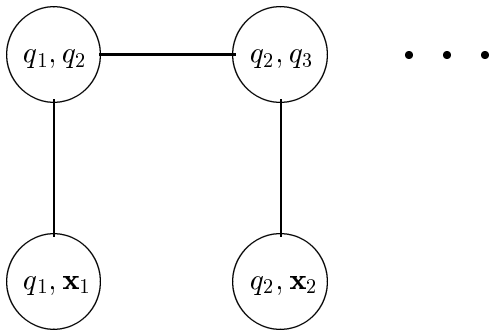
1. Given and observation sequence $O = \{X_1, X_2, ..., X_T\}$ and a model $M = (\pi, \mathbf{A}, \mathbf{B})$, how can we compute $p(O)$, the probability of the observation sequence?

2. Given the observation sequence O, which is the most likely underlying state sequence?

3. How do we find the best model $M = (\pi, \mathbf{A}, \mathbf{B})$ given a set of observations, namely the model M that maximizes $P(O|M)$?

Problem (3) may seem to have the highest priority, but we'll find that we need the solutions to problems (1) and (2), in order to solve (3).

Following the methodology from section 2.1.2, we first triangulate the graph in fig. 2-4:



The size of each clique is two, which means that the total complexity of the algorithm is $O(J^2T)$ and that a HMM architecture is one of the rare examples where exact inference is possible.

Message passing in the case of an HMM translates into the forward-backward algorithm, the key element of the Baum-Welsh algorithm. The Baum-Welsh algorithm in turn is a special case of EM. Similarly to mixture models, we observe that the estimation of the model parameters as well as the prediction of future $\mathbf{x}_t$ would be simple if only we knew the state sequence q_t. However, since the states are hidden we need to find more sophisticated strategies. We first assume that all the states, including the internal state sequence are observed, in which case we simply count observed events. The number of transitions $N_{j,i}$ from q_i into q_j corresponds to the transition probabilities

$$[\mathbf{A}]_{j,i} = \frac{N_{j,i}}{\sum_{k=1}^{K} N_{k,i}} \tag{2.21}$$

Likewise, we count the number of events $M_{j,i}$ that state q_i maps into an observation $\mathbf{x}_j$, to estimate the emission probabilities,

$$[\mathbf{B}]_{j,i} = \frac{M_{j,i}}{\sum_{k=1}^{L} M_{k,i}} \tag{2.22}$$

where L is the number of possible observations. We also evaluate the π_i by simply counting the number of observation sequences that start in state q_i and by renormalizing.

When the hidden states are unknown, the algorithm needs to be extended in two directions.

1. Events are not simply counted but are weighted by the probability that the event actually occurred. For example, the estimator for $\mathbf{A}$ is altered to

$$[\mathbf{A}]_{j,i} = \frac{\sum_{t=1}^{T} p(q_{i,t})\ p(q_{j,t+1} \mid q_{i,t})}{\sum_{t=1}^{T} \sum_{j=1}^{K} p(q_{i,t})\ p(q_{j,t+1} \mid q_{i,t})} \tag{2.23}$$

2. The search for the best model iterates between the estimation and a maximization step. Given some initial conditions, we assume the model M to be correct and evaluate the probabilities of the observations given the model (E-step). This is done using the junction-tree (forward-backward) algorithm (section 3.5). We then assume this new distribution to be correct and reestimate the model parameters $M = \{\pi, \mathbf{A}, \mathbf{B}\}$. Given the new and improved model we go back to step () and keep iterating (section 3.5).

By now we have indicated solutions to our set of problems. The solution to problem (1) consists of message passing in the junction tree given the evidence of an observation sequence. Since every single state sequence has a non-zero probability, we sum over all of them to evaluate probability of the observation. To solve to problem (2) we evaluate the probability of every single state q_t using the forward-backward procedure, we. We also evaluate the probability of any sequence of states relative to all the other sequences. We can then factor out the most likely individual state or the most likely sequence of states. The result depends on the time horizon of the prediction and on whether the prediction is real-time and causal or off-line and non-causal. The model estimation problem (3) is solved with the Expectation-Maximization algorithm [RJ86].

Extensions of HMMs include *coupled hidden Markov models* [BNP, Oli00] for modeling coupled processes, *parameterized hidden Markov models* [WB98, WB99] for parameterizing a process, and *factorial hidden Markov models* [GJ96] for estimating models with many states.

2.2 Nonlinear dynamic systems

In the beginning of this work there has been the realization that nonlinear systems theory provides tools that make it possible to fully estimate a dynamic system from observation. The enabling theoretical insight is the *embedding theorem*, which will be briefly reviewed along with some important system-characterization methodology. Reconstruction of the physical state space is explicitly or implicitly fundamental to the prediction and characterization applications in the succeeding chapters.

2.2.1 State space reconstruction

We first consider dynamic systems of the form

$$\begin{aligned} \frac{\partial \mathbf{x}}{\partial t} &= f(\mathbf{x}, \mathbf{x}_0) \\ \mathbf{y}(t) &= g(\mathbf{x}(t)) \quad . \end{aligned} \tag{2.24}$$

Equation 2.24 describes a system that runs freely given the initial conditions $\mathbf{x}_0$ and the internal governing equations f. It is fully characterized by its manifold of dynamic states encoded in the function f. Unfortunately, f is unknown for most real-world systems. With the formulation of the embedding theorem, Floris Takens presented a methodology to recover an equivalent description of f from observation of the system:

> *Theorem 1. Let M be a compact manifold of dimension m. For pairs (ϕ, y), $\phi : M \to M$ a smooth diffeomorphism and $y : M \to \mathcal{R}$ a smooth function , it is a generic property that the map $\Phi_{(\phi,y)} : M \to \mathcal{R}^{2m+1}$, defined by*
>
> $$\Phi_{(\phi,y)}(x) = (y(x), y(\phi(x)), ..., y(\phi^{2m}(x))) \tag{2.25}$$
>
> *is an embedding; by "smooth" we mean at least C^2.*[Tak81]

Consider that x is a state variable of a dynamic system, $y(x)$ is an observable of the system and M is the manifold of system states. If we furthermore specify that x is a function of time and that $\phi(x)$ is a delay operator, the enigmatic theorem 2.25 starts to make sense. The theorem states that there is a diffeomorphic mapping between the manifold $X \in \mathcal{R}^m$ of a dynamic system and a second manifold $Y \in \mathcal{R}^n$ which is reconstructed from an observable of the system. Given the output measurement $y(t)$ of the system, Y is reconstructed by using time delayed values $y(t - \tau)$ as the axes of the embedding space. The axes of Y are $y(t), y(t-\tau), y(t-2\tau), ..., y(t-(n-1)\tau)$, where τ is some constant. The new manifold Y is topologically invariant to X.

Fig. 2-5 illustrates this beautifully. The set of equations known as the Lorenz

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy \end{aligned} \tag{2.26}$$

set is one of the rare systems that are low dimensional and chaotic. Fig. 2-5a illustrates the time domain signal of the set. The flat power spectrum (fig. 2-5b) seemingly indicates that the system is random, however, the state space representation in fig. 2-5c reveals that there is structure in the data. Surprisingly, the reconstructed state space in fig. 2-5d gives almost precisely the same picture as the true state space in fig. 2-5c. Not only is the overall topological shape preserved, but also local details of the orbit are easily identified in the two plots. We will revisit the Lorenz attractor in sections 3.3 and 5.3.1.

For now we summarize the most important phenomenological implications of the embedding theorem:

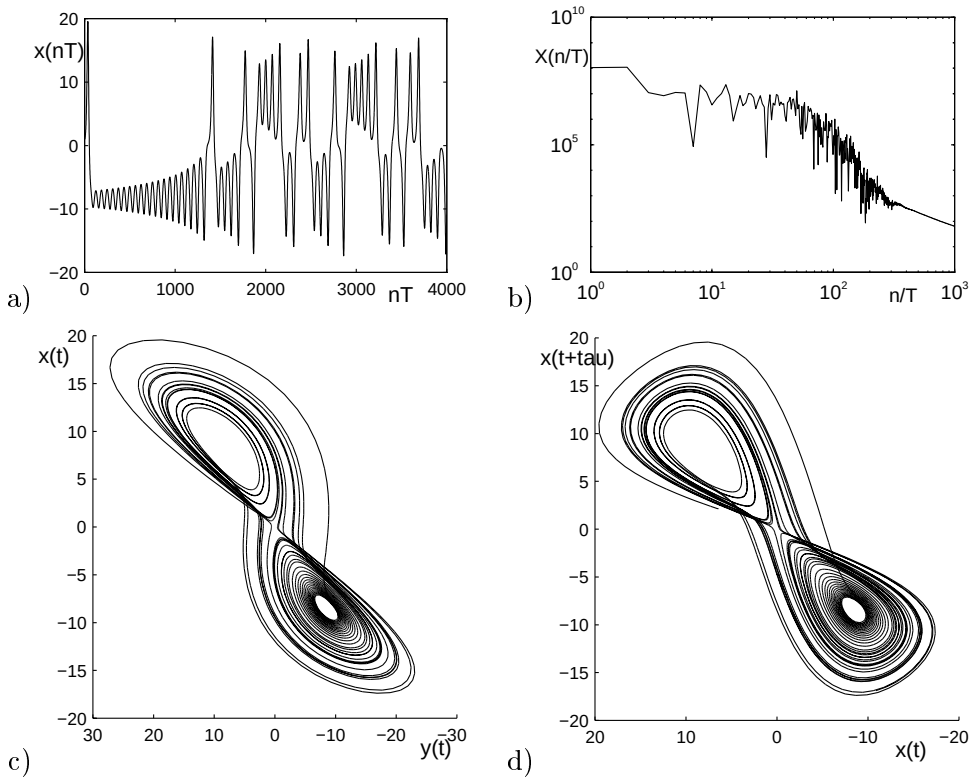


Figure 2-5: x-coordinate of the Lorenz set in (a) time and (b) frequency domain. (c) Lorenz attractor in the x-y-z state space projected in 2D. (d) 3D time-lag space of $x(t)$ projected in 2D ($\tau = 10$ samples).

1. The manifold formed by a dynamic system's Hamiltonian states is topologically identical to a second manifold formed by one single observable of the system and its $2m$ time lags. The two vector manifolds differ by no more than a smooth and invertible local change of coordinates. They therefore define a local diffeomorphism.

2. The above is true only if the dimension of the lag space, referred to as the embedding dimension, is large enough. In general, the embedding dimension is $2m+1$, where m is the dimensionality of the original manifold. The choice of $2m+1$ assures that the solution does not lie on a bad (ambiguous) subset in the embedding space.

3. The embedding theorem is true generically, i.e. there are only very few unlikely cases for which it doesn't work. In those pathological cases, a small change of variables fixes the problem.

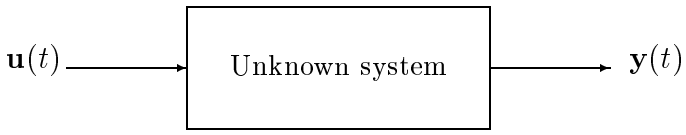
4. Due to the robustness of the theorem, the choice of τ is arbitrary. However, if τ is too small the manifold degenerates into a single line. If τ is too big, the locality of the dynamics gets lost.

2.2.2 Input-output embedding

Most of the systems in this world are not autonomous but depend on time variant input $\mathbf{u}(t)$. Those systems are formally described as

$$\begin{aligned} \frac{\partial \mathbf{x}}{\partial t} &= f(\mathbf{x}, \mathbf{u}(t), \mathbf{x}_0) \\ \mathbf{y}(t) &= g(\mathbf{x}(t)) \quad . \end{aligned} \tag{2.27}$$

Since we typically don't have access to $\mathbf{x}$, the block diagram resembles like a black-box.



The theory of embedding has been extended to cover this more general type of a dynamic system. Given a time-dependent scalar input $u(t)$ and an output signal $y(t)$ of a system, Casdagli and others proposed the following embedding [Cas92, Hun92]:

$$\begin{aligned} y(t) &= f(y(t-\tau), y(t-2\tau), ..., y(t-(d-1)\tau), u(t), \\ &\quad u(t-\tau), u(t-2\tau), ..., u(t-(d-1)\tau) \end{aligned} \tag{2.28}$$

Casdagli shows that there is a diffeomorphic mapping between the reconstructed manifold 2.29 and the state space of the driven system. Given an m-dimensional state vector, Casdagli claims that $2m+1$-dimensional time-lag vectors for both the input time series $u(t)$ and the output time series $y(t)$ results in a unique and singular-valued function $f()$.

The time-discrete version of expression 2.29 is

$$\begin{aligned} y(n) &= f(y(n-k), y(n-2k), ..., y(n-(m+1)k), \\ &\quad u(n), u(n-k), u(n-2k), ..., u(n-(l+1)k) \end{aligned} \tag{2.29}$$

We use this representation to sketch the proof of the input-output embedding theorem as developed in [Cas92]. Let's assume a finite-dimensional unknown system characterized by its d-dimensional state vector $\mathbf{x}_n$ and the following set of equations:

$$\begin{aligned} \mathbf{x}(n+1) &= f(\mathbf{x}(n), \mathbf{u}(n)) \\ y(n+1) &= h(\mathbf{x}(n+1)) \end{aligned} \tag{2.30}$$

If equation 2.29 is true, the lagged vector

$$\mathbf{v}_n = \left(y_n, y_{n-k}, y_{n-2k}, ..., y_{n-(m-1)k}, u_n, u_{n-k}, u_{n-2k}, ..., u_{n-(l-1)k}\right)$$

is needed to uniquely and smoothly describe the unknown state $x(n)$. Therefore there has to be a function $P : \mathcal{R}^{m+l} \to \mathcal{R}$ for all y_n such that

$$y(n+1) = P(v(n)) \quad . \tag{2.31}$$

We assume $m = l$ and define the smooth map $\Phi : \mathcal{R}^d \times \mathcal{R}^{m-1} \rightarrow \mathcal{R}^m$ by

$$\begin{aligned} \Phi(\mathbf{x}, u_{n-1}, ..., u_{n-(l-1)k}) &= ..., h(f(f(\mathbf{x}, u_{n-(l-2)k}, u_{n-(l-1)k}))), \\ & \quad h(f(\mathbf{x}, u_{n-(l-1)k}), h(\mathbf{x}) \\ &= y_n, y_{n-k} ... y_{n-(m-1)k} \quad . \end{aligned} \tag{2.32}$$

For expression 2.32 to have a unique solution $\mathbf{s}$, m must be chosen such that $m > d$. For $m = d$ there would be d simultaneous nonlinear equations for d unknown components of $\mathbf{x}$, which usually have more than one solution. If m is increased by one, one more equation is obtained which reduces the set of solutions to a unique solution for almost all cases. Yet, if we are unlucky, the solution still lies on a bad subset Σ^{d-1} of $\mathcal{R}^d$ of dimension $d-1$. Increasing m, we are reducing the subset dimension until finally for $m > 2d$, the bad subsets disappear. We ignore remaining bad subsets which generically have measure 0 and which do not disappear with a further increase of m.

Having shown that there is a solution for $\mathbf{s}$ for all n, we also proved that there is a smooth function P, as in expression 2.31. P is found by plugging $\mathbf{s}$ into 2.30 and iterating. To summarize: there is a smooth and unique function P for generically all input sequences given that $m, l > 2d$. For $m, l = d+1$ such a function P exists almost everywhere in the solution space [Cas92].

2.2.3 Characterization of nonlinear dynamic systems

Characterization tools for nonlinear dynamic systems find invariants of the system f (equ. 2.24 and 2.29). Useful characteristics are dimensionality and predictability estimates, or general statistical properties of the system. The tools that are reviewed here usually assume a reconstructed state space to start with.

The **correlation dimension** in general finds the dimensionality of a sub-manifold M in a space S. Given a data manifold in a reconstructed state space (equ. 2.29), the correlation integral $C(D, N, r)$ is defined as

$$C(D, N, r) = \frac{1}{N} \sum_{i=1}^{N} B_i(\mathbf{x}_i, D, r) \quad , \tag{2.33}$$

where D is the dimension of S, N is the number of data points, r is a radius around point $\mathbf{x}_i$. $B_i()$ is the proportion of points that falls inside the hyper-sphere around point $\mathbf{x}_i$. The correlation dimension ν is defined as the asymptotic scaling of $C(D, N.r) \propto r^{\nu}$. ν is strictly smaller or equal to D. It increases with D until D exceeds the dimensionality of M, at which point ν equals that dimensionality. For chaotic systems ν can be non-integer-valued.

Lyapunov Exponents quantify the rate of divergence of nearby trajectories. They measure the deformation of an infinitesimal sphere over time. Assuming a hyper sphere of radius $r_i(0)$ in a reconstructed state space of Dimension D, this sphere is being deformed with time into an ellipsoid with the principal axis $r_i(t), i = 1, 2, ..., D$. The Lyapunov coefficients λ_i are then defined as

$$\lambda_i = \lim_{t \rightarrow \infty} \lim_{r_i(0) \rightarrow 0} \frac{1}{t} \log \frac{r_i(t)}{r_i(0)} \quad . \tag{2.34}$$

where the λ_i are usually ordered as $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D$. The set $\{\lambda_i\}_{i=1}^{D}$ is called the Lyapunov Spectrum and $\sum_{i=1}^{D} \lambda_i$ quantifies the volume expansion rate. Systems with one or more positive Lyapunov coefficients have chaotic behavior. The Lyapunov spectrum can be related to the system dimension by the Kaplan-Yorke conjecture [Ger89].

Entropy measures the rate at which a system generates new information and hence indicates the predictability of its output. Let $x(t)$ be the output of the system. The scalar entropy of a time series equals

$$H_1(x_t) = -\sum_{N} p(x_t) \cdot \log_2 p(x_t) \quad , \tag{2.35}$$

and the entropy of a joint distribution in a two-dimensional time-lag space is defined as

$$H_2(\tau) = -\sum_{N_t} \sum_{N_{t-\tau}} p(x_t, x_{t-\tau}) \cdot \log_2 p(x_t, x_{t-\tau}) \quad . \tag{2.36}$$

Consequently, the block entropy for a D-dimensional lag-space vector is

$$H_D(\tau) = -\sum_{N_t} \sum_{N_{t-\tau}} ... \sum_{N_{t-(D-1)\tau}} p_D \cdot \log_2 p_D \quad , \tag{2.37}$$

where

$$p_D = p(x_t, x_{t-\tau}, ..., x_{t-(D-1)\tau}) \quad . \tag{2.38}$$

We can evaluate the redundancy between two-block entropies of different order D and $D+1$ as

$$R_D(\tau) = H_1(\tau) + H_{D-1}(\tau) - H_D(\tau) \quad . \tag{2.39}$$

R_D equals the scalar entropy of the signal $x_{t-(D-1)\tau}$, plus the joint entropy H_{D-1}, minus the joint entropy H_D. There are two limit cases to this expression. If D is a lot smaller than the embedding dimension, then the next sample is undetermined by the previous $D-1$ samples and the redundancy equals zero. On the other hand, if D is large, the new sample doesn't provide any new insight and the redundancy equals the entropy H_1. The embedding dimension equals the minimum D for which $R_D = H_1$.

Alternatively, we define the source entropy

$$h(\tau) = \lim_{d \to \infty} H_D(\tau) - H_{D-1}(\tau) \quad . \tag{2.40}$$

The dimension then equals the minimal D for which $h_D(\tau) = h_\infty(\tau)$.

Chapter 3

The cluster-weighted modeling framework

3.1 Model architecture

Cluster-weighted modeling (CWM) is a framework for supervised learning based on probability density estimation of a joint set of input feature and output target data. It is similar to mixture-of-experts type architectures [JJ94] and can be interpreted as a flexible and transparent technique to approximate an arbitrary function. However, its usage goes beyond the function fitting aspect, since the framework is designed to include local models that allow for the integrations of arbitrary modeling techniques within the global architecture. This section introduces the common elements of the framework.

Nonlinear function fitting can be classified into two major categories. The first category are linear coefficient models (generalized linear models) which use a sum over arbitrary nonlinear basis functions $f_k(\mathbf{x})$ weighted by linear coefficients a_k,

$$y(\mathbf{x}) = \sum_{k=1}^{K} a_k \; f_k(\mathbf{x}) \quad . \tag{3.1}$$

A prominent example of this architecture is the class of polynomial models (3.14). The optimal set of parameters $\mathbf{a}$[1] for a linear coefficient models (3.1) can be found with a single matrix inversion. However, it has been shown that the required number of basis terms in 3.1 increases exponentially with the dimensionality of $\mathbf{x}$ for a given error.

The second category of nonlinear models uses variable coefficients inside the nonlinear basis functions

$$y(\mathbf{x}) = \sum_{k=1}^{K} f(\mathbf{x}, \mathbf{a}_k) \quad . \tag{3.2}$$

Typical examples for type 3.2 are artificial neural networks. Nonlinear coefficient models are exponentially more powerful, so they can reduce the number of basis terms to be linear in the dimensionality of $\mathbf{x}$ [Bar93]. However, training of the coefficients $\mathbf{a}$ requires

[1] in a mean square sense.

an iterative search that can be tedious and time consuming. Moreover, one can never be sure that the trained coefficients are optimal, since search algorithms only converge to local minima of the cost function. These problems with fully nonlinear models are commonly summarized as the *curse of dimensionality* [WG93, Ger99a].

CWM uses the concept of fully nonlinear models as in (3.2) to build globally powerful nonlinear models. At the same time it describes local data features with simple models that satisfy expression (3.1). Hence CWM combines the efficient estimation of the latter approach (3.1) with the expressive power of the former approach (3.2, section 3.2).

We start with a set of discrete or real valued input features $\mathbf{x}$ and corresponding discrete or real valued target vectors $\mathbf{y}$. $\mathbf{x}$ consists of measured sensor data, processed features or discrete classifiers. It is composed of independent observations or of time-delayed values of an embedded time series. $\mathbf{y}$ may be the scalar valued sample of a time series, an independent target feature or a classifying label. We consider the joint input-output set $\{\mathbf{y}_n, \mathbf{x}_n\}_{n=1}^N$, and it is our goal to infer the joint density $p(\mathbf{y}, \mathbf{x})$, which, given unlimited data, is the most general, compact, and statistically sufficient description of the data set. We parameterize $p(\mathbf{x}, \mathbf{y})$ as a Gaussian mixture density which allows us to derive conditional probabilities from the estimated density.

$p(\mathbf{x}, \mathbf{y})$ is expanded in a sum over clusters c_k. Each cluster contains an input distribution, a local model, and an output distribution. The input distribution is parameterized as an unconditioned Gaussian and defines the domain of influence of a cluster.

$$\begin{aligned} p(\mathbf{y}, \mathbf{x}) &= \sum_{k=1}^{K} p(\mathbf{y}, \mathbf{x}, c_k) \\ &= \sum_{k=1}^{K} p(\mathbf{y}, \mathbf{x}|c_k)\ p(c_k) \\ &= \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}, c_k)\ p(\mathbf{x}|c_k)\ p(c_k) \quad . \end{aligned} \tag{3.3}$$

The sum over clusters is normalized by the unconditional cluster probabilities $p(c_k)$ to satisify $\sum_{k=1}^{K} p(c_k) = 1$. If there is unordered, discrete-valued feature data $\mathbf{x}_d$ (labels) in addition to real valued input data $\mathbf{x}_r$, the cluster probability $p(c_k)$ is conditioned on the state and hence is replaced by $p(c_k|\mathbf{x}_d)$.

$$\begin{aligned} p(\mathbf{y}, \mathbf{x}) &= p(\mathbf{y}, \mathbf{x}_r, \mathbf{x}_d) \\ &= \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}_r, \mathbf{x}_d, c_k)\ p(\mathbf{x}_r|\mathbf{x}_d, c_k)\ p(c_k|\mathbf{x}_d) \quad . \end{aligned} \tag{3.4}$$

We assume the components of $\mathbf{x}_d$ statistically independent and set $\sum_{k=1}^{K} p(c_k|\mathbf{x}_d) = 1$ for all $\mathbf{x}_d$.

Many problems require a distinction between slowly varying variables describing the global boundary conditions and state of the system and fast varying variables describing the dynamics of the system. We will provide examples of systems in which the controlling states are very distinct from the internal states and the output dynamics. A classic example is linear predictive coding (section 6), where the local dynamics are described by

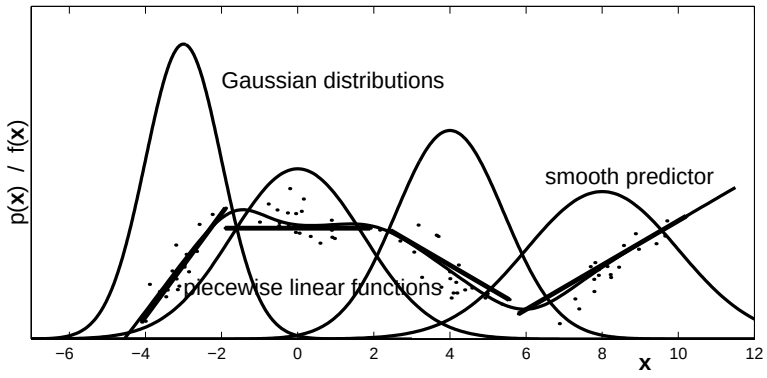


Figure 3-1: One dimensional function approximation with locally linear models weighted by Gaussian kernels.

an IIR filter applied to an excitation sequence, while the selection of the filter coefficients depends on different information concerning the state of the system. In order to distinguish between global (slow) and local (fast) state variables, we decompose $\mathbf{x}$ into $\mathbf{x}_s$ and $\mathbf{x}_f$, and obtain

$$\begin{aligned} p(\mathbf{y}, \mathbf{x}) &= p(\mathbf{y}, \mathbf{x}_s, \mathbf{x}_f) \\ &= \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}_f, c_k)\ p(\mathbf{x}_s|c_k)\ p(c_k) \quad , \end{aligned} \tag{3.5}$$

where $\mathbf{x}_s$ and $\mathbf{x}_f$ may be identical, overlapping in some dimensions or completely distinct.

The input distribution is taken to be a Gaussian distribution,

$$p(\mathbf{x}|c_k) = \frac{|\mathbf{P}_k^{-1}|^{1/2}}{(2\pi)^{D/2}} e^{-(\mathbf{x}-\mathbf{m}_k)^T \cdot \mathbf{P}_k^{-1} \cdot (\mathbf{x}-\mathbf{m}_k)/2} \quad , \tag{3.6}$$

where $\mathbf{P}_k$ is the cluster-weighted covariance matrix in the feature space. The covariance matrix can be reduced to the diagonal terms when simplicity is important, which reduces the Gaussians to

$$p(\mathbf{x}|c_k) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}\sigma_{d,k}} e^{-\sum_{d=1}^{D} \frac{(x_d - m_{d,k})^2}{2\sigma_{d,k}^2}} \quad . \tag{3.7}$$

where D is the dimension of $\mathbf{x}$.

In the case of a continuous valued output vector $\mathbf{y}$, the output distribution is taken to be

$$p(\mathbf{y}|\mathbf{x}, c_k) = \frac{|\mathbf{P}_{k,y}^{-1}|^{1/2}}{(2\pi)^{D_y/2}} e^{-(\mathbf{y}-\mathbf{f}(\mathbf{x},\mathbf{a}_k))^T \cdot \mathbf{P}_{k,y}^{-1} \cdot (\mathbf{y}-\mathbf{f}(\mathbf{x},\mathbf{a}_k))/2} \quad , \tag{3.8}$$

where the mean value of the output Gaussian is replaced by the function $\mathbf{f}(\mathbf{x}, \mathbf{a}_k)$ with unknown parameters $\mathbf{a}_k$. As in (3.7) the off-diagonal terms in the output-covariance

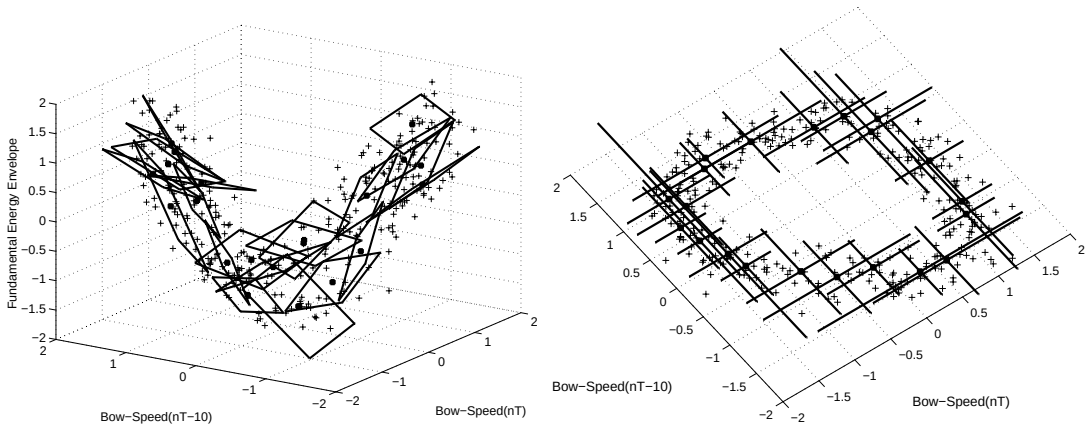


Figure 3-2: Clustered violin data. Two-dimensional function approximation with locally linear models where the input $\mathbf{x}$ consists of time lagged bowing data and the output is the amplitude of the first harmonic. *Left:* vertical view on the input space. *Right:* three-dimensional view of input space and output space.

matrices $\mathbf{P}_{k,y}$ can be dropped if needed.

To understand (3.8) consider the conditional forecast of $\mathbf{y}$ given $\mathbf{x}$,

$$\begin{aligned}
\langle \mathbf{y}|\mathbf{x} \rangle &= \int \mathbf{y}\ p(\mathbf{y}|\mathbf{x})\ d\mathbf{y} \qquad (3.9)\\
&= \int \mathbf{y}\ \frac{p(\mathbf{y},\mathbf{x})}{p(\mathbf{x})}\ d\mathbf{y}\\
&= \frac{\sum_{k=1}^{K} \int \mathbf{y}\ p(\mathbf{y}|\mathbf{x},c_k)\ d\mathbf{y}\ p(\mathbf{x}|c_k)\ p(c_k)}{\sum_{k=1}^{K} p(\mathbf{x}|c_k)\ p(c_k)}\\
&= \frac{\sum_{k=1}^{K} \mathbf{f}(\mathbf{x},\mathbf{a}_k)\ p(\mathbf{x}|c_k)\ p(c_k)}{\sum_{k=1}^{K} p(\mathbf{x}|c_k)\ p(c_k)} \quad .
\end{aligned}$$

We observe that the predicted $\mathbf{y}$ is a superposition of all the local functions, where the weight of each contribution depends on the posterior probability that an input point was generated by a particular cluster. The denominator assures that the sum over the weights of all contributions equals unity.

We choose expression 3.9, the expectation of $\mathbf{y}$ given $\mathbf{x}$, to be our predictor of $\hat{\mathbf{y}}$ (*Bayes' least square* estimator, Section 5.1.1). This is not the only choice, but a very convenient and powerful one considering the two alternatives: the *maximum a posteriori* predictor of $\mathbf{y}$ finds $\hat{\mathbf{y}}_{\mathrm{MAP}} = \mathrm{argmax}_y\ p(\mathbf{y}|\mathbf{x})$. This may seem a good choice, however $\hat{\mathbf{y}}$ can't be evaluated analytically. The medium of $\mathbf{y}$ (*minimum absolute error* estimator) is equally difficult to compute. Considering that one or two basis terms are predominant for most input $\mathbf{x}$, the three options yield very similar results anyway, since medium, mean, and mode are identical for the Gaussian distribution.

We also compute the conditional covariance of $\mathbf{y}$ given $\mathbf{x}$,

$$\begin{aligned}
\langle \mathbf{P}_y|\mathbf{x}\rangle &= \int (\mathbf{y}-\langle \mathbf{y}|\mathbf{x}\rangle)(\mathbf{y}-\langle \mathbf{y}|\mathbf{x}\rangle)^T \; p(\mathbf{y}|\mathbf{x}) \; d\mathbf{y} \qquad (3.10)\\
&= \int (\mathbf{y}\mathbf{y}^T - \langle \mathbf{y}|\mathbf{x}\rangle\langle \mathbf{y}|\mathbf{x}\rangle^T) \; p(\mathbf{y}|\mathbf{x}) \; d\mathbf{y}\\
&= \frac{\sum_{k=1}^K \int \mathbf{y}\mathbf{y}^T p(\mathbf{y}|\mathbf{x},c_k) d\mathbf{y} \; p(\mathbf{x}|c_k) \; p(c_k)}{\sum_{k=1}^K p(\mathbf{x}|c_k) \; p(c_k)} - \langle \mathbf{y}|\mathbf{x}\rangle\langle \mathbf{y}|\mathbf{x}\rangle^T\\
&= \frac{\sum_{k=1}^K [\mathbf{P}_{k,y} + \mathbf{f}(\mathbf{x},\mathbf{a}_k)\mathbf{f}(\mathbf{x},\mathbf{a}_k)^T] \; p(\mathbf{x}|c_k) \; p(c_k)}{\sum_{k=1}^K p(\mathbf{x}|c_k) \; p(c_k)} - \langle \mathbf{y}|\mathbf{x}\rangle\langle \mathbf{y}|\mathbf{x}\rangle^T
\end{aligned}$$

which equals the expected variance if only a single output dimension is considered,

$$\langle \sigma_y^2|\mathbf{x}\rangle = \frac{\sum_{k=1}^K [\sigma_{k,y}^2 + f(\mathbf{x},\mathbf{a}_k)^2] \; p(\mathbf{x}|c_k) \; p(c_k)}{\sum_{k=1}^K p(\mathbf{x}|c_k) \; p(c_k)} - \langle y|\mathbf{x}\rangle^2 \quad . \qquad (3.11)$$

Espressions (3.10) and (3.11) indicate the conditional uncertainty associated with an input $\mathbf{x}$ and a prediction $\mathbf{y}$, which in turn is a measure of the potential prediction error.

The choice of Gaussian Kernels is mostly one of convenience. Gaussians are described compactly with view parameters, and they generalize nicely in higher dimensions. Furthermore, one may claim that the world, according to the Central Limit theorem, can be interpreted as a bunch of Gaussian distributions afterall [Ger99a]. However, the truth is that we don't really care a lot about the form of our kernels, since we are using them mostly as a means to select the local predictors. For most applications we are not interested in the actual density estimator but use the kernels to weight the local predictors.

While the general framework of CWM applies independently of the application, the choice of the local model and the output distribution provides the flexibility needed to deal with specific applications. In general, the output function expresses prior beliefs about the data set and the dynamics of a specific system.

If the output function is unordered and discrete valued, the output distribution collapses into a simple probability table, which has as many columns as there are discrete valued dimensions.

$$p(\mathbf{y}|\mathbf{x},c_k) = p(\mathbf{y}|c_k) = \begin{pmatrix} p(x_1=1|c_k) & p(x_2=1|c_k) & ... & p(x_D=1|c_k) \\ p(x_1=2|c_k) & p(x_2=2|c_k) & ... & p(x_D=2|c_k) \\ ... & ... & ... & ... \\ p(x_1=L|c_k) & p(x_2=L|c_k) & ... & p(x_D=L|c_k) \end{pmatrix} \quad , \qquad (3.12)$$

where L is the number of different labels and D is the number of discrete dimensions. Every column in the probability table sums to unity.

If the output function is real-valued we constrain the models $\mathbf{f}_k$ to be generalized linear models as in (3.1). Although any functional form could be used, (3.1) provides a good trade off between model performance and ease of use. Common models that satisfy form (3.1) are linear models

$$f_k(\mathbf{x}) = a_{0,k} + \sum_{i=1}^D a_{i,k} x_i \quad , \qquad (3.13)$$

where D is the dimensionality of $\mathbf{x}$, and higher order polynomial models

$$\begin{aligned} f_k(\mathbf{x}) &= a_{0,k} + \sum_{m=1}^{M} a_{m,k}\Psi_m(\mathbf{x}) \quad \text{, with} \\ \Psi_m(\mathbf{x}) &= \prod_i x_i^{e_{i,m}} \end{aligned} \tag{3.14}$$

where $e_{i,m}$ depends on the order of polynomial approximation (section 5.1.4). For some applications, non-polynomial basis terms may be preferable, for example cosine or wavelet filterbanks.

$\mathbf{f}$ expresses prior beliefs about the nature of the data or the mechanics of the system, and hence functions as a regularizer of the model. Machine learning architectures and estimation algorithms typically depend on global regularizers that handle prior beliefs about what a good model is. This is problematic since global statements may not apply locally. For example, the maximum entropy principle is good at handling discontinuities, but has no notion of local smoothness, whereas integrated curvature is good in enforcing local smoothness but rounds out discontinuities [SG00]. In our approach, the model is constrained only by the local architecture which may enforce local smoothness but at the same time allows for discontinuities where needed.

In order to control under versus over-fitting we trade the complexity of the local models for the complexity of the global weighting framework [WG93]. Consider the approximation of a one-dimensional smooth function as in Fig. 7-6. In the one extreme case we use a single cluster along with a 5th order local polynomial model. The CWM model collapses into a global polynomial model. In the other extreme case we use ten clusters along with constant local functions, hence the predictive power of the model is only determined by the number of Gaussian kernels. Both approximations are reasonably good. However, a perfect fit is obtained by using a moderate number of five clusters along with moderate $3rd$-order polynomials.

3.2 Model estimation

The model parameters are found in an iterative search which uses two estimators combined in a joint update. We use the expectation-maximization algorithm (EM) to find the parameters of the Gaussian kernels and fit the local model parameters by an inversion of the local covariance matrix.

The EM algorithm has been widely appreciated as an efficient training algorithm for probabilistic networks [DLR77, NH93, Ama95]. Given some observed data, EM assumes that there is a set of known states (the observed data) and a set of hidden states, characterizing the model. If the hidden states were known, model estimation would be easy, because we would only need to maximize a parameterized distribution. Yet, since we don't know the hidden states we need an iterative search for a solution that satisfies the constraints on the hidden states and maximizes the likelihood of the known states. The EM algorithm converges to a maximum of the likelihood of the observed data, reachable from the initial conditions. Given that there are many equivalent solutions for the model,

this approach is typically acceptable.[2]

Unlike conventional kernel-based techniques, CWM requires only one hyper-parameter to be fixed beforehand, the number of Gaussian kernels K. Other parameters of the model, such as the *bandwidth* (the domain of influence) of the Kernels, are results of the estimation process rather than an input to the training algorithm [CD88]. K is determined by cross-validation on left-out data or in a bootstrapingp approach (section 5.3.2).

Clusters are initialized in space locations where there is data, which saves time, since clusters don't have to find subspaces that are populated with data points. We pick as many random points from the data set as there are clusters in the models and use the data coordinates to initialize the cluster means for input and output. The remaining output coefficients are set to zero. The cluster input and output variances are chosen to cover the support of the data, i.e. the diagonal terms of the cluster covariance matrices equal the variance of the data set in each dimension. We set cluster probabilities $p(c_k)$ to $1/K$ in order to give clusters equal starting weight. The measured data is normalized to zero mean and unit variance since arbitrary scaled data may cause probabilities to become smaller than machine precision.

The iteration process is kicked off with the **E-step**, for which we assume the current cluster parameters to be correct and evaluate the posterior probabilities that relate each cluster to each data point in the conditional probability $p(c_k|\mathbf{y},\mathbf{x})$. These posterior probabilities can be interpreted as the probability that a particular piece of data was generated by a particular cluster. Other authors refer to them as the responsibilities of a cluster for a point [JJ94].

$$\begin{aligned} p(c_k|\mathbf{y},\mathbf{x}) &= \frac{p(\mathbf{y},\mathbf{x}|c_k)\ p(c_k)}{p(\mathbf{y},\mathbf{x})} \\ &= \frac{p(\mathbf{y},\mathbf{x}|c_k)\ p(c_k)}{\sum_{l=1}^{K} p(\mathbf{y},\mathbf{x}|c_l)\ p(c_l)} \end{aligned} \tag{3.15}$$

The sum in the denominator causes clusters to interact, fight over points, and specialize in data they best explain. Each cluster wants to own as many points as possible, but gets pushed back by competing clusters.

In the **M-step** we assume the current data distribution to be correct and find the cluster parameters that maximize the likelihood of the data. We estimate the new unconditional cluster probabilities $p(c_k)$ factoring over the entire current density $p(\mathbf{y},\mathbf{x})$.

$$\begin{aligned} p(c_k) &= \int p(c_k|\mathbf{y},\mathbf{x})\ p(\mathbf{y},\mathbf{x})\ d\mathbf{y}\ d\mathbf{x} \\ &\approx \frac{1}{N}\sum_{n=1}^{N} p(c_k|\mathbf{y}_n,\mathbf{x}_n) \end{aligned} \tag{3.16}$$

In the second line, the integral over the density is replaced by the sum over the known data. In the limit of infinite data, these expressions are equal. In the case of limited data,

[2] Gershenfeld [GW93] refers to this feature of large parameter spaces typical for neural networks the *blessing of dimensionality* in reference to the *curse of dimensionality* mentioned earlier. Appendix A provides different theoretical explanations of the EM algorithm.

the given data distribution is still the best approximation.

Next we compute the expected input mean of each cluster, which is an estimate for the new cluster means:

$$\begin{aligned}
\mathbf{m}_k &= \int \mathbf{x}\ p(\mathbf{x}|c_k)\ d\mathbf{x} \qquad (3.17)\\
&= \int \mathbf{x}\ p(\mathbf{y},\mathbf{x}|c_k)\ d\mathbf{y}\ d\mathbf{x}\\
&= \int \mathbf{x}\ \frac{p(c_k|\mathbf{y},\mathbf{x})}{p(c_k)}\ p(\mathbf{y},\mathbf{x})\ d\mathbf{y}\ d\mathbf{x}\\
&\approx \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} \mathbf{x}_n\ p(c_k|\mathbf{y}_n,\mathbf{x}_n)\\
&= \frac{\sum_{n=1}^{N} \mathbf{x}_n\ p(c_k|\mathbf{y}_n,\mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|\mathbf{y}_n,\mathbf{x}_n)}
\end{aligned}$$

The formal introduction of $\mathbf{y}$ into the density has the important result that cluster parameters are found with respect to the joint input-output space. Clusters get pulled depending on where there is data to be explained and how well their model explains the data. In a similar way, we can define the cluster-weighted expectation of any function $\theta(\mathbf{x})$,

$$\begin{aligned}
\langle\theta(\mathbf{x})\rangle_k &\equiv \int \theta(\mathbf{x})\ p(\mathbf{x}|c_k)\ d\mathbf{x} \qquad (3.18)\\
&\approx \frac{1}{N} \sum_{n=1}^{N} \theta(\mathbf{x}_n)\ \frac{p(c_k|\mathbf{y}_n,\mathbf{x}_n)}{p(c_k)}\\
&= \frac{\sum_{n=1}^{N} \theta(\mathbf{x}_n)\ p(c_k|\mathbf{y}_n,\mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|\mathbf{y}_n,\mathbf{x}_n)} \quad .
\end{aligned}$$

which lets us update the cluster weighted covariance matrices,

$$\begin{aligned}
[\mathbf{P}_k]_{ij} &= \langle (x_i - m_i)(x_j - m_j)\rangle_k \qquad (3.19)\\
&= \frac{\sum_{n=1}^{N} (x_i - m_i)(x_j - m_j)\ p(c_k|\mathbf{y}_n,\mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|\mathbf{y}_n,\mathbf{x}_n)} \quad .
\end{aligned}$$

Alternatively, parameters can be derived by taking the derivative of the log-likelihood of the data with respect to the parameters. For example, the new cluster means need to satisfy

$$0 = \frac{\partial}{\partial m_{k,i}}\ \log \prod_{n=1}^{N}\ p(\mathbf{y}_n,\mathbf{x}_n) \quad , \qquad (3.20)$$

which leads to the update rule:

$$0 = \sum_{n=1}^{N} \frac{\partial}{\partial m_{k,i}}\ \log\ p(\mathbf{y}_n,\mathbf{x}_n) \qquad (3.21)$$

$$
\begin{aligned}
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \frac{2(x_{n,i} - m_{k,i})}{2\sigma_k^2} \; (-1) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \frac{m_{k,i}}{\sigma_k^2} - \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \frac{x_{n,i}}{\sigma_k^2} \quad .
\end{aligned}
$$

Resolving expression 3.21 for m_k we get back equation 3.17:

$$
\begin{aligned}
\sum_{n=1}^{N} \frac{1}{p(y_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \; m_{k,i} &= \sum_{n=1}^{N} \frac{1}{p(y_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \; x_{n,i} \qquad (3.22) \\
m_{k,i} &= \frac{\sum_{n=1}^{N} \frac{1}{p(y_n, x_{n,i})} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \; x_{n,i}}{\sum_{n=1}^{N} \frac{1}{p(y_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k)} \\
&= \frac{1}{N \; p(c_k)} \sum_{n=1}^{N} x_{n,i} \; p(c_k | \mathbf{y}_n, \mathbf{x}_n)
\end{aligned}
$$

In order to update the parameters of the local linear coefficient models, we take the derivative of the log-likelihood with respect to each of the model parameters. Considering a single output dimension y and a single coefficient a_k, this yields

$$
\begin{aligned}
0 &= \sum_{n=1}^{N} \frac{\partial}{\partial a_{k,i}} \; \log \; p(\mathbf{y}_n, \mathbf{x}_n) \qquad (3.23) \\
&= \sum_{n=1}^{N} \frac{1}{p(y_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \frac{y_n - f(\mathbf{x}_n, a_k)}{\sigma_{m,y}^2} \frac{\partial f(\mathbf{x}_n, a_k)}{\partial a_{k,i}} \\
&= \frac{1}{N p(c_k)} \sum_{n=1}^{N} p(c_k | \mathbf{y}_n, \mathbf{x}_n) [y_n - f(\mathbf{x}_n, a_k)] \frac{\partial f(\mathbf{x}_n, a_k)}{\partial a_{k,i}} \\
&= \left\langle [y - f(\mathbf{x}, \mathbf{a}_k)] \; \frac{\partial f(\mathbf{x}, \mathbf{a}_k)}{\partial a_{k,i}} \right\rangle_k
\end{aligned}
$$

Plugging expression 3.1 into expression 3.23, we can solve for $\mathbf{a}_k$

$$
\begin{aligned}
0 &= \langle [y - f(\mathbf{x}, \mathbf{a}_k)] f_j(\mathbf{x}) \rangle_k \qquad (3.24) \\
&= \underbrace{\langle y f_j(\mathbf{x}) \rangle_k}_{c_{j,k}} - \sum_{i=1}^{I} a_{k,i} \; \underbrace{\langle f_j(\mathbf{x}) f_i(\mathbf{x}) \rangle_k}_{\mathbf{B}_{ji,k}} \\
\mathbf{a}_k &= \mathbf{B}_k^{-1} \cdot \mathbf{c}_k \quad .
\end{aligned}
$$

If we consider the full set of model parameters for vector-valued input and output, compact writing of the formulas yields

$$
\mathbf{A}_k = \mathbf{B}_k^{-1} \cdot \; \mathbf{C}_k \quad , \qquad (3.25)
$$

with

$$
[\mathbf{B}_k]_{ij} = \langle f_i(\mathbf{x}, \mathbf{a}_k) \cdot f_j(\mathbf{x}, \mathbf{a}_k) \rangle_k
$$

$$
\begin{aligned}
&= \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} f_i(\mathbf{x}, \mathbf{a}_k) \cdot f_j(\mathbf{x}, \mathbf{a}_k)\ p(c_k|\mathbf{y}_n, \mathbf{x}_n) \\
[\mathbf{C}_k]_{ij} &= \langle y_i \cdot f_j(\mathbf{x}, \mathbf{a}_k) \rangle_k \\
&= \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} y_i \cdot f_j(\mathbf{x}, \mathbf{a}_k)\ p(c_k|\mathbf{y}_n, \mathbf{x}_n) \quad .
\end{aligned}
$$

(3.25) is the well-known maximum likelihood estimator for linear models, modified to fit a local subspace of data instead of the total data set. Because the matrix $\mathbf{A}$ is symmetric, we can use efficient matrix inversion algorithms to compute the inverse, for example a Cholesky decomposition [PTVF92, P.96]. However, it is preferable to use the more expensive but robust *singular value decomposition* (SVD) to compute the inversion, since subspaces may become very small.

In the case of the popular, and often sufficient, local linear models (3.13) expression (3.26) reduces to

$$
\begin{aligned}
[\mathbf{B}_k]_{ij} &= \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} x_i(\mathbf{x}, \mathbf{a}_k) \cdot x_j(\mathbf{x}, \mathbf{a}_k)\ p(c_k|\mathbf{y}_n, \mathbf{x}_n) \qquad (3.26) \\
[\mathbf{C}_k]_{ij} &= \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} y_i \cdot x_j(\mathbf{x}, \mathbf{a}_k)\ p(c_k|\mathbf{y}_n, \mathbf{x}_n) \quad .
\end{aligned}
$$

where $x_0 = 1$ (the constant term in the linear model) and x_i is the i-th component of vector $\mathbf{x}$ for $i > 0$.

Finally, the output covariance matrices $\mathbf{P}_{y,k}$ associated with each model can be estimated. Estimation of $\mathbf{P}_{y,k}$ is analogous to estimation of the error matrix in linear estimation theory. Again, the difference is that we estimate the error in the local neighborhood of the cluster and its predictor.

$$
\begin{aligned}
\mathbf{P}_{y,k} &= \langle [\mathbf{y} - \mathbf{f}(\mathbf{x}, \mathbf{a}_k)] \cdot [\mathbf{y} - \mathbf{f}(\mathbf{x}, \mathbf{a}_k)]^T \rangle_k \\
&= \frac{1}{N\ p(c_k)} \sum_{n=1}^{N} [\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k)] \cdot [\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k)]^T\ p(c_k|\mathbf{y}_n, \mathbf{x}_n) \quad .
\end{aligned}
$$

In the case of a discrete output model, the estimation of the probability table (3.12) basically means *counting points of ownership*. Each cluster is given probabilities that represent its relative amount of ownership with respect to a certain class. Hence the maximization step for the probability table is

$$
p(y = i|c_k) = \frac{\sum_{n|y_n=i} p(c_k|\mathbf{y}_n, \mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|\mathbf{y}_n, \mathbf{x}_n)} \qquad (3.27)
$$

Let's quickly summarize the model estimation procedure:

1. pick some initial conditions;
2. evaluate the local data distribution $p(\mathbf{y}, \mathbf{x}|c_k)$

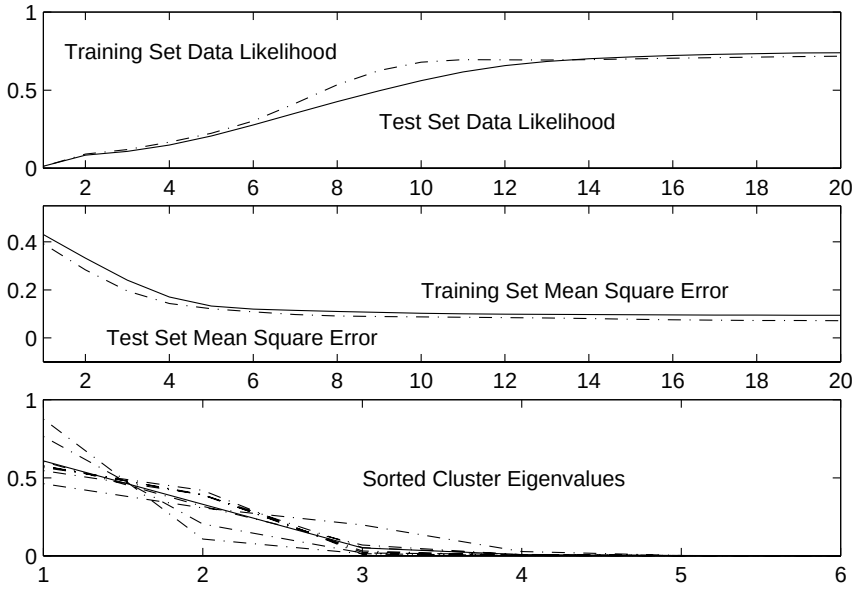


Figure 3-3: Fitting the Lorenz set. *Top*: Data likelihood as a function of iterations. *Middle*: Mean square error as a function of iteration: *Bottom*: Sorted eigenvalues of the local covariance matrices; individual matrices (dashed) and average (solid).

3. given $p(\mathbf{y}, \mathbf{x}|c_k)$ find the posterior probability of the clusters given the data $p(c_k|\mathbf{y}, \mathbf{x})$;
4. update the cluster weights $p(c_k)$, the cluster-weighted input means $\mathbf{m}_k^{new}$, the input variances $\sigma_{k,d}^{2\ new}$ or covariances $\mathbf{P}_k^{new}$, the model parameters $\mathbf{a}_k^{new}$, and finally the output variances $\sigma_{k,y}^{2\ new}$ or covariances $\mathbf{P}_{k,y}^{new}$;
5. go back to (2) until the total data-likelihood stops increasing (fig. 3-3) [DLR77].

3.3 Model evaluation and system characterization

From the probability density (3.3), error estimates and statistics can be derived that provide useful insights as well as a self-consistency checks on the model. Some statistics are derived directly from the model, others require operations on the densities. All of them relate to some established analysis tools in nonlinear systems theory.

The density itself indicates the model uncertainty. We won't obtain a good model where the data density is low, but assume that the certainty of the model estimation is proportional to the data density in a subspace. This is obviously a relative measure, given that the density integrates to one independently from the total amount of training data.

The conditional covariance (3.10), on the other hand, indicates the prediction uncertainty given an input $\mathbf{x}$. Rather than training a second model that predicts error bars for the forecast, the error bars are a result of the estimation process (fig. 3-6). If the model is derived from time series data embedded in a lag space, the scalar-valued conditional

variance can be related to other characterizations of uncertainty, such as entropy and Lyapunov exponents. The differential entropy of a Gaussian process is $H = \log_2(2\pi e\sigma^2)/2$. We ignore the additive terms and consider $H = \log_2(\sigma)$, because only differences in a differential entropy matter. The asymptotic rate of growth of the entropy with time is equal to the source entropy h, which in turn is equal to the sum of positive Lyapunov exponents multiplied by the time lag τ between samples, $h = \tau \sum \lambda^+$. Therefore, assuming that the prediction errors are roughly Gaussian, the asymptotic value of the log of the output width as the input dimension increases provides a local estimate of the source entropy of the system. The sum of the negative exponents can similarly be found by analyzing the time series in reverse order (thereby exchanging positive and negative exponents).

Because clusters find the subspace that is occupied by data, we can use the cluster parameters to find the dimension of the data set even in a high-dimensional space. Intuitively, the number of significant eigenvalues of the local covariance matrices provides an estimate of the dimensionality of the data manifold. For example, we obtain three significant eigenvalues for the Lorenz attractor embedded in six dimensions (fig. 3-3). To quantify this further we use the eigenvalues of the local covariance matrices $E_k = \{e_{1,k}, e_{2,k}, ..., e_{3,k}\}$ to evaluate the radial correlation integral

$$\begin{aligned} C_k(r) &= \int_{-r}^{r} ... \int_{-r}^{r} p(x_1, \ldots, x_D|C_k)\ dx_1 \ldots dx_D \\ &= \operatorname{erf}\left(\frac{r}{\sqrt{2e_{1,m}^2}}\right) \cdots \operatorname{erf}\left(\frac{r}{\sqrt{2e_{D,m}^2}}\right) \end{aligned} \tag{3.28}$$

which in turn lets us compute the cluster's correlation dimension [Ger99a]

$$\begin{aligned} \nu_k &= \frac{\partial \log C_k(r)}{\log r} \\ &= \sum_{d=1}^{D} \frac{1}{\operatorname{erf}\left(\frac{r}{\sqrt{2e_{d,m}^2}}\right)} \sqrt{\frac{2}{\pi e_{d,m}^2}}\ e^{-r^2/2e_{d,m}^2}\ r \end{aligned} \tag{3.29}$$

In the limit $r \to 0$, this dimension is equal to the dimension of the space, because the curvature of the clustered space cannot be seen locally. If it is evaluated at $r = 0.1\sigma_{\max}$, for the $e_{\max}$ direction, the contribution is still 0.997. However, for a direction with variance $e_{\max}/100$, the contribution drops to 10^{-21}. The expected dimension of the whole data set is finally given by the expectation

$$\langle \nu \rangle = \sum_{k=1}^{K} \nu_k\ p(c_k) \quad . \tag{3.30}$$

Unlike a conventional $O(N^2)$ calculation of the dimension of a data set from all the pairs of points, the clusters find the significant places to evaluate the dimension, as well as the appropriate length at which to test the scaling.

3.4 Online implementation

Many applications require on-line estimation or adjustment of the model parameters rather than a fixed model, e.g. echo reduction in bidirectional communication channels or compression filters for differential coding of audio signals. In the context of inference models for musical instruments (chapter 10), the challenge consists in optimizing the model during data collection such that a player can himself verify the quality of the model and improve on its weaknesses.[3]

We redefine the EM-update rules to be used incrementally with new incoming data. We also propose a recursive linear least-square estimator (Kalman filter) to update the coefficients of the local models [Met96]. We consider the case of a single new data point, and scalar outputs $y(n)$. The local models are taken to be linear.

3.4.1 Online EM updates

Let's assume that $N-1$ data points have been seen so far. The cluster probability based on a data is then given by

$$p^{(N-1)}(c_k) = \frac{1}{N-1} \sum_{n=1}^{N-1} p^{(N)}(c_k|y_n, \mathbf{x_n}) \quad , \tag{3.31}$$

where $p^{(N-1)}(c_k|y_n, \mathbf{x_n})$ is the posterior probability estimated from the set of N points. The cluster probability based on N can be estimated as

$$\begin{aligned} p^{(N)}(c_k) &= \frac{1}{N} \sum_{n=1}^{N} p^{(N)}(c_k|y_n, \mathbf{x_n}) \\ &= \frac{1}{N} \left[\sum_{n=1}^{N-1} p^{(N)}(c_k|y_n, \mathbf{x}_n) + p^{(N)}(c_k|y_N, \mathbf{x}_N) \right] \\ &\approx \frac{N-1}{N} p^{(N-1)}(c_k) + \frac{1}{N} p^{(N)}(c_k|y_N, \mathbf{x_N}) \end{aligned} \tag{3.32}$$

The sum over N points is replaced by a single sum which adds in the contribution of the posterior for the new point, assuming that the posteriors of the existing data are unchanged. Similarly, the cluster-weighted expectation for $N-1$ points is

$$\langle \Theta(y, \mathbf{x}) \rangle_k^{(N-1)} = \frac{1}{(N-1)\ p^{(N-1)}(c_k)} \sum_{n=1}^{N-1} \Theta(y_n, \mathbf{x}_n) p^{(N-1)}(c_k|y_n, \mathbf{x}_n) \quad , \tag{3.33}$$

For N points it can be approximated as

$$\langle \Theta(y, \mathbf{x}) \rangle_k^{(N)} = \frac{1}{(N)\ p^{(N)}(c_k)} \sum_{n=1}^{N} \Theta(y_n, \mathbf{x}_n) p^{(N)}(c_k|y_n, \mathbf{x}_n) \tag{3.34}$$

[3] ...an application that has yet to be implemented.

$$\approx \frac{(N-1)\ p^{(N-1)}(c_k)}{N\ p^{(N)}(c_k)}\ \langle \Theta(y,\mathbf{x}) \rangle_k^{(N-1)} + \frac{1}{(N)\ p^{(N)}(c_k)}\ \Theta(y_N,\mathbf{x}_N)\ p^{(N)}(c_k|y_N,\mathbf{x}_N) \quad .$$

This expression has natural limits. If $p^{(N)}(c_k|y_N,\mathbf{x}_N) = 0$ then the cluster expectation is unchanged; if the weight $p^{(N)}(c_k) = 0$ then the cluster takes on the value of the new point. The normalization by the factor N assumes the system is stationary. For non-stationary systems new data points should be given relatively more weight, so that the influence of old data decreases over time. The weights fulfill the same role as the coefficients of a classic moving average filter.

We can exploit expression 3.34 with respect to the first order moment to find the new cluster means, and with respect to the second order moment to find the new cluster input covariances.

$$\mathbf{m}_k^{(N)} = \frac{(N-1)\ p^{(N-1)}(c_k)}{N\ p^{(N)}(c_k)}\ \mathbf{m}_k^{(N-1)} + \frac{1}{N\ p^{(N)}(c_k)}\ \mathbf{x}_N\ p^{(N)}(c_k|y_N,\mathbf{x}_N) \quad . \tag{3.35}$$

$$\mathbf{C}_k^{(N)} = \frac{(N-1)\ p^{(N-1)}(c_k)}{N\ p^{(N)}(c_k)}\ \mathbf{C}_k^{(N-1)} + \frac{1}{N\ p^{(N)}(c_k)}\ \cdot (\mathbf{x}_N - \mathbf{m}_k^N)(\mathbf{x}_N - \mathbf{m}_k^N)^T\ p^{(N)}(c_k|y_N,\mathbf{x_N}) \tag{3.36}$$

3.4.2 Cluster-weighted Kalman-filter updates for the local models

Now we update the coefficients of the local linear models using local Kalman filters for each single model. The updates for a cluster are weighted by the posterior probabilities $p(c_k|\mathbf{x},\mathbf{y})$ indicating the relevance of a new point for an *old* cluster. The derivation of a linear recursive filters are well documented in the literature [Ger99a, BH92]. A classic Kalman filter is defined in terms of a system of linear state space equations,

$$\begin{aligned} \mathbf{x}[n+1] &= \mathbf{E}[n]\cdot\mathbf{x}[n] + \mathbf{F}[n]\cdot\mathbf{v}[n] \\ \mathbf{y}[n+1] &= \mathbf{H}[n]\cdot\mathbf{x}[n] + \mathbf{w}[n] \quad . \end{aligned} \tag{3.37}$$

where the first equation defines the evolution of the state vector, and the second equation describes the measurement update. $\mathbf{v}[n]$ and $\mathbf{w}[n]$ denote the noise associated with the two processes. In order to recursively estimate the coefficients $\mathbf{a}_k$, we choose the following state space representation (switching from n to N to match equations 3.31 - 3.35):

$$\begin{aligned} \mathbf{a}_k[N+1] &= \mathbf{E}_k\cdot\mathbf{a}_k[N] + \mathbf{F}_k\cdot\mathbf{v}_k[N] \\ y_k[N] &= \mathbf{H}_k\cdot\mathbf{a}_k[N] + w[N] \quad . \end{aligned} \tag{3.38}$$

with

$$\mathbf{E}_k = \mathbf{I} \tag{3.39}$$

$$\begin{aligned} \mathbf{F}_k &= \mathbf{I} \\ \mathbf{H}_k &= \begin{bmatrix} 1 & \mathbf{x}^T \end{bmatrix} \quad . \end{aligned}$$

$\mathbf{x}$ is the observed input vector as opposed to the state vector of the filter. $\mathbf{v}_k$ and w_k are assumed to be white Gaussian noise. Both noise terms relate naturally to the update of the error covariance matrices associated with a cluster. The assumption of Gaussianity is not restrictive but should be considered a reasonable working hypothesis that lets us manipulate equations easily. The system's state $\mathbf{a}_k$ is updated to minimize the error covariance of the predictions y, taking into account the posterior probability of the new data owned by c_k. The solution to the problem has the form

$$\mathbf{a}_k[N|N] = \mathbf{a}_k[N|N-1] + \mathbf{G}_k[N] \cdot (y[N] \; - \; \mathbf{H}_k[N] \; \mathbf{a}[N|N-1]) \tag{3.40}$$

where $\mathbf{G}[N]$ is the Kalman gain matrix.

Let's summarize the filter updates:

1. In case that there are no initial values from a prior batch estimation, we initialize the estimators,

$$\begin{aligned} \mathbf{a}[0|-1] &= \mathbf{0} \\ \mathbf{\Lambda}_e[0|-1] &= \mathbf{\Lambda} \end{aligned} \tag{3.41}$$

   Set $N = 0$ and choose $\mathbf{\Lambda}$ in the order of the covariance of the data.

2. Update the cluster parameters according to equations 3.31 through 3.35.

3. Compute the Kalman Gain matrices G_k, each weighted by the posterior probability of cluster c_k, given data N,

$$\begin{aligned} \mathbf{G}_k[N] &= \mathbf{\Lambda}_{e,k}[N|N-1]\mathbf{H}^T[N] \\ &\quad \cdot \Big(\mathbf{H}[N]\mathbf{\Lambda}_{e,k}[N|N-1]\mathbf{H}^T[N]) + \mathbf{\Lambda}_{w,k}[N]\Big)^{-1} \\ &\quad \cdot p^N(c_k|\mathbf{x}_N, y_N) \quad . \end{aligned} \tag{3.42}$$

   Update the $\mathbf{a}_k$ and the associated error covariance matrix $\mathbf{\Lambda}_e$,

$$\begin{aligned} \mathbf{a}_k[N|N] &= \mathbf{a}_k[N|N-1] + \mathbf{G}_k[N] \cdot (y[N] - \mathbf{H}[N] \cdot \mathbf{a}_k[N|N-1]) \\ \mathbf{\Lambda}_{e,k}[N|N] &= \mathbf{\Lambda}_{e,k}[N|N-1] - \mathbf{G}_k[n] \cdot \mathbf{F} \cdot \mathbf{\Lambda}_{e,m}[N|N-1] \quad . \end{aligned} \tag{3.43}$$

4. Predict the new vector of coefficients $\mathbf{a}_k$, as well as the new error covariance matrix,

$$\begin{aligned} \mathbf{a}_k[N+1|N] &= \mathbf{a}_k[N] \\ \mathbf{\Lambda}_{e,k}[N+1|N] &= \mathbf{\Lambda}_{e,k}[N|N] + \mathbf{\Lambda}_{v,k}[N] \end{aligned} \tag{3.44}$$

5. Increment N and go back to (2).

The parameter vector of a cluster and its state space representation are updated depending on how significant a new point is to the cluster. In the limit where the posterior

probability tends towards zero, all the values of a cluster remain the same. $\mathbf{\Lambda}_v[N]$ is used to adjust the inertia of the estimator. The bigger $\mathbf{\Lambda}_v[N]$ is, the slower $\mathbf{a}_k$ is updated. $\mathbf{\Lambda}_{w,k}[N]$ is identical to the error variance $\mathbf{\Lambda}_{yy}[N]$ which is estimated in the EM update.

3.5 A cluster-weighted input-output hidden Markov model

We have presented an input-output model that is sufficiently complex to handle a large variety of prediction problems. However, there are applications where additional hierarchical structure is helpful if not crucial. For example, CWM does not take into account temporal dependency in the data, but treats observations as independent data points. However, considering, for example, audio samples of a violin signal, adjacent data points clearly relate to each other. The current state and output of the instrument depends on the player action milliseconds or even seconds ago. In addition there is dependence at different timescales, such as the note or a phrase level. CWM does not take into consideration any of this temporal structure.

Hidden Markov models (HMMs) have been developed to model and exploit temporal structure in observed data (section 3-5). Since they are strictly derived from probabilistic principles, the hidden state sequence of an HMM can serve as an outer network layer for a system of cluster-weighted models. Assuming J hidden states, we construct J CWM models, each of which is conditioned on a system state.

HMMs are defined in terms of

- the hidden states $q_1, q_2, ..., q_J$;
- the initial unconditional state probabilities $p(q_i)$;
- the state transition probability matrix $\mathbf{A}$ with $a_{i,j} = p(q_{t+1} = i|q_t = j)$, where $p(q_{t+1} = i|q_t = j)$ is the probability that state i succeeds state j;
- the emission probability $p(o|q_t = j)$, which is the probability that the system generates observation o, given that it is in state j (fig. 2-4) [Rab89].

For our purpose, the discrete emission probabilities $p(o \mid q_t = j)$ are replaced with the continuous probability density function $p(\mathbf{x}, \mathbf{y}|q_t = j)$, which generates an input-output HMM as opposed to an HMM-based classifier. Every state q_j is represented by its own little CW model containing one or more clusters (fig. 3-4). For example the data density given state j is

$$\begin{aligned} p(\mathbf{y}, \mathbf{x} \mid q_t = j) &= \sum_{k=1}^{K_j} p(\mathbf{y}, \mathbf{x} \mid c_{k,j}) \\ &= \sum_{k=1}^{K_j} p(\mathbf{y}|\mathbf{x}, c_{k,j})\ p(\mathbf{x}|c_{k,j})\ p(c_{k,j}) \end{aligned} \tag{3.45}$$

and the corresponding estimator of y is

$$\langle \mathbf{y_t} \mid q_t = j \rangle = \frac{\sum_{k=1}^{K_j} \mathbf{f}_k(\mathbf{x}_t)\ p(c_{k,j} \mid \mathbf{x}_t)}{\sum_{k=1}^{K_j} p(c_{k,j} \mid \mathbf{x}_t)} \quad . \tag{3.46}$$

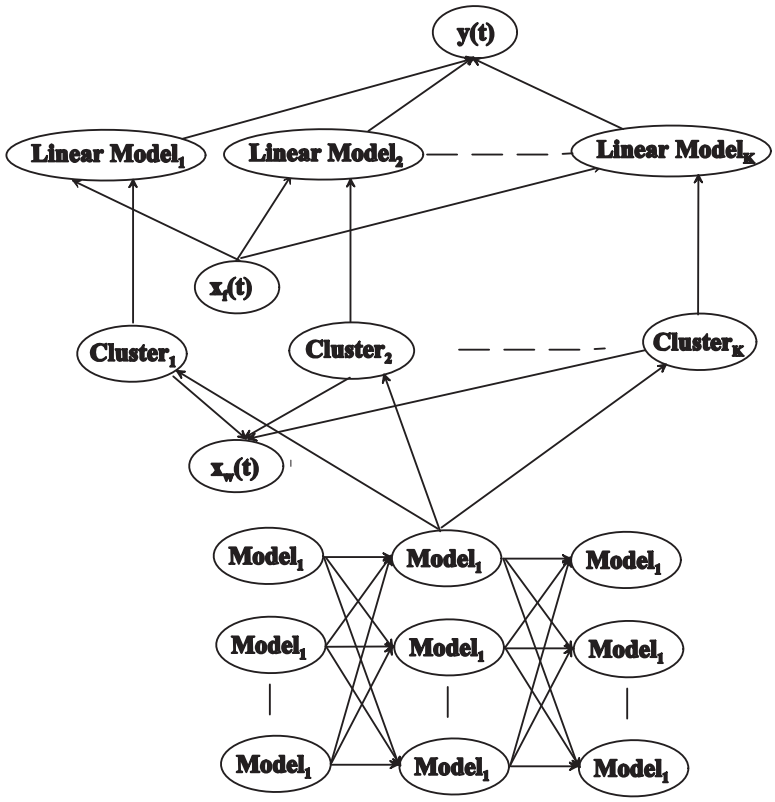


Figure 3-4: Cluster-weighted hidden Markov model. State trellis pointing to local CWM models.

To predict $\mathbf{y}$ we integrate over all the states and get

$$\hat{\mathbf{y}}_t = \frac{\sum_{j=1}^{J} \sum_{k=1}^{K_j} \mathbf{f}_k(\mathbf{x}_t)\ p(c_{k,j}|\mathbf{x}_t)\ p(q_t = j)}{\sum_{j=1}^{J} \sum_{k=1}^{K_j} p(c_{k,j}|\mathbf{x}_t)\ p(q_t = j)} \quad . \tag{3.47}$$

For training, we use the classic forward-backward procedure, a variant of the EM algorithms known as the Baum-Welch algorithm [Rab89]. In the **E-step**, we evaluate the probability of seeing a particular sequence $O \equiv \{\mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_T, \mathbf{x}_T\}$,

$$p(\mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_T, \mathbf{x}_T) = P(O) = \sum_{q_1=1}^{J} ... \sum_{q_T=1}^{J} p(q_1, ..., q_T, O) \tag{3.48}$$

In order to make this expression computable, we expand it recursively.

$$p(O) \quad = \quad \sum_{q_T=1}^{J} p(q_T, O) \tag{3.49}$$

$$
\begin{aligned}
&= \sum_{q_T=1}^{J} p(\mathbf{y}_T, \mathbf{x}_T \mid q_T, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_{T-1}, \mathbf{x}_{T-1})\; p(q_T, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_{T-1}, \mathbf{x}_{T-1}) \\
&= \sum_{q_T=1}^{J} p(\mathbf{y}_T, \mathbf{x}_T \mid q_T)\; p(q_T, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_{T-1}, \mathbf{x}_{T-1}) \\
&= \sum_{q_T=1}^{J} p(\mathbf{y}_T, \mathbf{x}_T \mid q_T) \sum_{q_{T-1}=1}^{J} p(q_T, q_{T-1}, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_{T-1}, \mathbf{x}_{T-1}) \quad ,
\end{aligned}
$$

where the third line follows from the fact that observations depend on the current state only. Factoring over the entire sequence wet get the *forward algorithm* [Ger99a, p.200],

$$
\begin{aligned}
p(O) &= \sum_{q_T=1}^{J} p(\mathbf{y}_T, \mathbf{x}_T \mid q_T) \sum_{q_{T-1}=1}^{J} p(q_T \mid q_{T-1}) p(\mathbf{y}_{T-1}, \mathbf{x}_{T-1} \mid q_{T-1}) \\
&\quad ... \sum_{q_2=1}^{J} p(q_3 \mid q_2)\; p(\mathbf{y}_2, \mathbf{x}_2 \mid q_2) \sum_{q_1=1}^{J} p(q_2 \mid q_1)\; p(\mathbf{y}_1, \mathbf{x}_1 \mid q_1) \qquad (3.50)
\end{aligned}
$$

Likewise we estimate the probability of seeing a sequence O going backwards in time (*backwards algorithm*),

$$
\begin{aligned}
p(O \mid q_t = i) &= \sum_{q_{t+1}=1}^{J} p(q_{t+1}, \mathbf{y}_t, \mathbf{x}_t, ..., \mathbf{y}_T, \mathbf{x}_T \mid q_t = i) \qquad (3.51) \\
&= p(\mathbf{y}_t, \mathbf{x}_t \mid q_t = i) \sum_{q_{t+1}=1}^{J} p(q_{t+1} \mid q_t) p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1}, ..., \mathbf{y}_T, \mathbf{x}_T \mid q_{t+1}) \\
&= p(\mathbf{y}_t, \mathbf{x}_t \mid q_t = i) \sum_{q_{t+1}=1}^{J} p(q_{t+1} \mid q_t)\; p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1} \mid q_{t+1}) \cdot \\
&\quad \sum_{q_{t+2}=1}^{J} p(q_{t+2} \mid q_{t+1})\; p(\mathbf{y}_{t+2}, \mathbf{x}_{t+2} \mid q_{t+2}) \cdot ... \cdot \\
&\quad \sum_{q_{T-1}=1}^{J} p(q_{T-1} \mid q_{T-2})\; p(\mathbf{y}_{T-1}, \mathbf{x}_{T-1} \mid q_{T-1}) \cdot \\
&\quad \sum_{q_T=1}^{J} p(q_T \mid q_{T-1})\; p(\mathbf{y}_T, \mathbf{x}_T \mid q_T)
\end{aligned}
$$

In combining backwards and forward algorithms, we infer joint and conditional probabilities, for example, the probability of a state transition and an observed sequence,

$$
\begin{aligned}
p(q_t = i, q_{t+1} = j, O) &= p(q_t = i, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_t, \mathbf{x}_t)\; p(q_{t+1} = j \mid q_t = i, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_t, \mathbf{x}_t) \cdot \\
&\qquad p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1}, ..., \mathbf{y}_T, \mathbf{x}_T \mid q_t = i, q_{t+1} = j, \mathbf{y}_t, \mathbf{x}_t, ..., \mathbf{y}_T, \mathbf{x}_T) \\
&= p(q_t = i, \mathbf{y}_1, \mathbf{x}_1, ..., \mathbf{y}_t, \mathbf{x}_t)\; p(q_{t+1} = j \mid q_t = i) \cdot \\
&\qquad p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1}, ..., \mathbf{y}_T, \mathbf{x}_T \mid q_{t+1} = j) \quad , \qquad (3.52)
\end{aligned}
$$

We also evaluate the probability of a state given a sequence and the current model,

$$p(q_t = i \mid O) = \frac{p(O \mid q_t = i)}{\sum_{q_\tau=1}^{J} p(O \mid q_\tau)} \quad . \tag{3.53}$$

The data-cluster distribution then is conditioned on the state probability.

$$\begin{aligned} p(c_{k,j} \mid \mathbf{y}, \mathbf{x}, q_t = j) &= \frac{p(\mathbf{y}, \mathbf{x} \mid c_{k,j})\ p(c_{k,j})}{p(\mathbf{y}, \mathbf{x})} \\ &= \frac{p(\mathbf{y}, \mathbf{x} \mid c_{k,j})\ p(c_{k,j})}{\sum_{l=1}^{K} p(\mathbf{y}, \mathbf{x} \mid c_{l,j})\ p(c_{l,j})} \end{aligned}$$

In the **M-step** the HMM as well as the CWM parameters are reestimated. We evaluate the probability of a particular transition at time t given the observed sequence

$$\begin{aligned} p(q_{t+1} = j \mid q_t = i, O) &= \frac{p(q_{t+1} = j, q_t = i, O)}{p(q_t = i, O)} \\ &= \frac{p(q_{t+1} = j, q_t = i, O)}{\sum_{q_{t+1}=1}^{J} p(q_{t+1}, q_t = i, O)} \end{aligned} \tag{3.54}$$

as well as the probability of a hidden state at time t

$$\begin{aligned} p(q_t = i \mid O) &= \frac{p(q_t = i, O)}{p(O)} \\ &= \frac{\sum_{q_{t+1}=1}^{J} p(q_{t+1}, q_t = i, O)}{\sum_{q_t=1}^{J} \sum_{q_{t+1}=1}^{J} p(q_{t+1}, q_t, O)} \end{aligned} \tag{3.55}$$

Averaging over these quantities leads to estimates of the new unconditional state probabilities

$$\begin{aligned} p(q_t = i) &= \frac{\sum_{\tau=1}^{T} p(q_\tau = i \mid O)}{\sum_{j=1}^{J} \sum_{\tau=1}^{T} p(q_\tau = j \mid O)} \\ &= \frac{1}{T} \sum_{\tau=1}^{T} p(q_\tau = i \mid O) \end{aligned} \tag{3.56}$$

and the new state transition probabilities

$$p(q_{t+1} = j \mid q_t = i) = \frac{\sum_{\tau=1}^{T-1} p(q_{\tau+1} = j \mid q_\tau = i)\ p(q_t = i \mid O)}{\sum_{\tau=1}^{T-1} p(q_\tau = i \mid O)} \quad . \tag{3.57}$$

The estimation of cluster parameters is modified with respect to the new posterior probabilities. Hence the new unconditioned cluster probabilities are

$$p(c_{k,j}) = \frac{\sum_{\tau=1}^{T} p(c_{k,j} \mid \mathbf{y}_\tau, \mathbf{x}_\tau, q_\tau = j)}{\sum_{k=1}^{K} \sum_{\tau=1}^{T} p(c_{k,j} \mid \mathbf{y}_\tau, \mathbf{x}_\tau, q_\tau = j)}$$

$$= \frac{1}{T}\sum_{\tau=1}^{T} p(c_{k,j} \mid \mathbf{y}_\tau, \mathbf{x}_\tau, O)$$

Likewise, the cluster-weighted expectation of a function $\Theta(\mathbf{x})$ is modified to

$$\langle\theta(\mathbf{x})\rangle_{k,j} = \frac{\sum_{\tau=1}^{T} \theta(\mathbf{x}_\tau)\ p(c_{k,j} \mid \mathbf{y}_\tau, \mathbf{x}_\tau, q_\tau = j)}{\sum_{\tau=1}^{T} p(c_{k,j} \mid \mathbf{y}_\tau, \mathbf{x}_\tau, q_\tau = j)} \quad .$$

which lets us update the remaining cluster parameters in the usual fashion (equ. 3.17-3.27).

For the purpose of prediction, two basic scenarios may apply. In the case where all the input observations are known beforehand, we choose the output sequence that maximizes the total likelihood of the data with respect to the full input sequence.

$$\langle \mathbf{y}_t \mid \mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T \rangle = \frac{\sum_{q_t=1}^{J}\sum_{k=1}^{K_j} \mathbf{f}_k(\mathbf{x}_t)\ p(c_{k,t}|\mathbf{x}_t)\ p(q_t \mid \mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T)}{\sum_{q_t=1}^{J}\sum_{k=1}^{K_j} p(c_{k,t} \mid \mathbf{x}_t)\ p(q_t \mid \mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T)} \tag{3.58}$$

with

$$p(q_t = j \mid \mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T) = \frac{p(\mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T \mid q_t = j)}{\sum_{q_\tau=1}^{J} p(\mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T \mid q_\tau)} \quad . \tag{3.59}$$

$p(\mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T \mid q_t = j)$ is computed in a recursive forward-backwards procedure.

In the case of real time prediction, we only have access to current and past observations of input data. We therefore maximize the predicted output with respect to past observations only.

$$\langle \mathbf{y}_t \mid \mathbf{x}_1, ..., \mathbf{x}_t \rangle = \frac{\sum_{q_t=1}^{J}\sum_{k=1}^{K_j} \mathbf{f}_k(\mathbf{x}_t)\ p(c_{k,t}|\mathbf{x}_t)\ p(q_t \mid \mathbf{x}_1, ..., \mathbf{x}_t)}{\sum_{q_\tau=1}^{J}\sum_{k=1}^{K_j} p(c_{k,t}|\mathbf{x}_t)\ p(q_t \mid \mathbf{x}_1, ..., \mathbf{x}_t)} \tag{3.60}$$

with

$$p(q_t = j \mid \mathbf{x}_1, ..., \mathbf{x}_t) = \frac{p(\mathbf{x}_1, ..., \mathbf{x}_t \mid q_t = j)}{\sum_{q_\tau=1}^{J} p(\mathbf{x}_1, ..., \mathbf{x}_t \mid q_\tau)} \quad . \tag{3.61}$$

$p(\mathbf{x}_1, ..., \mathbf{x}_t \mid q_t = j)$ is computed recursively in a forward procedure only.

Let's quickly revisit the violin example (Fig. 3-5). Given the state model, a particular sequence of states now reflects a sequence of input gestures and internal states of the violin. Fig. 3-5 illustrates a sequence for simple détaché bowing. We follow a note from the attack, to the sustained part, to the next bow change. Hence the predictor slowly transitions from one state to the next given the boundary conditions $\mathbf{x}$.

3.6 Function approximation under constraints

Any model estimation algorithm should include as many *a priori* constraints on the data as possible. If up front we know about specific properties of the data or the model, this prior information should be imposed as a constraint. Not only can we make sure that the final model fulfills the constraints, we also make more efficient use of the training data. Two applications of constrained models jump to mind.

- When modeling a physical system, such as a microwave device, we have a lot of prior insight into the behavior of the system. In the case of a transistor, for example, a zero input should result in zero output, and the output should saturate below the supply voltage. In addition there may be symmetries. A resistor with a nonlinear current-voltage characteristic should have a symmetric response for positive and negative currents. This means that modeling one half of the curve is sufficient to characterize the device, however, we need to make sure that the boundary conditions at the mirror points are constrained properly.
- No dataset will cover the full real support $\mathcal{R}^n$, yet we would like models to behave reasonably for any feature vector. If we have insights about what the model should do in domains where no data is available, this insight should be added as a constraint to the model. Ideally the model smoothly transitions from the data-driven function into the constrained function, assuring that there is reasonable extrapolation outside the data support.

Lagrange multipliers provide the classic methodology for solving optimization problems under constraints [BS91, P.389]. If we need to minimize the function $f(x, y, y')$ under the constraints $g_j(x, y, y', ...)$, the Lagrange function L is

$$L(x, y, \lambda) = f(x, y) + \sum_{j=1}^{J} \lambda_j(x) \cdot g_j(x, y, y', ...); \quad y = (y_1, ..., y_n) \quad , \tag{3.62}$$

where the λ_j are the Lagrange multipliers. If f is parameterized as $\mathbf{a} = (a_1,\ a_2,\ ...,\ a_I)$, we derive L with respect to the a_i and λ_j, which yields a system of equations to solve for the set of coefficients:

$$\begin{aligned} 0 = \frac{\partial L}{\partial a_i} &= \frac{\partial}{\partial a_i} f(x, y) + \frac{\partial}{\partial a_i} \sum_{j=1}^{J} \lambda_j(x) \cdot g_j(x, y, y', ...) \\ 0 = \frac{\partial L}{\partial \lambda_j} &= g_j(x, y, y', ...) \end{aligned} \tag{3.63}$$

The CWM architecture can be integrated with the Lagrange multipliers in two ways. The first approach constrains the global model with respect to subspaces given by the constraints. Local models are only constrained if the constraint concerns their domain of influence. An implementation of this approach would have to provide an absolute measure of relevance and domain of influence, since models are fit with respect to posterior probabilities indicating the importance of a piece of data for a cluster.

The second solution applies the constraints to all the local functions independently of their domain of influence. If every function satisfies the constraints the linear superposition of all of them does so as well. In this case the set of constraints only changes the search for the local model parameters, while the basic estimation algorithm remains the same (section 3.2). The Lagrange function L is

$$L(\mathbf{y}, \mathbf{x}) = \log \prod_{n=1}^{N} p(\mathbf{y}_n, \mathbf{x}_n) + \sum_{j=1}^{J} \lambda_{j,k}(x) \cdot g_j(x, y, y', ...) \tag{3.64}$$

which is used independently for all of the kernels. The solution for cluster c_k is given by

$$\begin{aligned} 0 = \frac{\partial}{\partial a_{i,k}} L &= \frac{\partial}{\partial a_{i,k}} \log \prod_{n=1}^{N} p(\mathbf{y}_n, \mathbf{x}_n) + \frac{\partial}{\partial a_{i,k}} \sum_{j=1}^{J} \lambda_j(x) \cdot g_j(x, y, y', ...) \\ &= \sum_{n=1}^{N} \frac{\partial}{\partial a_{i,k}} \log\ p(\mathbf{y}_n, \mathbf{x}_n | c_k)\ p(c_k) + \frac{\partial}{\partial a_{i,k}} \sum_{j=1}^{J} \lambda_j(x) \cdot g_j(x, y, y', ...) \\ &\quad \text{with} i = 1...\text{I, and} \\ 0 = \frac{\partial}{\partial \lambda_{j,k}} L &= g_j(x, y, y', ...). \end{aligned} \tag{3.65, 3.66}$$

Equations 1 through I can be rewritten as

$$0 = \left\langle [y - f(\mathbf{x}, \mathbf{a}_k)]\ \frac{\partial f(\mathbf{x}, \mathbf{a}_k)}{\partial a_{i,k}} \right\rangle_k + \frac{\partial}{\partial a_i} \sum_{j=1}^{J} \lambda_j(x) \cdot g_j(x, y, y', ...) \quad . \tag{3.67}$$

If the constraints are generalized linear functions of $\mathbf{x}$ and $\mathbf{y}$, the system of equations is linear and the solution requires a simple matrix inverse. Given that the derivatives of the basis functions are known and continuous, we can also add constraints on the derivatives of f. In this case, in order for solutions to be nontrivial when using polynomial basis functions, the order of the polynomial needs to be sufficiently high.

Example I

We consider the example of a locally polynomial function approximation, which is constrained to vanish on the x_1 axis, i.e. $y(\mathbf{x} = [x_1, 0, ..., 0]) = 0$. The Lagrange function is

$$L(y, \mathbf{x}) = \log \prod_{n=1}^{N} p(\mathbf{y}_n, \mathbf{x}_n) + \lambda_1 \cdot [y([x_1, 0, ..., 0]) - 0] \tag{3.68}$$

which yields

$$\begin{aligned} 0 &= \left\langle [y - f(\mathbf{x}, \mathbf{a}_k)]\ \frac{\partial f(\mathbf{x}, \mathbf{a}_k)}{\partial a_{i,k}} \right\rangle_k + \frac{\partial}{\partial a_i} \lambda_1 \cdot f([x_1, 0, ..., 0], \mathbf{a}_k) \\ &= \langle [y - f(\mathbf{x}, \mathbf{a}_k)]\ \Psi_i(\mathbf{x}) \rangle_k + \lambda_1 \Psi_i([x_1, 0, ..., 0]) \\ 0 &= \sum_{i=1}^{I} a_{i,k} \Psi_i([x_1, 0, ..., 0]) \end{aligned} \tag{3.69}$$

The solution of this system in terms of $a'_{i,k}$ and λ_1 is

$$\mathbf{a}'_k = \mathbf{B}'^{-1}_k \cdot \mathbf{c}'_k \quad , \tag{3.70}$$

with

$$\mathbf{a}'_k \equiv \begin{pmatrix} a_{0,k} \\ a_{1,k} \\ ... \\ a_{I,k} \\ \lambda_1 \end{pmatrix}; \qquad \mathbf{c}'_k = \begin{pmatrix} \langle y\Psi_0(\mathbf{x})\rangle \\ ... \\ \langle y\Psi_I(\mathbf{x})\rangle \\ 0 \end{pmatrix}; \tag{3.71}$$

$$\mathbf{B}'_k = \begin{pmatrix} \langle\Psi_0(\mathbf{x})\Psi_0(\mathbf{x})\rangle & ... & \langle\Psi_0(\mathbf{x})\Psi_I(\mathbf{x})\rangle & -\Psi_0([x_1,0,...,0]) \\ ... & ... & ... & ... \\ \langle\Psi_I(\mathbf{x})\Psi_0(\mathbf{x})\rangle & ... & \langle\Psi_I(\mathbf{x})\Psi_I(\mathbf{x})\rangle & -\Psi_I([x_1,0,...,0]) \\ -\Psi_0([x_1,0,...,0]) & ... & -\Psi_I([x_1,0,...,0]) & 0 \end{pmatrix}$$

Example II

We consider another example $f : \mathcal{R}^2 \Rightarrow \mathcal{R}$ $(\mathbf{x} \Rightarrow y)$ where f as well as the first derivative of f vanish for $x_2 = 0$. The local polynomials are chosen to be up to second-order polynomials. In this case the constraints are

$$\begin{aligned} f(\mathbf{x})|_{\mathbf{x}=[x_1,0]^T} &= 0 \\ \left.\frac{\partial f(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}=[x_1,0]^T} &= 0 \end{aligned} \tag{3.72}$$

yielding the Lagrange function

$$L(y,\mathbf{x}) = \log \prod_{n=1}^{N} p(y_n,\mathbf{x}_n) + \lambda_1[f([x_1,x_2]) - 0] + \lambda_2[\frac{\partial f(\mathbf{x})}{\partial x_2}|_{\mathbf{x}=[x_1,0]} - 0] \quad . \tag{3.73}$$

with

$$\begin{aligned} f_k(\mathbf{x}) &= \sum_{i=0}^{6} a_i\Psi_i(\mathbf{x}) = a_0\ 1 + a_1\ x_1 + a_2\ x_1^2 + a_3\ x_1\ x_2 + a_4\ x_2 + a_5\ x_2^2 \\ f_k|_{\mathbf{x}=[x_1,0]^T} &= a_0\ 1 + a_1\ x_1 + a_2\ x_1^2 = 0 \\ \left.\frac{\partial f_k}{\partial x_2}\right|_{\mathbf{x}=[x_1,0]^T} &= a_3\ x_1 + a_4 + 2\ a_5\ x_2|_{x_2=0} = a_3\ x_1 + a_4 = 0 \end{aligned} \tag{3.74}$$

The solution in terms of $\mathbf{a}_k$ and λ_1 is

$$\mathbf{a}'_k = \mathbf{B}'^{-1}_k \cdot \mathbf{c}'_k \quad , \tag{3.75}$$

with

$$\mathbf{a}'_k \equiv \begin{pmatrix} a_{0,k} \\ a_{1,k} \\ ... \\ a_{I,k} \\ \lambda_1 \\ \lambda_2 \end{pmatrix}; \qquad \mathbf{c}'_k = \begin{pmatrix} \langle y \rangle \\ \langle y\ x_1 \rangle \\ ... \\ \langle y x_2^2 \rangle \\ 0 \\ 0 \end{pmatrix}; \tag{3.76}$$

$$\mathbf{B}'_k = \begin{pmatrix} \langle 1 \rangle & \langle x_1 \rangle & ... & \langle x_2^2 \rangle & -1 & 0 \\ \langle x_1 \rangle & \langle x_1^2 \rangle & ... & \langle x_1\ x_2^2 \rangle & x_1 & 0 \\ ... & ... & ... & ... & ... & ... \\ \langle x_2^2 \rangle & \langle x_1\ x_2^2 \rangle & ... & \langle x_2^4 \rangle & x_2^2 & 0 \\ -1 & x & ... & x_2^2 & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & 0 \end{pmatrix} \tag{3.77}$$

From equations 3.72 we see that the coefficients $a_0, ..., a_4$ need to equal 0, which lets us eliminate the equations in the linear system and solve for $a_5 \neq 0$.

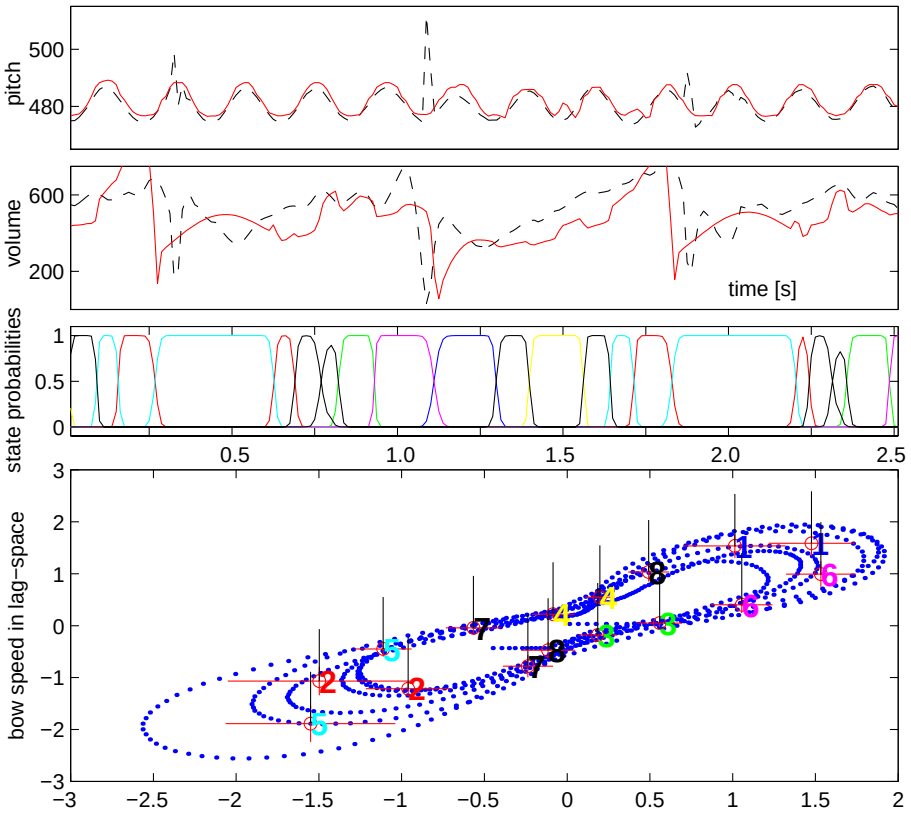


Figure 3-5: Hidden Markov model, *from bottom*: cluster/model input space, two clusters per state; state probabilities; predicted out-of-samples amplitude (measured (dashed line) and predicted (solid line)); predicted out-of-samples pitch (measured (black and dashed) and predicted (red and solid line)). Although measured and predicted data are visibly different, the reconstructed audio sounds similar to the original audio data, since the spectral characteristics and the basic characteristics of the sound envelope are preserved. The model abstracts from the glitches in the measured data.

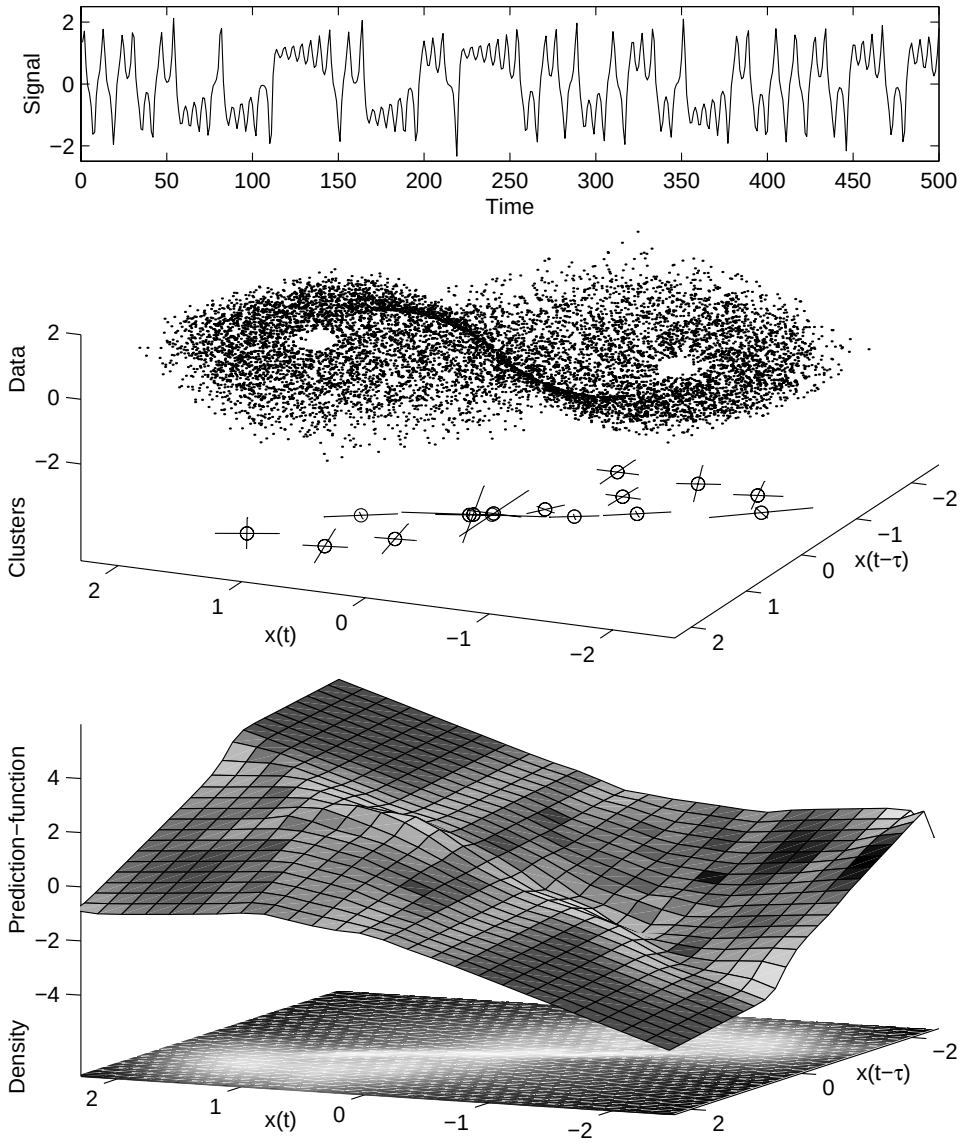


Figure 3-6: Lorenz set, embedded in a three dimensional lag space. The dense dots show the embedded data. Below it are the cluster means and covariances and the derived input density estimate; above it is the prediction surface shaded by the conditional uncertainty, showing the maxima associated with the orbit reinjection.

Part II

Synthesis Architectures and Applications

> Handle so, daß die Maxime Deines Willens jederzeit zugleich als Prinzip einer allgemeinen Gesetzgebung gelten könne.
> [...]Die reine Geometrie hat Postulate as praktische Sätze, die aber nichts weiter enthalten als die Voraussetzungen, daß man etwas tun *könne*, wenn etwa gefordert würde, man *sole* es thun und diese sind die einzigen Sätze derselben, die ein Dasein betreffen.
> I. Kant, *Kritik der praktischen Vernunft*, V30.[4]

In the second of his three major critiques, Kant discusses the fundamentals of human ethics. *Practical* is his term for reasoning, decisions, and actions that affect the world around us, as opposed to the pure ability of reasoning, which was the subject of the first critique. In his famous categorical imperative, Kant says that whatever you do, make sure that the intent of your action could just as well be the universal law.

In this second part, we introduce CWM-related algorithms that are motivated by practical problems and specific data. Often times the decision for a model architecture trades performance for elegance and compactness of description. We will decide in favor of beauty more often than past engineering *practice* without compromising the results.

Many sophisticated applications of linear signal processing require extra layers of often ad hoc processing to handle genuinely nonlinear problems. For example, in speech synthesis, different linear filters are used; however, there is no systematic filter selection. With our generalized framework, we hope to unify some of these *hacks* in a single language. We typically embed mature techniques in the CWM framework and obtain globally nonlinear models that locally collapse into models well known from signal processing past practice.

Each chapter begins with a brief summary of basic concepts and related techniques. These tools are then integrated into CWM, and finally applications and experimental results are discussed. Some of the algorithms will be applied in Part III instead.

[4] So act that the maxim of your will could always hold at the same time as the principle of a universal legislation.
[...]Pure geometry has postulates as practical propositions, which, however, contain nothing more than the presupposition that one *can* do something and that, when some result is needed, one *should* do it; these are the only propositions of pure geometry which apply to an existing thing.
I. Kant. *Critique of Practical Reason.* English translation by L.W. Beck.

Chapter 4

Classification

4.1 Classification and decision theory

Many engineering problems, ranging from communication to fire alarm systems, require decision-making based on a number of measurements [WWS96]. Given a noisy communication channel corrupting the message, the receiver needs to decide if a 0 bit or a 1 bit was sent. Likewise, a fire alarm system interprets sensor data at any moment in time and decides if there was a fire and whether the alarm should go off or not. The sensor readings will be noisy for a variety of reasons. Inevitably, the sensors will be affected by measurement noise [Ger00b], but there could also be systematic noise (e.g. people smoking in the room).

The common methodology for these kinds of problems is that of hypothesis testing. We know beforehand how many and what kind of possible decisions there are and denote the set of hypotheses $\{H_0, H_1, ..., H_{L-1}\}$. For example, in the case of the fire alarm system and the binary communication channel $L = 2$, in the case of ternary logic $L = 3$. We denote the vector of measurements $\mathbf{x} = [x_1, x_2, ..., x_D]$. Each H_i is considered a random variable with an *a priori* (non-conditional) probability $P(H = H_i)$, which summarizes the prior believes regarding the likelihood of a hypothesis. We also establish a measurement model which relates $\mathbf{x}$ to a set of hypotheses and is expressed in terms of the conditional probabilities $p(\mathbf{x} \mid H = H_i)$.

Ultimately, we are interested in the probability of a hypothesis given a measurement, which we compute using Bayes's theorem.

$$\begin{aligned} p(H = H_i \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid H = H_i) \cdot p(H = H_i)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x} \mid H = H_i) \cdot p(H = H_i)}{\sum_{l=0}^{L-1} p(\mathbf{x} \mid H = H_l) \cdot p(H = H_l)} \end{aligned} \tag{4.1}$$

where the denominator assures that the $p(H = H_i \mid \mathbf{x})$ add up to unity.

The second important element in decision theory is the actual decision making. Based on the observation and the measurement model, we need to find an optimal decision rule with respect to some cost function. The cost function $C(H_i, H_j)$ is represented in matrix form, where each entry indicates the cost associated with the decision in favor of hypothesis

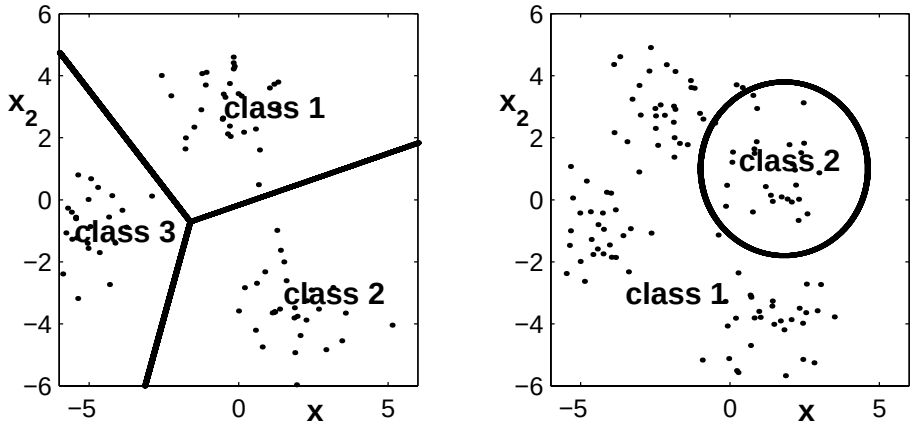


Figure 4-1: *left:* linear classifier with decision regions separated by straight lines, *right:* nonlinear classifier with disconnected decision regions

H_i, given that hypothesis H_j was correct.

$$C = \begin{bmatrix} C(H = H_0 \mid H = H_0) & ... & C(H = H_0 \mid H = H_{L-1}) \\ ... & ... & ... \\ C(H = H_{L-1} \mid H = H_0) & ... & C(H = H_{L-1} \mid H = H_{L-1}) \end{bmatrix} \tag{4.2}$$

Often the diagonal terms will be zero, since the correct decision typically has no cost to it[1]. The decision rule minimizes the expected cost $E[C(H)]$ associated with picking any of the hypotheses.

$$E[C(H_i)] = \sum_{l=0}^{L-1} C(H = H_i \mid H = H_l) \cdot p(H = H_l \mid \mathbf{x}) \tag{4.3}$$

and

$$\hat{H} = \text{argmin}_{H_i} E[C(H_i)] \tag{4.4}$$

Minimizing the expected cost given a measurement model and a cost model equals partitioning the feature space into disjoint regions, each representing a different hypothesis (Fig. 4-1). The underlying concept is the *Likelihood Ratio Test* (LRT). In the case of binary hypothesis testing, the LRT finds the hypothesis that has the lower expected cost:

$$\begin{array}{c} C(H_0 \mid H_0) \cdot p(H = H_0 \mid \mathbf{X}) + \\ C(H_0 \mid H_1) \cdot p(H = H_1 \mid \mathbf{X}) \end{array} \begin{array}{c} \hat{H}(\mathbf{x}) = H_1 \\ \gtrless \\ \hat{H}(\mathbf{x}) = H_0 \end{array} \begin{array}{c} C(H_1 \mid H_0) \cdot p(H = H_0 \mid \mathbf{X}) + \\ C(H_1 \mid H_1) \cdot p(H = H_1 \mid \mathbf{X}) \end{array}$$

[1] There maybe situations where the best (correct) decision is still costly. Say, one finds the ideal mate and decides to get married. Arguably he/she still pays a price in marrying her/him. Just an aside ... .

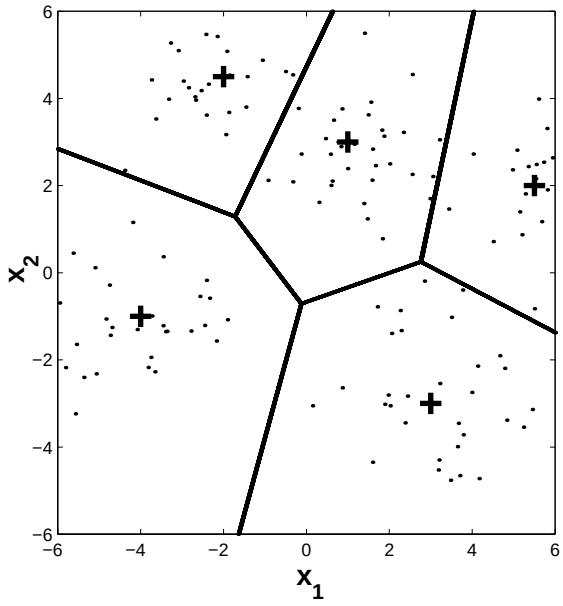


Figure 4-2: Voronoi tesselation

$$(C(H_0 \mid H_1) - C(H_1 \mid H_1)) \cdot p(H = H_1 \mid \mathbf{X}) \underset{\hat{H}(\mathbf{x}) = H_0}{\overset{\hat{H}(\mathbf{x}) = H_1}{\gtrless}} (C(H_1 \mid H_0) - C(H_0 \mid H_0)) \cdot p(H = H_0 \mid \mathbf{X}) \tag{4.5}$$

For a symmetric cost function, all diagonal terms are zero and all off-diagonal terms are 1, which makes all errors equally costly. The decision rule becomes that of a minimum probability-of-error criterion, also named the *maximum a posteriori* (MAP) decision rule since

$$\hat{H} = \text{argmax}_{H_i} P(H = H_i | \mathbf{X}) \quad . \tag{4.6}$$

For binary hypothesis testing this simply means

$$p(H = H_1 \mid \mathbf{X}) \underset{\hat{H}(\mathbf{x}) = H_0}{\overset{\hat{H}(\mathbf{x}) = H_1}{\gtrless}} p(H = H_0 \mid \mathbf{X}) \quad . \tag{4.7}$$

4.1.1 Linear and nonlinear classifiers

Linear classifiers describe decision regions in the feature space which are separated by hyper-planes (straight lines in the two dimensional case, fig. 4-1). Assuming a Gaussian noise model, the measurement model $p(\mathbf{x} \mid H_i)$ is parameterized as a normal distribution

$$p(\mathbf{x} \mid H_i) = \frac{|\mathbf{P}_i|^{-1/2}}{(2\pi)^{D/2}} e^{-(\mathbf{x}-\mathbf{m}_i)^T \cdot \mathbf{P}_i^{-1} \cdot (\mathbf{x}-\mathbf{m}_i)/2} \quad . \tag{4.8}$$

Assuming a symmetric cost function, the LRT for the Gaussian model is

$$L = \frac{p(\mathbf{x} \mid H_1) = \frac{|\mathbf{P}_1|^{-1/2}}{(2\pi)^{D/2}} e^{-(\mathbf{x}-\mathbf{m}_1)^T \cdot \mathbf{P}_1^{-1} \cdot (\mathbf{x}-\mathbf{m}_i)/2}}{p(\mathbf{x} \mid H_0) = \frac{|\mathbf{P}_0|^{-1/2}}{(2\pi)^{D/2}} e^{-(\mathbf{x}-\mathbf{m}_0)^T \cdot \mathbf{P}_0^{-1} \cdot (\mathbf{x}-\mathbf{m}_i)/2}} \begin{array}{c} \hat{H}(\mathbf{x}) = H_1 \\ \gtrless \\ \hat{H}(\mathbf{x}) = H_0 \end{array} \frac{C(H_1 \mid H_0)}{C(H_0 \mid H_1)}\eta \tag{4.9}$$

which, taking the logarithm on both sides, can be rewritten as

$$\begin{array}{ll} \frac{1}{2}\mathbf{x}^T \left[\mathbf{P}_0^{-1} - \mathbf{P}_1^{-1}\right] \mathbf{x} & \hat{H}(\mathbf{x}) = H_1 \\ \quad + \mathbf{x}^T \left[\mathbf{P}_1^{-1}\mathbf{m}_1 - \mathbf{P}_0^{-1}\mathbf{m}_0\right] & \gtrless \\ & \hat{H}(\mathbf{x}) = H_0 \end{array} \begin{array}{l} \ln\eta + \frac{1}{2}\ln\frac{|\mathbf{P}_1|}{|\mathbf{P}_0|} \\ \quad + \frac{1}{2}\left[\mathbf{m}_1^T\mathbf{P}_1^{-1}\mathbf{m}_1 - \mathbf{m}_0^T\mathbf{P}_0^{-1}\mathbf{m}_0\right] \end{array} \tag{4.10}$$

Furthermore, if we constrain the distribution to satisfy $\mathbf{P}_0 = \mathbf{P}_1 = \mathbf{P}$, we get

$$(\mathbf{x}-\mathbf{m}_0)^T\mathbf{P}^{-1}(\mathbf{x}-\mathbf{m}_0^T) \begin{array}{c} \hat{H}(\mathbf{x}) = H_1 \\ \gtrless \\ \hat{H}(\mathbf{x}) = H_0 \end{array} (\mathbf{x}-\mathbf{m}_0)^T\mathbf{P}^{-1}(\mathbf{x}-\mathbf{m}_0) + 2\ln\eta \tag{4.11}$$

and

$$(\mathbf{m}_1-\mathbf{m}_0)^T\mathbf{P}^{-1}\mathbf{x} \begin{array}{c} \hat{H}(\mathbf{x}) = H_1 \\ \gtrless \\ \hat{H}(\mathbf{x}) = H_0 \end{array} \frac{1}{2}(2\ln\eta + \mathbf{m}_1^T\mathbf{P}^{-1}\mathbf{m}_1 - \mathbf{m}_0^T\mathbf{P}^{-1}\mathbf{m}_0) = \eta' \tag{4.12}$$

From this we see that $\mathbf{x}$ is projected onto $(\mathbf{m}_1 - \mathbf{m}_0)^T\mathbf{P}^{-1}$ and compared to the threshold η'. Hence the decision regions are separated by hyper-planes (fig. 4-1) [WWS96].

Nonlinear classifiers allow for decision regions with complex shapes and for regions that are not connected (fig. 4-1). $p(\mathbf{y} \mid H_i)$ is not as easy to describe anymore, but we'll see in the next sections how a mixture density estimator can be used to approximate and parameterize a complex probability density. Let's first look at some less sophisticated techniques to approximate the decision boundaries.

K-means and Voronoi tesselation

A widely used technique to identify labeled regions is nearest neighbor lookup. Let's assume we have some training data labeled A and another set labeled B. We can use the training data itself as our model, by evaluating the distance between any new data point and the training data in terms of the Euclidean distance. We classify a new point according to the training data to which it is closest.

Often we can't afford to keep all the data but need to summarize it efficiently. We can represent (vector-quantize) the training data of class A in terms of a limited number of hard clusters. An easy way to do this is the K-means algorithm. Very much like EM,

K-means is an iterative procedure with two distinct steps per iteration.[2]

1. We start with some initial guesses for the cluster position.
2. We figure which points each cluster *owns* by assigning the closest cluster to each point.
3. We update the position of a cluster to be the center of gravity of all the points it owns.
4. We go back to step (2) unless there weren't any changes in point-cluster assignments.

By clustering data set A and B in the described way, we obtain a mixed representation of the two classes in the same feature space. We classify any new data according to the label of the cluster that it is closest to. The visualization of this decision rule is called a Voronoi tesselation. Two adjacent clusters are separated by the centerline between them (fig. 4-2).[3]

4.1.2 Feature extraction

Many classification problems require preprocessing of the feature data which reduces the dimensionality of the feature space and concentrates the discriminating information. A rough estimate of the amount of data needed to estimate a D-dimensional predictive model is 10^D. This number is not very encouraging but fortunately too conservative for many data sets. Since data clusters in regions of the space, the relationship between the amount of training data and the dimensionality of the space can be much better than exponential [GW93].

In general, we want the dimensionality of the feature space to be as small as possible. Since additional dimensions introduce noise, density estimation is tricky in high-dimensional spaces. At the same time, we need as many discriminating features as possible to reduce the overlap between classes in the feature space. This trade-off needs to be balanced to achieve an optimal result.

PCA

Principal Component Analysis (PCA) rotates the axis of a multidimensional Euclidean space in such a way that the dimensions of the new space are maximally uncorrelated. Hence the transform can be used to find the best orthogonal subspace of a data set. In order to find the PCA unmixing matrix $\mathbf{W}$ one computes the eigenvalues and eigenvectors of the covariance of the data set [Oja83]. The eigenvectors indicate orthogonal directions in

[2] K-means is in fact the *hard*-version of EM applied to Gaussian mixtures. Replacing the variance in the Gaussian by a delta peak and giving equal weight to all clusters, clusters own points or they don't at all. The notion of partial owning is replaced by a winner takes all strategy. [PP93, Pop97].

[3] Support Vector Machines are another sophisticated technique for nonlinear classification and should be mentioned here[Hay99]. They can be superior to density approximation in cases where there is little training data.

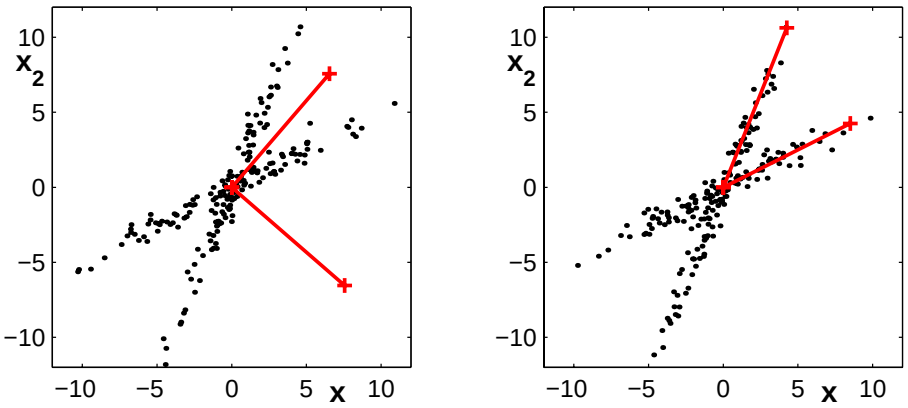


Figure 4-3: *left:* PCA transform of a data set. *right:* ICA transform of a data set.

space while the corresponding eigenvalues indicate the variance in that direction. We extract a subspace of arbitrary dimension by picking eigenvectors ordered by the eigenvalues (strong eigenvalues first) and project the data onto the eigenvectors (new axes).

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x} \tag{4.13}$$

where $\mathbf{x}$ is the original feature vector, $\mathbf{W}$ is the unmixing matrix, and $\mathbf{y}$ is the projected data. $\mathbf{y}$ is of the same or lower dimensionality than $\mathbf{x}$.

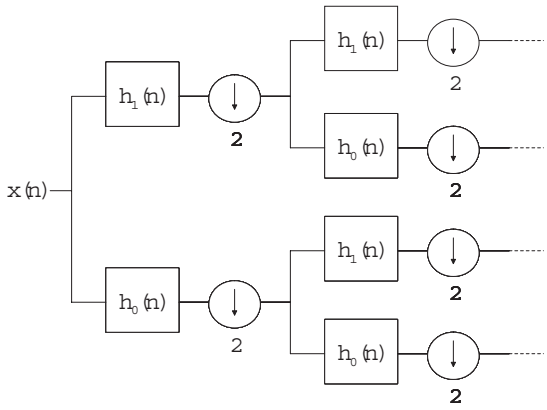


Figure 4-4: Wavelet packet transform. Implementation of the filter bank: $h_0(n)$ and $h_1(n)$ are the half band low-pass and high-pass filters, $2\downarrow$ stands for down-sampling by a factor 2.

The PCA transform is astonishingly simple and very effective at the same time. Not only does it quickly eliminate redundancy in a high dimensional data set, but it also

helps to quickly visualize what a data set is about. Fig. 4-6 shows an example of how a 16-dimensional measurement of brain activity is reduced to 3 to 4 substantial components.

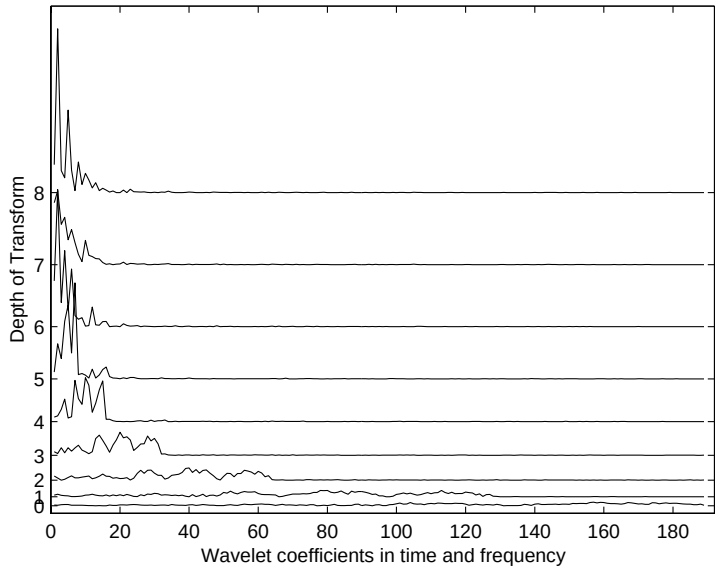


Figure 4-5: Wavelet packet transform applied to difference signal between two brain responses: the y-axis refers to the depth of the transform while the x-axis represents the sub-bands, ordered from left to right. The 0th-order data shows a pure time-domain difference signal while the 8th-order transform shows a pure frequency representation of the signal.

However, the PCA transform also has serious limitations. Since it is perfectly linear, nonlinear effects are missed. For example, it is not clear that the strongest components are also the components with the most discriminating power. Moreover, non-orthogonal signal mixtures can't be resolved.

ICA

Independent Component Analysis (ICA) finds non-orthogonal axes that let the projected data components be maximally independent. Assume, for example, that two independent audio signals $x_1(t)$ and $x_2(t)$ are mixed by a matrix A resulting in the signals $s_1(t)$ and $s_2(t)$,

$$\mathbf{s}(t) = \mathbf{A}\ \mathbf{x}(t) \quad . \tag{4.14}$$

Given $\mathbf{s}(t)$, the ICA algorithm finds the unmixing matrix $\mathbf{W} = \mathbf{A}^{-1}$. The components of $\hat{\mathbf{x}} = \mathbf{W}\mathbf{s}$ will be maximally independent with respect to each other. Unlike PCA, ICA is not restricted to orthogonal axes. It takes into account higher-order moments of the data distribution in order to achieve independence of the output. The price for this is an iterative gradient search that maximizes the joint entropy of the output [BS95].

Figure 4-3 illustrates how ICA picks the axes of the transform and how it relates to PCA. Unlike PCA, it does not preserve the scaling. Hence the selection of a suitable subspace requires a more sophisticated methodology. Fig. 4-7 shows ICA transformed MEG brain data, bringing out features such as the heart beat and the alpha-brain waves.

Wavelet packets

Wavelets and filter banks are used for a wide range of engineering applications such as audio coding [KMG91, JJS93], image compression [Wat95], and classification of one and two-dimensional signals [Sai94]. We use *wavelet packets* to compute the optimal orthonormal representation of a time series with respect to some discriminant measure.

Orthonormal wavelet packets expand signals in a time-frequency grid [CS94, Sai94]. The transform is based on the repeated application of a quadrature mirror filter to each of the sub-bands followed by a down-sampling step (fig. 4-4). The number of possible filters is endless but includes Daubechies wavelets of various orders [Dau88], Coiflet wavelets and the cosine transform. After each filtering step, the blocks of coefficients describe the time domain signal in a more refined frequency band. Fig. 4-5 shows the first 194 bins of a wavelet packet in time and frequency. The signal corresponds to the difference energy between two simulated brain responses. Repeated application of the filter causes the discriminating energy to concentrate in fewer and fewer coefficients.

A key problem is the optimal selection of coefficients from the time-frequency grid. Different measures regarding the discriminating power can be applied. One approach is to choose the coefficients that maximize the average square distance D_{SD} between the signal classes:

$$D_{SD} = (\bar{w}_{i1} - \bar{w}_{i2})^2 / (\sigma_{w_{i1}} \sigma_{w_{i2}}); \tag{4.15}$$

where $\bar{w}_{ic}$ denotes the coefficient i of class c signals, and $\sigma_{w_{ic}}$ is the standard deviation of coefficients w_{ic}.

A second approach is to select the maximal entropy subspace of coefficients. The discriminating power of the squared and normalized coefficients is evaluated in terms of the symmetrized relative entropy (Kullback-Leibler distance, D_{KL}) between signals of different label:

$$D_{KL} = \sum_i \bar{w}_{i1} \log \frac{\bar{w}_{i1}}{\bar{w}_{i2}} + \bar{w}_{i2} \log \frac{\bar{w}_{i2}}{\bar{w}_{i1}} \tag{4.16}$$

From the orthonormal coefficients $\bar{w}_i$ those that maximize D_{KL} are chosen ([CW92]).

4.2 Cluster-weighted classification

Cluster-weighted classification (CWC) is a framework to approximate the measurement model $p(\mathbf{x} \mid H = H_i)$. While mixture density estimators typically use separate parameterizations for different hypotheses H_i, CWC estimates a single parameterization that takes H_i as an argument.

The density expands into

$$p(y, \mathbf{x}) = \sum_{k=1}^{K} p(y \mid c_k)\ p(\mathbf{x} \mid c_k)\ p(c_k) \tag{4.17}$$

where $p(y \mid c_k) = p(H = H_i \mid c_k)$ is an entry into a probability table and $p(\mathbf{x} \mid c_k)$ is defined in expression 3.6.

The EM algorithm of CWC simplifies to evaluating

$$p(c_k|y, \mathbf{x}) = \frac{p(y|c_k)p(\mathbf{x}|c_k)p(c_k)}{\sum_{l=1}^{K} p(y|c_l)p(\mathbf{x}|c_l)p(c_l)} \tag{4.18}$$

in the E-step, and

$$\begin{aligned}
p(c_k) &= \frac{1}{N}\sum_{n=1}^{N} p(c_k|y_n, \mathbf{x}_n) \\
\mathbf{m}_k &= \frac{\sum_{n=1}^{N} \mathbf{x}_n\ p(c_k|y_n, \mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|y_n, \mathbf{x}_n)} \\
[\mathbf{P}_k]_{ij} &= \frac{\sum_{n=1}^{N} (x_{i,n} - m_{i,k})(x_{j,n} - m_{j,k})\ p(c_k|y_n, \mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|y_n, \mathbf{x}_n)} \\
p(y = y_i \mid c_k) &= \frac{\sum_{n|y_n=y_i} p(c_k|y_n, \mathbf{x}_n)}{\sum_{n=1}^{N} p(c_k|y_n, \mathbf{x}_n)} \quad ,
\end{aligned} \tag{4.19}$$

in the M-step (section 3.2).

Figures 4-8, 4-11, and 4-10 illustrate different data sets in the feature space.

4.2.1 Simplifications

Practical implementations of CWC may require a simplified software version in order to accommodate design constraints such as limited hardware, processing speed, and memory. CWC is simple enough to be implemented on a low-end microprocessor. However, it can be simplified to reduce memory and processing needs thereby gracefully reducing the performance requirements of the algorithms. Since most constraints will be due to the device that does the classification rather than the model estimation, the latter can still use a powerful computer.

1. A straightforward simplification assigns fixed values to cluster-weights and variances. The data density then expands into

$$p(y, \mathbf{x}) = \sum_{k=1}^{K} p(y \mid c_k)\ \frac{1}{K}\ \exp\left\{\sum_{d=1}^{D} \frac{1}{\beta}(x_{d,n} - m_{d,k})^2\right\} \quad . \tag{4.20}$$

   There is no normalization factor, since only relative probabilities are of interest. β is estimated from the data and scales the Gaussians appropriately.

2. CWC evaluates the posterior likelihood of a data point and one cluster relative to all

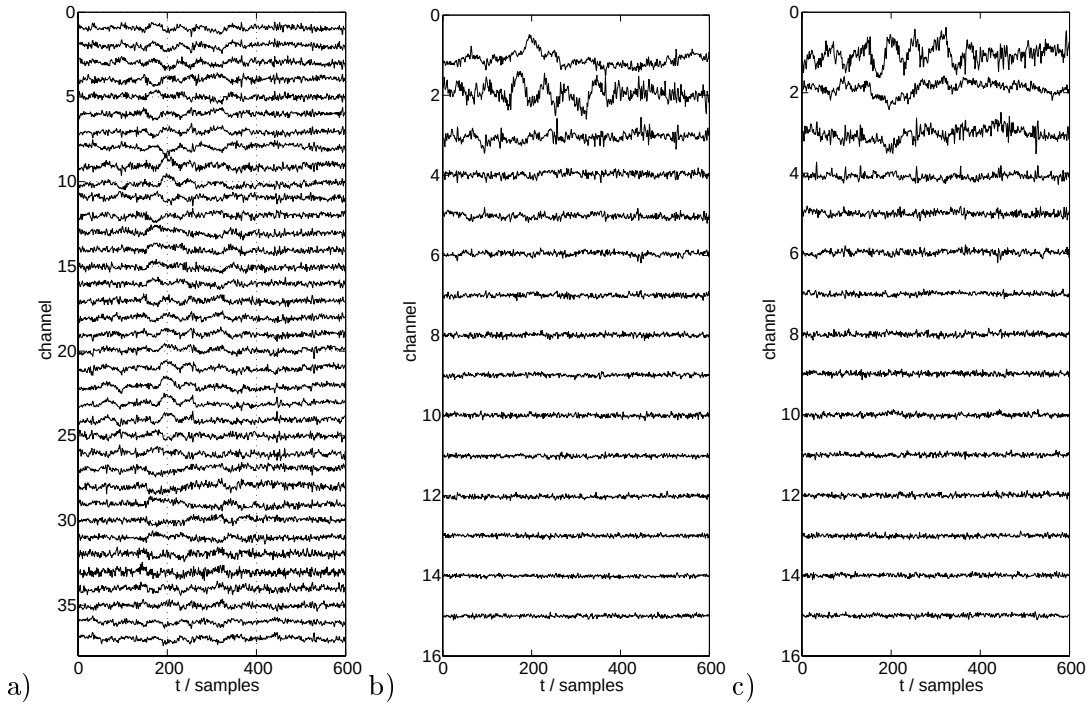


Figure 4-6: MEG data. a) All channels of one raw epoch. b) The data run through a PCA transform (the PCA was defined on the average of all epochs) c) The data run through a PCA transform (the PCA was defined on single epochs).

the others. Since our feature space is Euclidean we can approximate the posteriors by measuring the distance between data and clusters. We force the clusters to have equal weight during estimation and evaluate the scaled square distance between data n and cluster k

$$D_{c_k,\mathbf{x}_n} = \sum_{d=1}^{D} \frac{(x_{d,n} - m_{d,k})^2}{\sigma_{d,k}^2} \tag{4.21}$$

A point is classified according to the class dominating the cluster that is closest to it in terms of $D_{c_k,\mathbf{x}_n}$. This simplification eliminates the evaluation of exponential functions.

3. Further simplification eliminates $\sigma_{d,k}^2$.

$$D_{c_k,\mathbf{x}_n} = \sum_{d=1}^{D} (x_{d,n} - m_{d,k})^2 \tag{4.22}$$

The discriminating criterion 4.22 is simply the square-distance between points and clusters. Points are strictly classified according to the cluster they are closest to. This means we are back in *K-means* world.

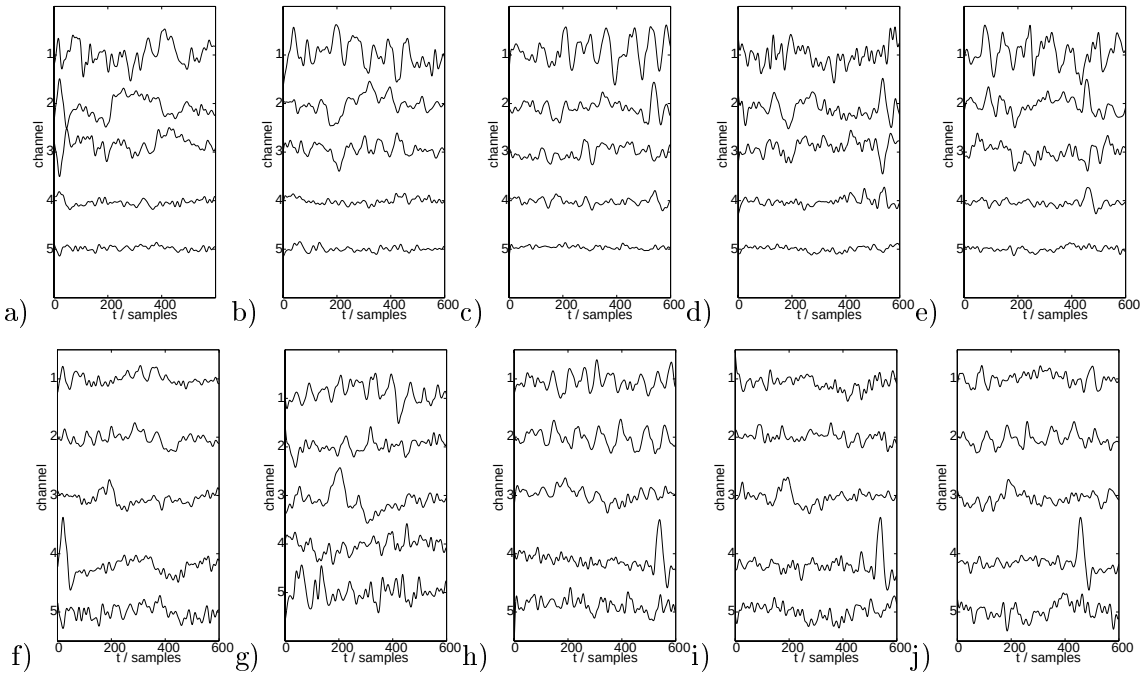


Figure 4-7: Recorded MEG epochs stimulated by /dæ/,/pæ/,/tæ/,/bæ/ and /pæ/. a-e) PCA transformed responses. f-j) Same epochs ICA transformed as suggested in Makeig et al. 1996 and 1997. Some events come out clearly, such as the heart beat in channel 4 and the stimulus response in channel 3; however, the noise level is fairly high.

4.3 Applications

4.3.1 Stimulus detection from MEG brain data

Data

Magnetoencephalography (MEG) uses SQUID technology to measure the small magnetic fields induced by electrical activity in the brain. Sensitive to roughly the same neural activity as EEG/ERP, MEG offers some advantages in data analysis and source localization. Multi-sensor MEG systems recording magnetic flux at kilohertz sampling rates provide an incredibly rich source of data about brain activity. We try to determine the type of audio stimulus applied to the subject given the MEG-recorded brain response of the subject [LSM00].

The data was collected as part of the experiment reported in [PYP+96]. Briefly, the stimuli were 4 synthesized 300 ms syllables, /bæ/, /pæ/, /dæ/, and /tæ/. The voiced-voiceless pairs /bæ/-/pæ/ and /dæ/-/tæ/ differ acoustically only in "voicing onset time," with the first member of each pair containing 20 ms of "aspiration" prior to the onset of the voiced portion of the syllable, and the second member containing 80ms of aspiration. The SQUID-based sensors recorded 37 channels of data over 600 ms triggered 100 ms after

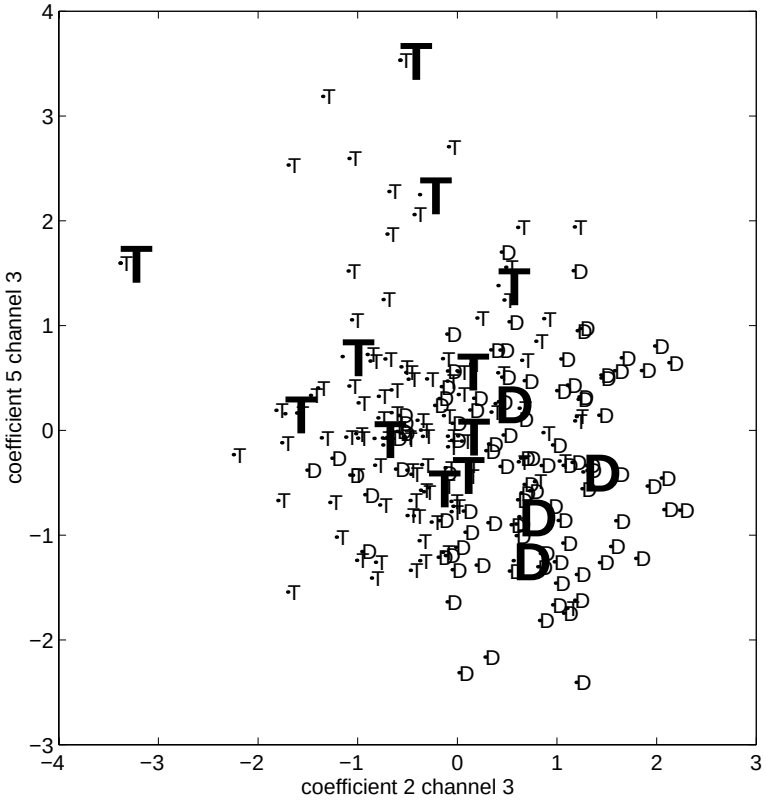


Figure 4-8: Two dimensions of the feature vector for the tæ/dæ discrimination: The small letters refer to the actual sample points; the large letters are the centers of the local experts. The letter T refers to the voiceless version of the consonant, and D to the voiced version.

the stimulus onset. One hundred epochs of each stimuli were recorded (fig. 4-6).

Results

The analysis of the MEG data proceeds in three steps. First we apply a PCA transform to reduce the dimensionality of the data from 37 to the order of three to four channels (section 4.1.2). Fig. 4-7 illustrates the effect of the PCA transform in comparison to that of an ICA transform. Second we apply a wavelet packet transform to the principal components, from which we retain a 2 to 5-dimensional feature vector. We then build a CWC model on the feature vector. Fig. 4-8 shows the clustered data.

The epochs for each stimulus were randomly divided into a training set of 70 and a testing set of 30 epochs. Two different windows with different offsets were tested, both 256 samples long. The offset for the second window is beyond the acoustic difference between the stimuli, which ensures that we are detecting based on brain activity and not simply a MEG recording of the actual stimulus.

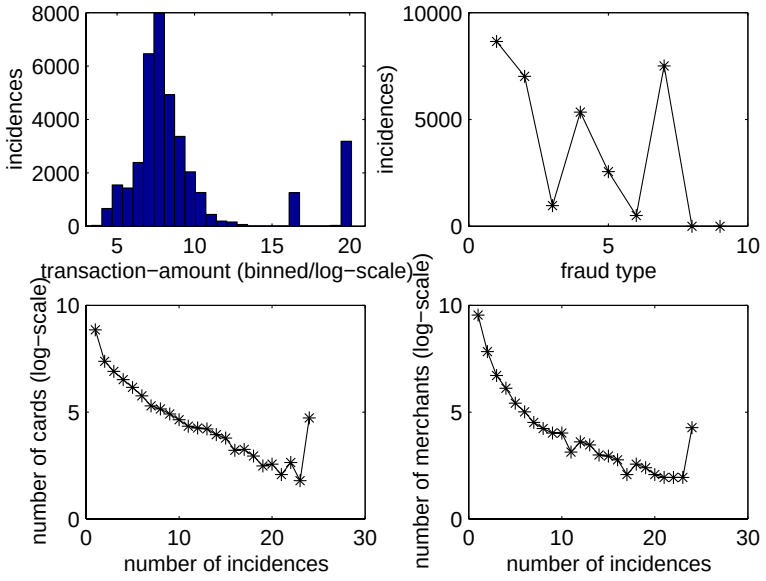


Figure 4-9: Credit fraud transaction statistics. a) Incidences per transaction amount, b) incidences per fraud type, c) number of cards over number of incidences per issued card, d) number of merchants over number of incidences.

Classification of voiced and voiceless consonants (/tæ/-/dæ/) and (/pæ/-/bæ/) epochs results in 78% of correct answers. Classification of voiced or voiceless pairs (/tæ/-/pæ/) and (/dæ/-/bæ/) was impossible with the available data [LSM00].

4.3.2 Consumer fraud characterization

Data

Credit card companies are experiencing an increase in fraudulent activity in terms of dollar value and number of incidences. In addition to this trend, there is an increase of *creativity* regarding what kinds of fraud are committed, to the extent that there are types of fraud that become *a la mode*. While there are certain types that have been popular for a long time, such as stealing cards, there are others that have almost the character of a new exciting sport. In *account takeover*, for example, the perpetrator takes on the identity and plays the role of the card owner to his or her own benefit.

Card issuers have a strong interest in early detection of new types of fraud, in order to prevent a *new idea* from causing too much damage. While the detection of fraudulent card transactions is almost a classic application for ANN-type networks, the classification of fraudulent transactions with respect to fraud categories is new.

All fraudulent transactions are reported to the credit card companies from the issuing banks with a delay of about three months. By then, the fraud has been classified in seven different categories (lost, stolen, not received, fraudulent application, counterfeit, other, mail/phone order, multiple imprint, account takeover). The quantity of data is

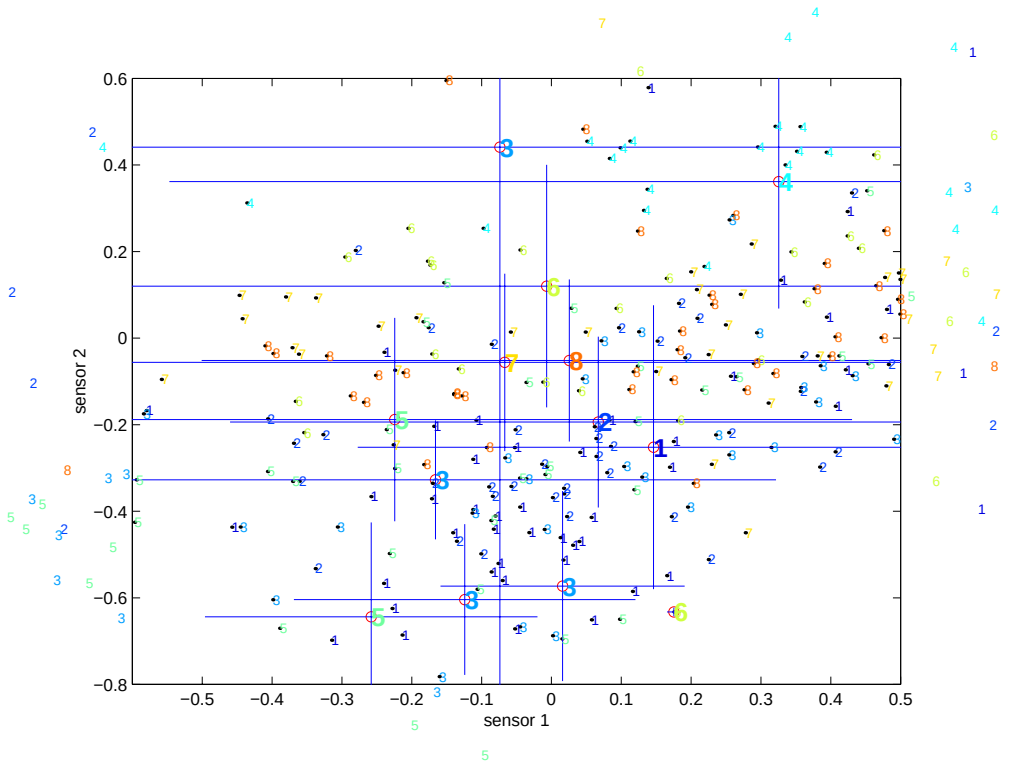


Figure 4-10: Bacterial data in a two-dimensional clustered space. Substance labels range from one to eight, sensor readings range from one to two.

enormous, and the regions are widely overlapping in feature space. It is our goal to learn how to predict the fraud type given transaction patterns such as transaction amount, the merchant type or demographic information.

Results

Continuous and discrete-valued features are constructed from the data. The log-scale transaction amount is used rather then the transaction amount. The total period of fraud on a particular card is inferred from the first and last fraud incidence reported. Merchants are categorized according to 120 different types. This number is reduced to 20 types, which covers about two thirds of the incidences, while the remaining merchants are summarized as *others*.

Table 4.2 indicates the classification results. Approximately 50% of out-of-sample test data was classified correctly. Purely random classification has the expectation $\frac{1}{7} \cdot 100\% \approx 15\%$ of correct answers. A uniform classification in terms of the strongest class would have resulted in 27.9% correct answers.

Alternatively a model is built to classify the two strongest types. In this case, 91.4% of the out-of-sample data is classified correctly. Pure chance had an expectation of 50% correct answers, and classifying according to the strongest label would have given 53.2%

Table 4.1: Electronic noses: out-of-sample discrimination results. Labeled substances are discriminated based on bacterial readings.

	1	2	3	4	5	6	7	8
1	**20**	0	0	0	0	0	0	0
2	0	**15**	0	0	1	0	0	0
3	0	0	**21**	0	0	1	0	0
4	0	0	0	**22**	0	0	0	0
5	2	0	0	0	**20**	0	0	0
6	0	0	4	0	0	**17**	0	0
7	0	0	0	0	0	0	**29**	0
8	0	0	0	0	0	0	0	**20**

164 correct classifications (95%),
8 wrong classifications.

of correct answers.

4.3.3 Electronic noses

In this example different substances are classified based on bacterial data. The data was provided by a sponsor company in the process of developing electronic noses, that is, low cost electronic devises for chemical sensing. The data set consists of eight substances, labeled 1...8, and a 32-dimensional measurement vector consisting of bacterial readings, associated with each substance.

As can be seen from the strong diagonal elements in the detection table (4.1) this task is considerably easier than the examples introduced earlier. Ninety-five percent of the out-off-sample data points were classified correctly. The winning model used a feature vector containing 22 parameters from the data set along with 25 clusters (fig 4-10).

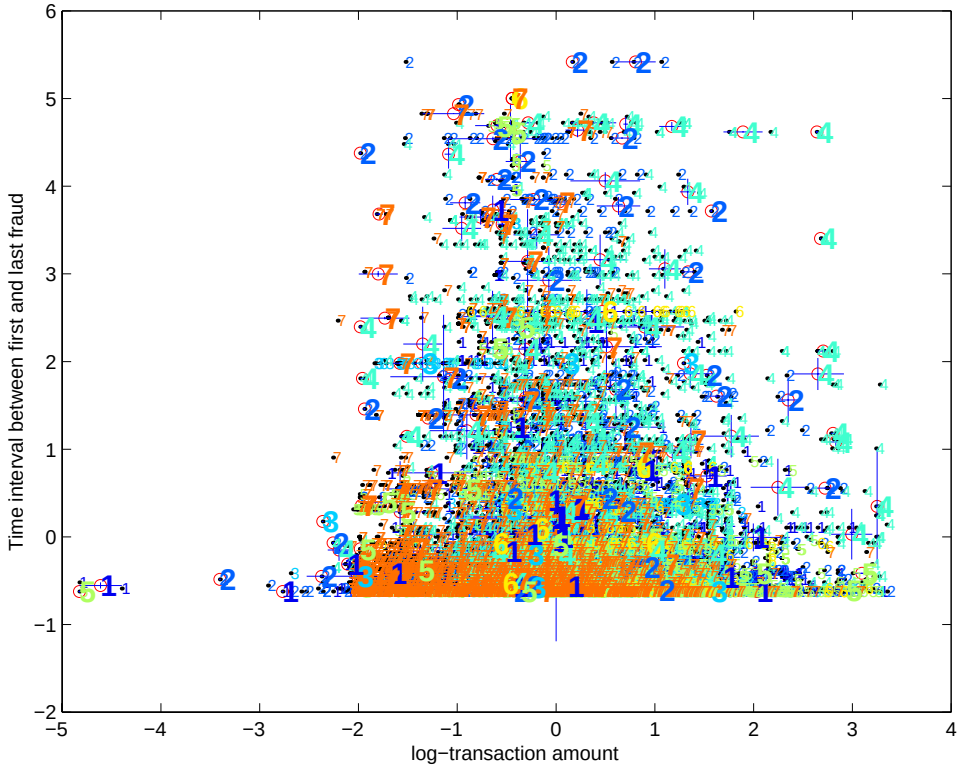


Figure 4-11: Credit card fraud data in two-dimensional clustered space. Time interval of fraudulent transactions versus log-transaction amount of singular fraud incidence.

Table 4.2: Detection results – credit card fraud discrimination. The columns indicate the true label and the rows indicate the classification. The values in the diagonal correspond to correct classifications.

a) in-sample

	lost	stolen	not rec.	fr.appli.	counterf.	other	mail/phone
lost	**1386**	778	43	396	25	15	410
stolen	760	**949**	50	418	38	14	352
not rec.	191	103	**92**	46	1	5	34
fr.appli.	450	240	33	**1526**	21	49	200
counterf.	447	337	13	216	**141**	8	198
other	47	23	2	17	7	**118**	6
mail/phone	86	54	5	125	11	9	**3497**

7709 correct classifications (55.1%), 6283 wrong classifications;

b) out-off-sample

	lost	stolen	not rec.	fr.appli.	counterf.	other	mail/phone
lost	**956**	706	29	761	39	15	395
stolen	1185	**1061**	57	715	84	5	791
not rec.	199	64	**3**	67	3	0	30
fr.appli.	556	340	11	**1318**	45	31	262
counterf.	306	214	4	131	**56**	3	132
other	17	13	3	50	7	**8**	10
mail/phone	102	38	3	264	20	10	**2873**

6275 correct classifications (44.8%), 7717 wrong classifications;

c) in-sample

	lost	mail/phone
lost	**2690**	197
mail/phone	113	**3500**

6190 correct classifications (95.2%),
310 wrong classifications;

d) out-off-sample

	lost	mail/phone
lost	**2739**	306
mail/phone	254	**3201**

5940 correct classifications (91.4%),
560 wrong classifications;

Chapter 5

Prediction and Estimation

In the context of time series analysis *prediction* refers to the effort of forecasting future values of a time series, given past observations of the time series. Usually a single observable is measured. The time series may originate from an autonomous system, such as the laser from figure 5-1, or from a driven system, such as the violin. It may be deterministic or stochastic and predictable or chaotic (Lorenz set, fig. 2-5). In a generalized approach to prediction one forecasts statistical characteristics, that is moments of the time series. This takes into consideration that often only statistical statements can be made and that it is important to be clear about the uncertainty of a prediction. A typical application is the estimation of volatility, that is variance, of a financial time series for the purpose of option pricing.

Gershenfeld and Weigend [GW93] also distinguish between *learning* and *understanding* of observations. While understanding refers to the mathematical insights into the governing equations of the system, learning refers to the process of emulating the structure of a time series given a mathematical approximation framework. The effort in this chapter relates to both aspects of modeling and forecasting a time series. We try to include as much insight (priors) about the nature of a data set in the modeling, but always train the model based on a set of measurements.

5.1 Background

5.1.1 Estimation Theory

The prediction of unknown and/or future values $\mathbf{y}$ based on observations of a vector $\mathbf{x}$ requires the solution of two estimation problems: estimation of the model parameters $\mathbf{a}$ and estimation of $\mathbf{y}$ given $\mathbf{x}$, the actual prediction problem.

In the random approach to parameter estimation, the conditional density $p(\mathbf{x}|\mathbf{y})$, where $\mathbf{y}$ is the target quantity and $\mathbf{x}$ is a measured quantity, fully characterizes the relationship between $\mathbf{x}$ and $\mathbf{y}$. We choose to analyze this relationship from a Bayesian point of view. We refer to the density $p(\mathbf{y})$ as the prior density which, independently from any measurement, let's us express prior beliefs about the quantity to be estimated. We also assume a measurement model characterized by the conditional density $p(\mathbf{x}|\mathbf{y})$. This density indicates the probability that a certain data $\mathbf{x}$ is generated by the system state characterized

by $\mathbf{y}$.

Given $p(\mathbf{y})$ and $p(\mathbf{x}|\mathbf{y})$ we find the posterior density for $\mathbf{x}$

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})\ p(\mathbf{y})}{p(\mathbf{x})} \tag{5.1}$$

We also introduce a cost function $C(\mathbf{y}, \hat{\mathbf{y}})$, which defines the cost associated with estimating a given $\mathbf{y}$ as $\hat{\mathbf{y}}$. From this and expression (5.1) we derive the estimator $\hat{\mathbf{y}}$,

$$\hat{\mathbf{y}}(\cdot) = \text{argmin}_{\mathbf{f}(\cdot)}\ E[C(\mathbf{y}), \mathbf{f}(\mathbf{x})] \tag{5.2}$$

with

$$\begin{aligned} E[C(\mathbf{y}, \mathbf{f}(\mathbf{x}))] &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} C(\mathbf{y}, \mathbf{f}(\mathbf{x}))\ p(\mathbf{y}, \mathbf{x}) d\mathbf{y} d\mathbf{x} \\ &= \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} C(\mathbf{y}, \mathbf{f}(\mathbf{x})) p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \right]\ p(\mathbf{x}) d\mathbf{x} \end{aligned} \tag{5.3}$$

Since $p(\mathbf{x}) \geq 0$ equation(5.3) simplifies and the final estimator for a particular measurement $\mathbf{x}$ is

$$\hat{\mathbf{y}}(\mathbf{x}) = \text{argmin}_{\text{a}} \int_{-\infty}^{+\infty} \text{C}(\mathbf{y}, \text{a}) \text{p}(\mathbf{y}|\mathbf{x}) \text{d}\mathbf{y} \tag{5.4}$$

Depending on the cost function C, different estimators are constructed. The *minimum absolute-error estimator* (MAE) results from the cost function $C(a, \hat{a}) = |a - \hat{a}|$;

$$\hat{y}_{\text{MAE}} = \text{argmin}_{\text{a}} \int_{-\infty}^{+\infty} |\text{y} - \text{a}| \text{p}(\text{y}|\mathbf{x}) \text{dy} \tag{5.5}$$

Rewriting this expression and differentiating with respect to $\mathbf{a}$ yields [WWS96],

$$\begin{aligned} \hat{y}_{\text{MAE}} &= \text{argmin}_a \left\{ \int_{-\infty}^{a} (a - y) p(y|\mathbf{x}) dy + \int_{a}^{+\infty} (y - a) p(y|\mathbf{x}) dy \right\} \\ 0 &= \left\{ \int_{-\infty}^{a} p(y|\mathbf{x}) dy + \int_{a}^{+\infty} p(y|\mathbf{x}) dy \right\} \Big|_{a = \hat{\mathbf{y}}_{\text{MAE}}} \quad , \end{aligned} \tag{5.6}$$

which shows that the MAE finds the median of the posterior probability for the estimator of y

$$\int_{-\infty}^{\hat{y}_{MAE}(\mathbf{x})} p(y|\mathbf{x}) = \int_{\hat{y}_{MAE}(\mathbf{x})}^{+\infty} p(y|\mathbf{x}) = \frac{1}{2} \quad . \tag{5.7}$$

The *Maximum A Posteriori* (MAP) estimator uses the peak (mode) of the posterior density as the estimator. It is derived from the *Minimum Uniform Cost* (MUC) estimator with the cost function

$$C(a, \hat{a}) = \begin{cases} 1 & \text{for } |\text{a} - \hat{\text{a}}| > \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

The MUC uniformly penalizes any error beyond the threshold ε. For $\varepsilon \to 0$ the MUC

estimator turns into the MAP.

$$\hat{y}_{\mathrm{MAP}}(\mathbf{x}) = \mathrm{argmax_a}\ \mathrm{p(a|\mathbf{x})} = \lim_{\varepsilon \to 0} \hat{\mathrm{y}}_{\mathrm{MUC}}(\mathbf{x}) \tag{5.9}$$

The mean-square error (MSE) cost criterion

$$C(a, \hat{a}) = (\mathbf{a} - \hat{\mathbf{a}})^T (\mathbf{a} - \hat{\mathbf{a}}) = \sum_{n=1}^{N} (a_i - \hat{a}_i)^2 \tag{5.10}$$

yields the Bayes' Least-Square (BLS) estimator

$$\hat{\mathbf{y}}_{BLS}(\mathbf{x}) = \mathrm{argmin_a} \int_{-\infty}^{+\infty} (\mathbf{y} - \hat{\mathbf{a}})^{\mathrm{T}} (\mathbf{y} - \hat{\mathbf{a}})\ \mathrm{p}(\mathbf{y}|\mathbf{x}) \mathrm{d}\mathbf{y} \tag{5.11}$$

Differentiating and rearranging of terms yields

$$\frac{\partial}{\partial \mathbf{a}} \left[\int_{-\infty}^{+\infty} (\mathbf{y} - \hat{\mathbf{a}})^T (\mathbf{y} - \hat{\mathbf{a}})\ p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \right] = -2 \int_{-\infty}^{+\infty} (\mathbf{y} - \hat{\mathbf{a}})\ p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \tag{5.12}$$

and

$$\begin{aligned} 0 &= \int_{-\infty}^{+\infty} (\mathbf{y} - \hat{\mathbf{a}})\ p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \Big|_{a=\hat{\mathbf{x}}_{\mathrm{BLS}}} \\ &= \int_{-\infty}^{+\infty} \mathbf{y}\ p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \Big|_{a=\hat{\mathbf{x}}_{\mathrm{BLS}}} - \int_{-\infty}^{+\infty} \hat{\mathbf{y}}_{\mathrm{BLS}}(\mathbf{x})\ p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \Big|_{a=\hat{\mathbf{x}}_{\mathrm{BLS}}} \\ &= E[\mathbf{y} \mid \mathbf{x}] - \hat{\mathbf{y}}_{\mathrm{BLS}}(\mathbf{x}) \int_{-\infty}^{+\infty} p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \\ &= E[\mathbf{y} \mid \mathbf{x}] - \hat{\mathbf{y}}_{\mathrm{BLS}}(\mathbf{x}) \end{aligned} \tag{5.13}$$

Hence we see that the BLS estimator is the mean of the posterior density $p(\mathbf{x}|\mathbf{y})$.

$$\hat{\mathbf{y}}_{\mathrm{BLS}}(\mathbf{x}) = E[\mathbf{y} \mid \mathbf{x}] \quad . \tag{5.14}$$

The BLS estimator has a number of important properties: it is unbiased and its estimation error $e(\mathbf{y}, \mathbf{x}) = \hat{\mathbf{y}}(\mathbf{x}) - \mathbf{y}$ is orthogonal to any function of the data

$$E[\hat{\mathbf{y}}(\mathbf{x}) - \mathbf{y}) \mathbf{g}^T(\mathbf{x})] = 0 \tag{5.15}$$

Moreover, the uncertainty of the BLS estimator is lower than that of any other estimator. If Λ_e denotes the error covariance of any estimator $\hat{\mathbf{y}}(\cdot)$, Λ_{BLS} satisfies $\Lambda_{BLS} \leq \Lambda_e$. CWM uses the BLS estimator to extract the predicted $\mathbf{y}$ from the posterior density function.

If the estimator is constrained to be a linear function of the data, the BLS estimator turns into the Linear Least-Squares (LLS) estimator.

$$\begin{aligned} \hat{\mathbf{y}}_{\mathrm{LLS}}(\cdot) &= \mathrm{argmin}_{\mathbf{f}(\mathbf{x})} E[||\mathbf{y} - \mathbf{f}(\mathbf{x})||^2] \\ \mathbf{f}(\mathbf{x}) &= \mathbf{A} \cdot \mathbf{x} + \mathbf{a}_0 \end{aligned} \tag{5.16}$$

If $\mathbf{y}$ and $\mathbf{x}$ are jointly Gaussian we observe that $\hat{\mathbf{y}}_{\mathrm{BLS}}(\mathbf{x}) = \hat{\mathbf{y}}_{\mathrm{LLS}}(\mathbf{x}) = \hat{\mathbf{y}}_{\mathrm{MAP}}(\mathbf{x}) =$

$\hat{\mathbf{y}}_{\mathrm{MAE}}(\mathbf{x})$.

5.1.2 Linear models

In 1927, Yule invented what was later named an auto-regressive moving-average (**ARMA**) model trying to predict the annual number of sunspots. The **MA** part in the acronym refers to the idea that past observables of a system indicate its present state and its future output (section 2.2.1). The output of a driven system can be approximated as the weighted sum of its delayed inputs $e(t)$:

$$x(t) = \sum_{n=0}^{N} a_n \cdot e(t-n) \tag{5.17}$$

While statisticians call this a moving-average model, it really is the implementation of a convolution filter and engineers refer to it as a *finite impulse response* filter (FIR). The many names indicate the importance of this technique in a number of different fields.

Model 5.17 can be characterized in three domains [WG93, P.12]. In the time domain, we generate the *impulse response* of the system by choosing an input signal that is non-zero for $t = 0$ only,

$$e(t) = \delta(t) \quad , \tag{5.18}$$

in which case $x(t)$ equals the series of coefficients

$$x(n) = a_n \quad . \tag{5.19}$$

Alternatively, we can infer the spectral description of the system in the frequency domain. Since the convolution in the time domain equals a multiplication in the frequency domain, and since the power spectrum of an impulse $\delta(t)$ is flat [OS89], the *transfer function* of the system equals the Fourier transform of the impulse response [BJ76]

$$b_f = \sum_{k=0}^{K} a_k e^{-i2\pi kf} \quad , \tag{5.20}$$

the power spectrum of which is

$$c_f = |1 + b_1 e^{-i2\pi f} + b_2 e^{-i2\pi 2f} + ... + b_K e^{-i2\pi Kf}|^2 \quad . \tag{5.21}$$

The information contained in the power spectrum can alternatively be represented in terms of the *auto-correlation coefficients* of a signal,

$$\rho_n = \frac{\langle (x_t - m_x)(x_{t-n} - m_x) \rangle}{\langle (x_t - m_x)(x_t - m_x) \rangle} \tag{5.22}$$

where $\langle \cdot \rangle$ is the expectation operator and μ_x is the mean of the time series. Likewise the cross-correlation coefficients between time series $x(t)$ and $e(t)$ are defined as

$$\nu_n = \frac{\langle (x_t - m_x)(e_{t-n} - m_e) \rangle}{\langle (x_t - m_x)(e_t - m_e) \rangle} \quad . \tag{5.23}$$

Before we use the auto-correlation and cross-correlation coefficients to estimate a linear model, let's introduce the auto-regressive (AR) part as well. Auto-regressive models feed back the output into the input. Once excited, they can keep running forever, which is why the engineering literature calls them *infinite impulse response* (IIR) filters.

$$x(t) = \sum_{n=1}^{N} a_n \cdot x(t-n) + e(t) \tag{5.24}$$

$e(t)$ is the system input.

If we combine the MA and AR model we obtain the ARMA model:

$$x(t) = \sum_{m=1}^{M} a_m \cdot x(t-m) + \sum_{n=1}^{N} b_n \cdot e(t-n) \tag{5.25}$$

Model 5.25 is best described in terms of the z-transform, designed to represent a discrete system in the spectral domain [OS89]:

$$\begin{aligned} X(z) &\equiv \sum_{n=-\infty}^{n=+\infty} x_n z^n \\ X(z) &= A(z)X(z) + B(z)E(z) \\ &= \frac{B(z)}{1-A(z)} E(z) \end{aligned} \tag{5.26}$$

5.1.3 Linear Regression

From a model estimation point of view, ARMA models fall into the category of linear regression models. $x(t)$ is regressed on some input $e(t-n)$, on itself ($x(t-m)$), or on both. Assuming a Gaussian error model, the optimal estimator minimizes the mean square error between the linear predictor and the data set. Given a set of data $g = (\mathbf{x}, y) \in \mathcal{R}^d \times \mathcal{R}$, we define the cost function

$$\begin{aligned} H &= \frac{1}{N} \sum_{n=1}^{N} \left[f(\mathbf{x}_n) - y_n \right]^2 \\ f(\mathbf{x}) &= \sum_{d=0}^{D} a_d \cdot x_d \quad , \end{aligned} \tag{5.27}$$

with $x_0 = 1$ (the constant term).

As in equation 3.23 we derive the coefficients $\mathbf{a}$

$$\begin{aligned} \nabla_{\mathbf{a}} H &= \frac{1}{N} \sum_{n=1}^{N} 2 \left[f(\mathbf{x}_n) - y_n \right] \frac{\partial}{\partial \mathbf{a}} f(\mathbf{x}_n) \\ &= \frac{1}{N} \sum_{n=1}^{N} \left[f(\mathbf{x}_n) - y_n \right] \mathbf{x} \\ &= \mathbf{0} \end{aligned}$$

which yields

$$\begin{aligned} \mathbf{a} &= \mathbf{B}^{-1} \cdot \mathbf{c} && (5.28) \\ \mathbf{B} &= \frac{1}{N} \begin{pmatrix} \sum_n x_{0,n}x_{0,n} & \dots & \sum_n x_{0,n}x_{D,n} \\ \dots & \dots & \dots \\ \sum_n x_{D,n}x_{0,n} & \dots & \sum_n x_{D,n}x_{D,n} \end{pmatrix} && (5.29) \\ \mathbf{c} &= \frac{1}{N} \begin{pmatrix} \sum_n y_n x_{0,n} \\ \dots \\ \sum_n y_n x_{D,n} \end{pmatrix} \end{aligned}$$

In (5.29), we recognize the covariance matrix of the data and the cross-correlation matrix between the input x_i and the output y. If $x_i(t) = y(t-i)$, as in (5.25), and if y is zero mean ($E[y] = 0$), expression (5.29) also equals the estimator for the auto-correlation coefficients (5.22).

If y is vector-valued, the full matrix of coefficients is

$$\begin{aligned} \mathbf{A} &= \mathbf{B}^{-1} \cdot \mathbf{C} \\ \mathbf{C} &= \frac{1}{N} \begin{pmatrix} \sum_n y_{0,n}x_{0,n} & \dots & \sum_n y_{0,n}x_{D,n} \\ \dots & \dots & \dots \\ \sum_n y_{D_y,n}x_{0,n} & \dots & \sum_n y_{D_y,n}x_{D,n} \end{pmatrix} && (5.30) \end{aligned}$$

ARMA models are good approximations and predictors for some processes. However, they break for all systems for which the power spectrum is not a useful characterization [GW93]. Even a system with a very simple nonlinearity, such as the logistic map, causes the ARMA model to fail terribly.

5.1.4 Generalized linear and polynomial models

A natural extension of linear models into the nonlinear domain are generalized linear models, that is models that consist of weighted nonlinear basis functions (section 3.1, equ. 3.1) as opposed to simple linear terms.

$$f(x) = \sum_{i=1}^{I} a_i \Psi_i(\mathbf{x}) \tag{5.31}$$

where the Ψ_k are arbitrary nonlinear functions.

Fitting the coefficients is analogous to the strictly linear case replacing the linear basis functions with the new basis functions.

$$\begin{aligned} \mathbf{A} &= \mathbf{B}^{-1} \cdot \mathbf{C} && (5.32) \\ \mathbf{B} &= \frac{1}{N} \begin{pmatrix} \sum_n \Psi_0(\mathbf{x}_n)\Psi_0(\mathbf{x}_n) & \dots & \sum_n \Psi_0(\mathbf{x}_n)\Psi_K(\mathbf{x}_n) \\ \dots & \dots & \dots \\ \sum_n \Psi_K(\mathbf{x}_n)\Psi_0(\mathbf{x}_n) & \dots & \sum_n \Psi_K(\mathbf{x}_n)\Psi_K(\mathbf{x}_n) \end{pmatrix} \end{aligned}$$

$$\mathbf{C} = \frac{1}{N}\begin{pmatrix} \sum_n y_{0,n}\Psi_0(\mathbf{x}_n) & ... & \sum_n y_{D_y,n}\Psi_0(\mathbf{x}_n) \\ ... & ... & ... \\ \sum_n y_{0,n}\Psi_K(\mathbf{x}_n) & ... & \sum_n y_{D_y,n}\Psi_K(\mathbf{x}_n) \end{pmatrix} \tag{5.33}$$

The most popular generalized linear models are polynomial models:

$$\Psi_i(\mathbf{x}) = x_1^{e_{i1}} \cdot {x_2}^{e_{i2}} \cdot ... x_d^{e_{id}} \tag{5.34}$$

Given the polynomial order O, we keep any combination of integer values for e that satisfy $e_{i1} + e_{i2} + ...e_{id} \leq O$ for all i. This includes the constant term $e_{0,d}$ for which $e_{ki} = 0$. The number of basis terms I then equals [Met96, P.83]

$$I(D, O) = \begin{pmatrix} (D+O) \\ D \end{pmatrix} = \frac{(D+O)!}{D! \cdot O!} \quad . \tag{5.35}$$

Polynomial expansions are orthogonal and complete for $O \to \infty$. The matrix $\mathbf{B}$ for polynomial models is known as the Vandermond matrix. Assuming a two dimensional vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and a second order approximation it is

$$\mathbf{B} = \frac{1}{N} \cdot \tag{5.36}$$

$$\begin{pmatrix} \sum_n 1 & \sum_n x_{1,n} & \sum_n x_{2,n} & \sum_n x_{1,n}x_{2,n} & \sum_n x_{1,n}^2 & \sum_n x_{2,n}^2 \\ \sum_n x_{1,n} & \sum_n x_{1,n}^2 & \sum_n x_{1,n}x_{2,n} & \sum_n x_{1,n}^2 x_{2,n} & \sum_n x_{1,n}^3 & \sum_n x_{1,n}x_{2,n}^2 \\ \sum_n x_{2,n} & \sum_n x_{1,n}x_{2,n} & \sum_n x_{2,n}^2 & \sum_n x_{1,n}x_{2,n}^2 & \sum_n x_{1,n}^2 x_{2,n} & \sum_n x_{2,n}^3 \\ \sum_n x_{1,n}x_{2,n} & \sum_n x_{1,n}^2 x_{2,n} & \sum_n x_{1,n}x_{2,n}^2 & \sum_n x_{1,n}^2 x_{2,n}^2 & \sum_n x_{1,n}^3 x_{2,n} & \sum_n x_{1,n}x_{2,n}^3 \\ \sum_n x_{1,n}^2 & \sum_n x_{1,n}^3 & \sum_n x_{1,n}^2 x_{2,n} & \sum_n x_{1,n}^3 x_{2,n} & \sum_n x_{1,n}^4 & \sum_n x_{1,n}^2 x_{2,n}^2 \\ \sum_n x_{2,n}^2 & \sum_n x_{1,n}x_{2,n}^2 & \sum_n x_{2,n}^3 & \sum_n x_{1,n}x_{2,n}^3 & \sum_n x_{1,n}^2 x_{2,n}^2 & \sum_n x_{2,n}^4 \end{pmatrix}$$

As was mentioned earlier, nonlinear models need to balance between under and overfitting [Ger99a]. Models should abstract from noisy data, e.g. model the true data and discard the noise. We can enforce this constraint using a regularizing term in the cost function. In general, a regularizer expresses prior beliefs about what a good model should look like. Here we propose a smoothness prior that penalizes strong first and second moments.

Since it simplifies equations significantly we re-normalize the data to fit the finite support $\mathcal{R}^D \cap [0, 1]$ and add the integral over the second derivative (curvature) to the cost function.

$$\begin{aligned} H_2 &= \int_{x_1=0}^{1} ... \int_{x_D=0}^{1} (\frac{\partial^2 f}{\partial \mathbf{x}^2})^2 dx_1 ... dx_D \\ &= \int_{x_1=0}^{1} ... \int_{x_D=0}^{1} \sum_{i=1}^{D} (\frac{\partial^2 f}{\partial x_i^2})^2 + 2 \cdot \sum_{i=1}^{D} \sum_{j=1}^{j-1} (\frac{\partial^2 f}{\partial x_i \partial x_j})^2 dx_1 ... dx_D \end{aligned} \tag{5.37}$$

Using λ to balance the variational terms the complete expression is

$$\begin{aligned} H(f) &= H_2 + \lambda \cdot H_1 \\ &= \int_{x_1=0}^{1} ... \int_{x_D=0}^{1} (\frac{\partial^2 f}{\partial \mathbf{x}^2})^2 dx_1 ... dx_D + \lambda \cdot \frac{1}{N} \sum_i N[f(\mathbf{x}_i) - y_i]^2 \end{aligned} \tag{5.38}$$

We find the optimal coefficients by taking the derivative with respect to the coefficients $\mathbf{a}$ and setting the expression to zero,

$$\begin{aligned} \frac{\partial H}{\partial \mathbf{a}} &= \frac{\partial H_1}{\partial \mathbf{a}} + \lambda \cdot \frac{\partial H_2}{\partial \mathbf{a}} \\ &= \mathbf{D} \cdot \mathbf{a} + \lambda \cdot (\mathbf{B} \cdot \mathbf{a} - \mathbf{c}) = 0 \quad . \end{aligned} \tag{5.39}$$

We then resolve for the vector of coefficents

$$\mathbf{a} = \lambda (\mathbf{D} + \lambda \mathbf{B})^{-1} \cdot \mathbf{c} \quad . \tag{5.40}$$

The optimal value for λ is determined by cross-validation with a test-set.

5.2 Cluster-weighted estimation

Cluster-weighted estimation and prediction uses simple local estimators, e.g. LLS estimators, and superimposes them to form globally powerful nonlinear estimators. The linear and generalized-linear models that were introduced in the past sections are now integrated in the cluster-weighted framework. As usual, we start with a set of measurements $\{\mathbf{y}, \mathbf{x}\}_{n=1}^N$. We assume that the data points have been generated by some unknown function $\Psi : \mathcal{R}^x \rightarrow \mathcal{R}^y$, and that the data is corrupted by noise. We expand the density $p(\mathbf{y}, \mathbf{x})$ around the local models. From this we infer the posterior density $p(\mathbf{y}|\mathbf{x})$. The BLS estimator $E[p(\mathbf{y}|\mathbf{x} = \mathbf{X})]$ is used to approximate the deterministic function Ψ.

The EM training procedure and the E and M updates have been explained in detail in Section 3.2. Here we give the update equations for the local models only (3.25). In the case of local linear models as in (5.27) the matrices $\mathbf{B}_k$ and $\mathbf{C}_k$ from equation (3.25) are

$$\begin{aligned} \mathbf{B}_k &= \begin{pmatrix} \langle x_0 x_0 \rangle & ... & \langle x_0 x_D \rangle \\ ... & ... & ... \\ \langle x_D x_0 \rangle & ... & \langle x_D x_D \rangle \end{pmatrix} \\ \mathbf{C}_k &= \begin{pmatrix} \langle y_0 x_0 \rangle & ... & \langle y_0 x_D \rangle \\ ... & ... & ... \\ \langle y_{D_y} x_0 \rangle & ... & \langle y_{D_y} x_D \rangle \end{pmatrix} \end{aligned} \tag{5.41}$$

with

$$\langle \theta \rangle = \frac{\sum_{n=1}^N \theta(\mathbf{x}_n)\ p(c_k|\mathbf{y}_n, \mathbf{x}_n)}{\sum_{n=1}^N p(c_k|\mathbf{y}_n, \mathbf{x}_n)} \tag{5.42}$$

In the case of second order polynomials as in (5.34) and (5.36), we get

$$\mathbf{B}_k = \begin{pmatrix} \langle 1 \rangle & \langle x_1 \rangle & \langle x_2 \rangle & \langle x_1 x_2 \rangle & \langle x_1^2 \rangle & \langle x_2^2 \rangle \\ \langle x_1 \rangle & \langle x_1^2 \rangle & \langle x_1 x_2 \rangle & \langle x_1^2 x_2 \rangle & \langle x_1^3 \rangle & \langle x_1 x_2^2 \rangle \\ \langle x_2 \rangle & \langle x_1 x_2 \rangle & \langle x_2^2 \rangle & \langle x_1 x_2^2 \rangle & \langle x_1^2 x_2 \rangle & \langle x_2^3 \rangle \\ \langle x_1 x_2 \rangle & \langle x_1^2 x_2 \rangle & \langle x_1 x_2^2 \rangle & \langle x_1^2 x_2^2 \rangle & \langle x_1^3 x_2 \rangle & \langle x_1 x_2^3 \rangle \\ \langle x_1^2 \rangle & \langle x_1^3 \rangle & \langle x_1^2 x_2 \rangle & \langle x_1^3 x_2 \rangle & \langle x_1^4 \rangle & \langle x_1^2 x_2^2 \rangle \\ \langle x_2^2 \rangle & \langle x_1 x_2^2 \rangle & \langle x_2^3 \rangle & \langle x_1 x_2^3 \rangle & \langle x_1^2 x_2^2 \rangle & \langle x_2^4 \rangle \end{pmatrix} \tag{5.43}$$

The cluster-weighted equivalent of the fully general case (5.31 and 5.32), yields

$$\begin{aligned} \mathbf{B}_k &= \begin{pmatrix} \langle \Psi_0(\mathbf{x}) \Psi_0(\mathbf{x}) \rangle & \ldots & \langle \Psi_0(\mathbf{x}) \Psi_I(\mathbf{x}) \rangle \\ \ldots & \ldots & \ldots \\ \langle \Psi_I(\mathbf{x}) \Psi_0(\mathbf{x}) \rangle & \ldots & \langle \Psi_I(\mathbf{x}) \Psi_I(\mathbf{x}) \rangle \end{pmatrix} \\ \mathbf{C}_k &= \begin{pmatrix} \langle y_{0,n} \Psi_0(\mathbf{x}) \rangle & \ldots & \langle y_{D_y,n} \Psi_0(\mathbf{x}) \rangle \\ \ldots & \ldots & \ldots \\ \langle y_{0,n} \Psi_I(\mathbf{x}) \rangle & \ldots & \langle y_{D_y,n} \Psi_I(\mathbf{x}) \rangle \end{pmatrix} \end{aligned} \tag{5.44}$$

5.3 Applications

5.3.1 Predicting physical systems

Section 2.2.1 introduced the embedding theorem and algorithmic tools regarding the prediction and characterization of physical systems. We showed that embedding has the conceptual potential of predicting any physical system, but also observed that in practice the number of systems actually tractable with the method is rather small. In this section we will discuss data set A from the Santa Fe Time Series Competition, a laser fluctuating at the gain threshold [HWAT93].

Figure 5-1 illustrates various aspects of the laser data set, e.g. the time domain signal. We embed the data in a two-dimensional lag space and build a predictive CWM model in this space. Figure 5-1 shows the data along with the clusters and a prediction surface. We indicate the density estimation of the data and the predictor uncertainty by shading the surface.

The Lorenz set (section 2.2.3) was originally discovered to describe chaotic fluid dynamics. The three-dimensional set of equations (equ. 2.26) generates varying dynamics depending on the choice of parameters.[1] It is famous for its strange attractor which is rare phenomena for systems with so few degrees of freedom. Hübner et al. [HWAT93] find that the NH_3-FIR laser generates a physical signal with dynamics very similar to those of the chaotic Lorenz set. They find that the parameters $r = 1...20, \sigma = 2, b = 0.25$ used with the Lorenz equations nicely simulate the laser (fig. 5-1).

We use CWM to forecast future values of the laser using such simulated data, working in the signal rather than in the intensity space. See fig. 5-2 for results.

[1] Figure 2-5 and 3-6 illustrates the Lorenz set (equ. 2.26) for the parameters $\sigma = 10$, $\rho = 8/3$,$b = 28$.

5.3.2 Prediction of the M3-competition time series

The M3-competition data set[2] consists of 3003 time series, most of which describe economical, industrial and demographic data from the past decades. Typical examples are the revenue of a specific industrial sector in a specific country, interest rates, and social security expenses for subpopulations. There are six different types of series (micro, industry, finance, demographic and other) sampled at four different time intervals. The amount of data provided is ≥ 14 points for yearly data, ≥ 16 points for quarterly data, ≥ 48 points for monthly data and ≥ 60 points for other data. The forecasting time horizons are six periods for yearly data, eight periods for quarterly data, 18 periods for monthly data and eight periods for other data. Fig. 5-3 shows some representative time series from the set [HM00].

Table 5.1: Prediction results - M3 competition, The error metric is the average symmetric MAPE [Hib99]. Errors are in percentages and ordered by time periods (vertical) and prediction horizons in terms of time period (horizontal).

pred. Horizon	1	2	3	4	5	6	8	12	15	18
all data (3003)	9.0	10.0	10.3	10.7	11.5	12.4	12.5	11.9	14.8	16.3
yearly data (645)	8.8	13.4	18.9	20.8	23.9	26.0				
quaterly data (756)	5.9	8.0	9.3	10.7	12.1	13.6	15.0			
monthly data (1428)	13.5	13.3	14.3	16.1	14.3	14.0	15.0	15.3	19.0	20.8
other data (174)	1.9	3.1	4.6	5.4	6.2	6.2	7.6			

To start, we extract two subsets of points from each set: the training set (about 80% of the data) and the test set (about 20% of the data). The training set is used to fit models with different hyper-parameter settings. We randomly select the auto-regressive dimension for the model as well as the number of clusters and the order of the local fit. However, we constrain the window of possible values depending on the type of series and on the amount of data available. The less data there is, the less model resources are allocated. For all series, one of the parameter choices is a low-dimensional single-cluster linear model. The different model choices are trained on parts (75%) of the training set randomly choosing about five different subsets for training (bootstrapping). Each incidence of a model is tested on the unused data from the training set. The performance of a parameter set is taken to be the average error of all the bootstrapingp cycles.

We build a different model for the different forecasting horizons (fig. 5-4) in order to avoid adding up errors in the iterative scheme. We also train each set in absolute as well as differential mode. In the differential mode, we predict the increment per period, which simplifies the prediction for time series that follow an exponential growth, such as economic time series. The set of parameters with the smallest error on the boot-strapped training data is used to predict the testset. We choose the differential or non-differential method depending on which performs best on the testset. We then rebuild the model one more time, using the selected combination of parameters with the full data set. This very last model is used to predict the values of the requested forecasting horizon (Table 5.1).

[2]The data set of the M3-competition is provided by M. Hibon and S. Makridakis, INSEAD, France [HM00].

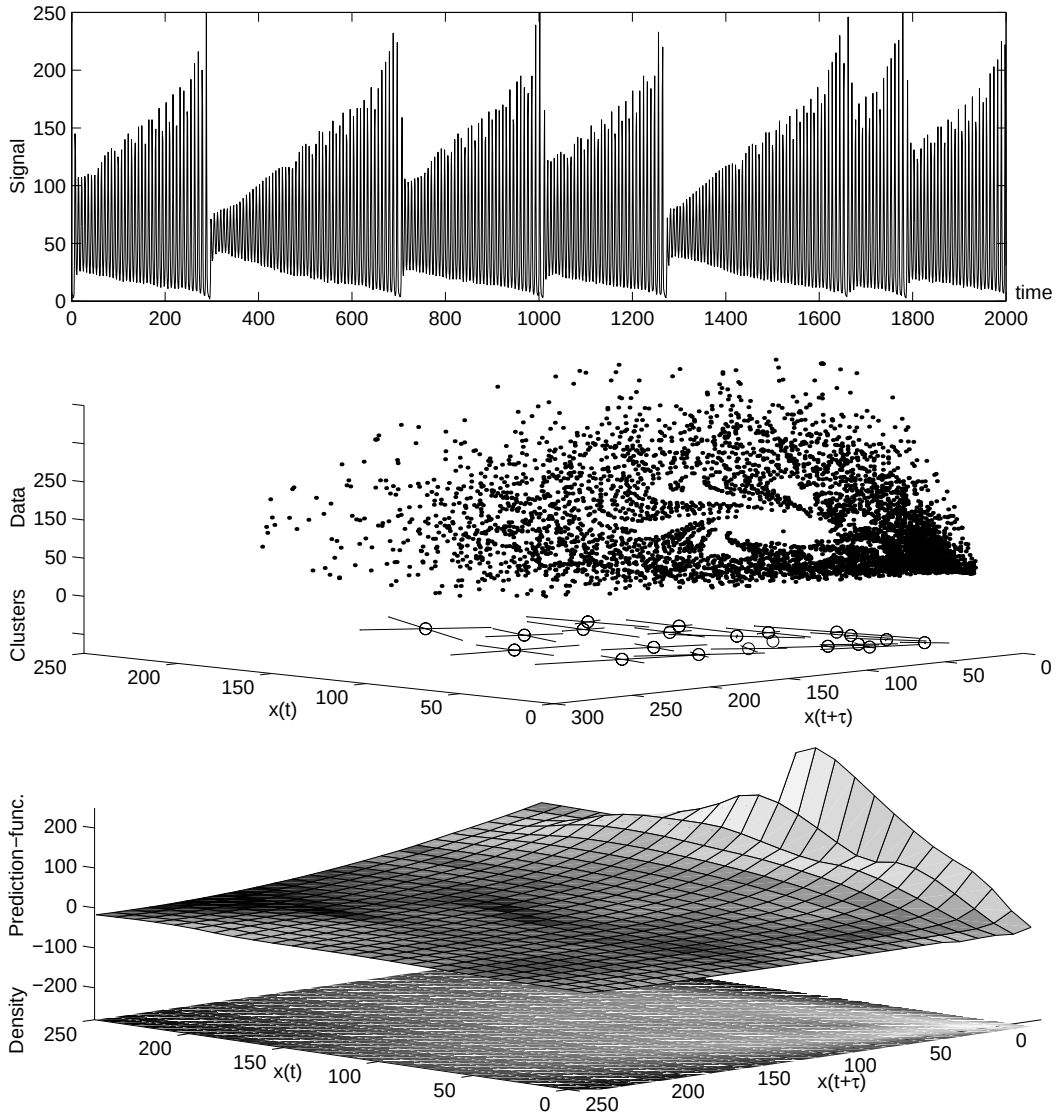


Figure 5-1: Modeling of a laser fluctuating near the gain threshold. *Top:* the measured time series. *Middle:* the data in a three dimensional time-lag space and the resulting cluster means and covariances. *Bottom:* the prediction surface derived from the resulting density, colored by the conditional uncertainty in the prediction, plotted above the input density estimate.

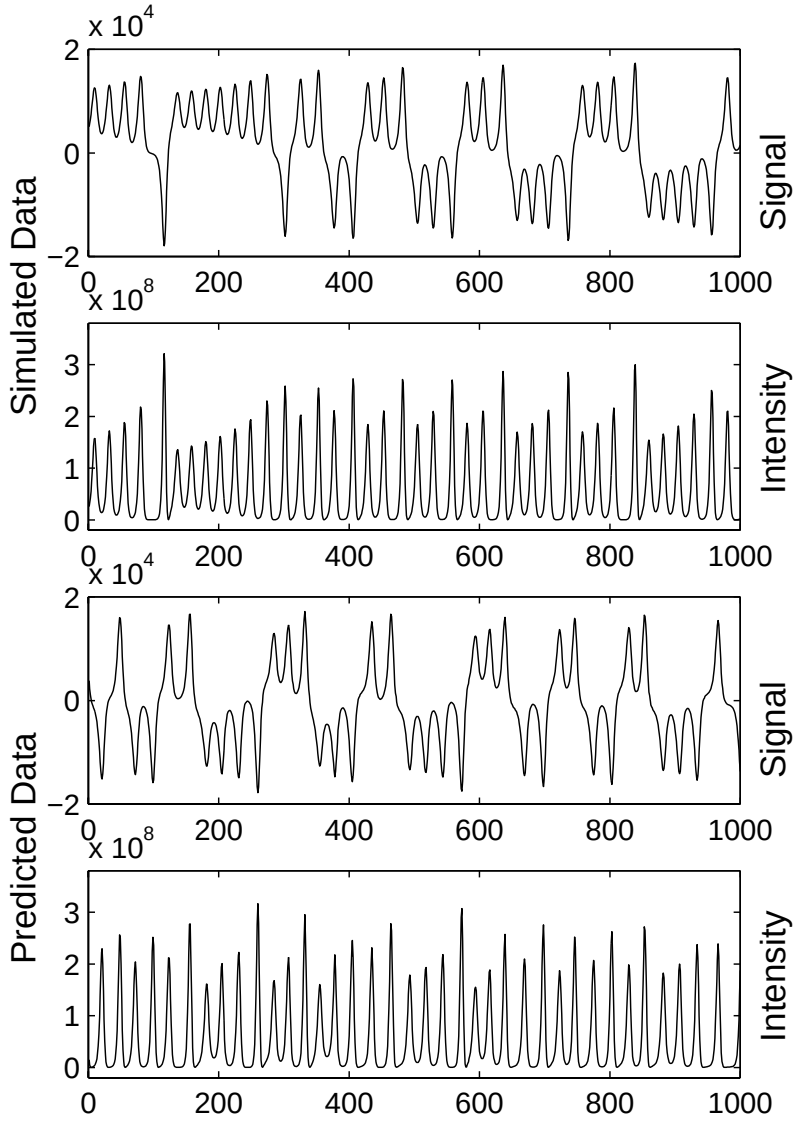


Figure 5-2: Predicting a laser/Lorenz model. *Top:* synthesized training signal and its intensity. *Bottom:* predicted signal and its intensity. The model uses a four-dimensional lag-space (lag = 16 samples) and ten clusters representing linear models.

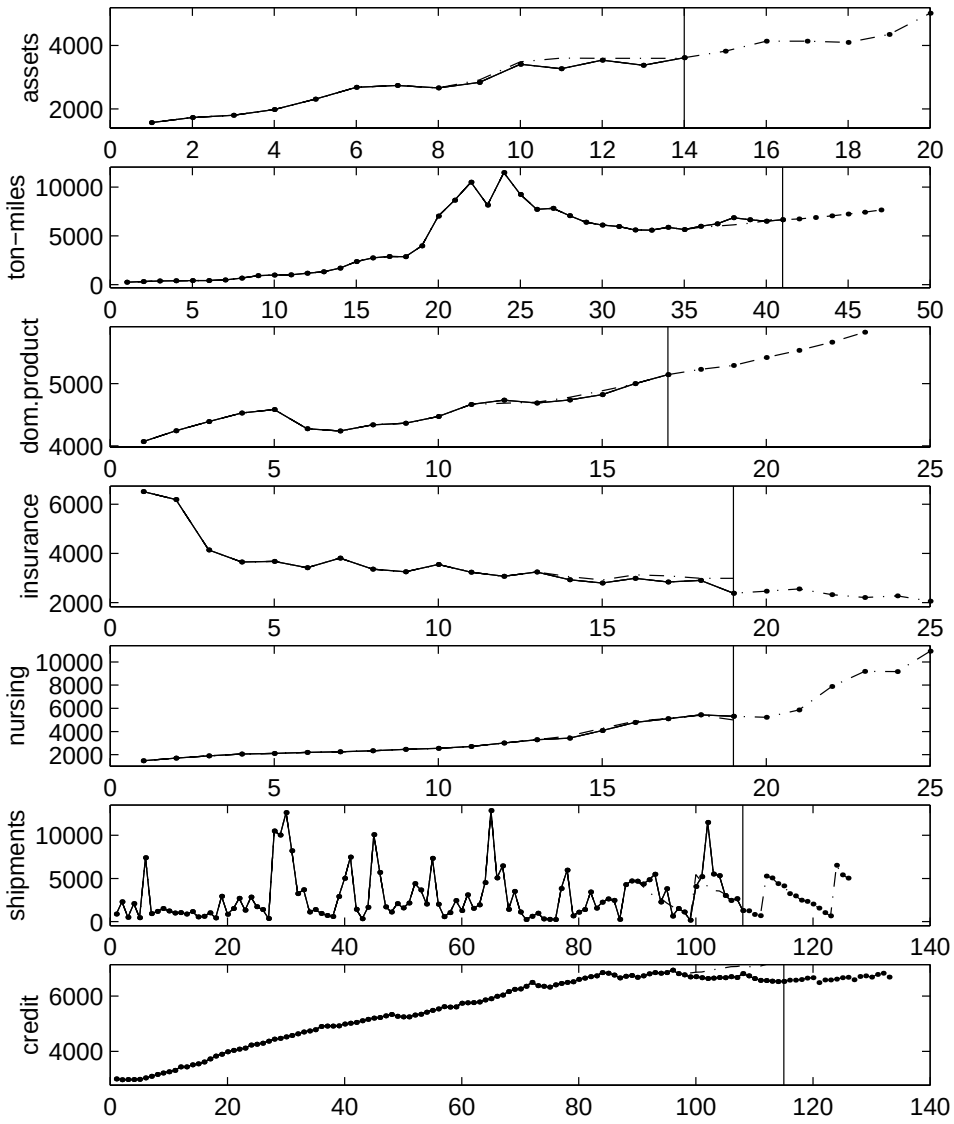


Figure 5-3: Examples of M3-competition time series. *From top to bottom:* assets (yearly/micro,starting 1/1975), air carriers (international), mail ton-miles (yearly/industry, starting 1/1947), OECD ECONOMIC OUTLOOK - SWITZERLAND, Gross Domestic Product- Volume Indices (yearly/macro, starting 1/1970), Veterans Servicemen'group life insurance - Post separation (yearly/demographic, starting 1/1971), Nursing Home Care - Veterans Medical Center (yearly/demographic, starting 1/1971), SHIPMENTS (Code TD-AUTOUNITS) (monthly/micro, starting 10/1984), Consumer installment credit outstanding, commercial banks (monthly/finance, starting 1/1083). The training data reaches up to the vertical bars. The dash-doted lines indicate the predictions.

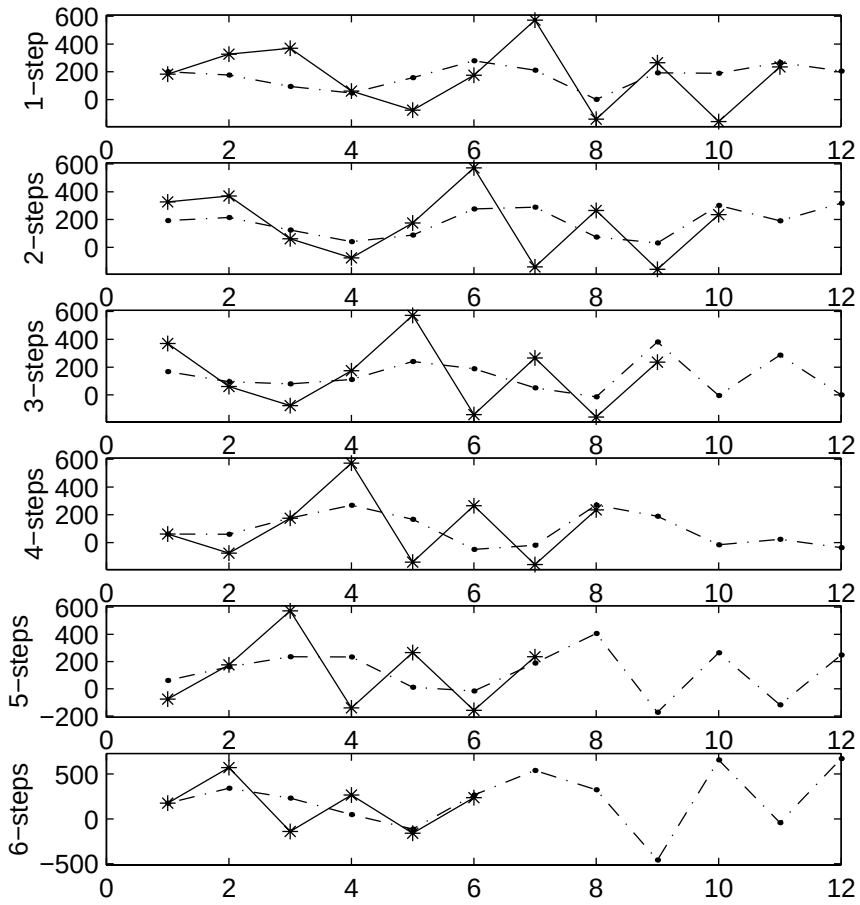


Figure 5-4: Example of a M3-competition prediction. Different models handle different prediction horizons, such that a one-step (p-step) predictor looks ahead one (p) data point(s) and predicts one step (p steps) ahead in the future.

Chapter 6

Linear predictive coding

6.1 Introduction and related work

Linear Predictive Coding (LPC) is a successful synthesis technique for time series with peaky spectra [MG76]. It is used first and foremost in speech synthesis. LPC aims to best approximates the spectrum of a given time series in a mean-square sense [Moo78]. In general, it is based on a stochastic or periodic excitation function which is convolved with a linear filter (IIR-filter). The filter coefficients as well as the excitation functions can be derived from observed data.

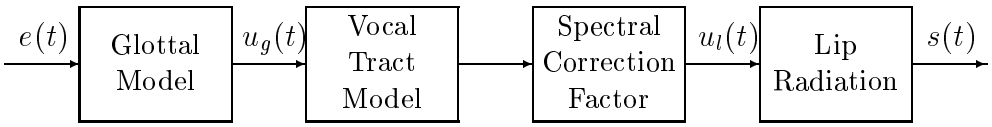


Figure 6-1: Linear speech production model [MG76, p.6]

In linear speech synthesis the vocal tract is typically decomposed into an excitation element, representing the vocal chords and a series of linear filters. In the z-domain we get

$$S[z] = E[z] \cdot G[z] \cdot V[z] \cdot L[z] \quad , \tag{6.1}$$

where $E[z]$ is the excitation function, typically a periodic function plus a noise term, $G[z]$ models the glottal shaping, $V[z]$ is the impulse response of the vocal tract, and $L[z]$ models the lip radiation. The transfer functions can be summarized in the all-pole filter $1/A[z]$.

$$\begin{aligned} A[z] &= \sum_{m=0}^{M} a_k z^{-m} \\ &= \frac{1}{G[z]V[z]L[z]} \end{aligned} \tag{6.2}$$

$$S[z] \quad = \quad E[z] \cdot \frac{1}{A[z]}$$

The set of linear filter coefficients **a** (equ. 6.3) of the all-pole filter $1/A[z]$ is the most efficient linear representation and is easy to estimate from observed data.

Equ. (6.1) is referred to as the synthesis model. It is presented here in the context of speech modeling, but generalizes to all linear signal sources. Given the order of approximation I, that is the dimension of the linear filter structure, we find the filter coefficients of $1/A[z]$ in a linear least-square fit. The auto-regressive structure of the model allows for two alternative training techniques. The **covariance method** minimizes the linear prediction error over the interval $[M, N-1]$, where M is the order of the filter and N is the number of points considered [MG76]. The method solves

$$\sum_{i=1}^{M} a_i \; c_{i,j} = -c_{0,j} \quad , \tag{6.3}$$

for $j = 1, 2, ...M$, with

$$c_{i,j} = \sum_{n=M}^{N-1} s(n-i) \; s(n-j) \quad . \tag{6.4}$$

The **auto-correlation method** minimizes the approximation error on the full real axis $[-\infty \;\; \infty]$. The method solves

$$\sum_{i=1}^{M} a_i r(|i-j|) = -r(j) \quad , \tag{6.5}$$

for $j = 1, 2, ...M$, with

$$r(l) = \sum_{n=0}^{N-1-l} s(n)s(n+l) \quad l \geq 0 \quad . \tag{6.6}$$

The sequence $e[n]$ is referred to as the excitation function, but also corresponds to the error sequence of the linear fit. It can be estimated by convolving $S[z]$ with the inverse filter $A[z]$,

$$E[z] = S[z]A[z] \quad . \tag{6.7}$$

This equation describes the analysis model. Alternatively $e[n]$ can be estimated in the time domain by subtracting a one-step-ahead predicted audio signal from the true signal. The equivalent of equation 6.7 in the time domain is

$$\begin{aligned} e(n) \quad &= \quad \sum_{i=0}^{M} a_i s(n-i) \\ &= \quad s(n) + \sum_{i=1}^{M} a_i s(n-i) \end{aligned} \tag{6.8}$$

since $a_0 = 1$. With the predicted sample $\hat{s} = -\sum_{i=1}^{M} a_i s(n-i)$ the time domain error

signal is

$$e(n) = s(n) - \hat{s}(n) \quad . \tag{6.9}$$

This expression can be exploited to define the excitation function [MG76].

Multiple excitation-filter pairs are estimated from different segments of the time series. Fundamentally, voiced sounds require a pitched periodic excitation while unvoiced sounds are generated by a non-periodic excitation. Therefore the excitation is typically composed of a periodic and a noise component.

Since $1/A[z]$ is an IIR filter, stability of the estimated filters is a major concern. An all-pole filter $F[z]$, is stable if all the zeros of the inverse $1/F[z]$ are inside the unit circle in the z-plane ($|z| < 1$). This property is not enforced by the linear estimation of $1/A[z]$. However, we can test a filter for its stability [MG76, P.93ff] and alternate it in case it is unstable. One approach to this replaces all the roots outside the unit circle by their reciprocals,[1] which assures stability but alters the spectral properties of the filter. Alternatively, we can change some parameters and reestimate the coefficients.

Related to the stability issue is that of switching from one filter to another. An abrupt change from one filter to the next will cause an audible artifact. A smooth transition (linear interpolation) from one filter to the next can cause stability problems, since the stability of the interpolated filters does not insure stability of the mixture. Therefore the transition window should be short.

It has been shown how the excitation signal can be derived from observed data. However, it can also be designed from scratch by adding white noise to a periodic impulse train. The noise component is simply colored by the formant structure of the linear filter. The impulse train has a harmonic spectrum, the pitch being determined by the period of the impulses.

The *cross-synthesis* of speech and music combines LPC's application to speech synthesis and musical synthesis. Since LPC is based on two major ingredients, excitation and filtering, we can combine features from two different sources. For example, the excitation function of a violin can be modulated by the formant structure of human speech resulting in a *talking violin*. Using a non-processed recorded signal of a musical instrument as excitation to the speech filters is valid may result in unintelligibility of the speech. Whitening of the excitation signal seems to solve this problem for most instrument families [Pet76] [Moo78], but it may cause problems for instruments with a strong noise floor, such as the flute.

6.2 Cluster-weighted linear predictive coding

Cluster-weighted linear predictive coding (CWLPC) generates time-varying all-pole filters, the coefficients of which depend on a control sequence. We are given a scalar-valued time series y and a vector-valued input time series $\mathbf{x}_s$ which we assume is changing slowly relative to the dynamics of y (equ. 3.5). y is typically an audio signal, while $\mathbf{x}_s$ summarizes a slowly varying control input or the known state of the system.

CWLPC requires three CWM expert models, either as separate models or within the

[1] Reciprocals are the mirror points relative to the unit circle.

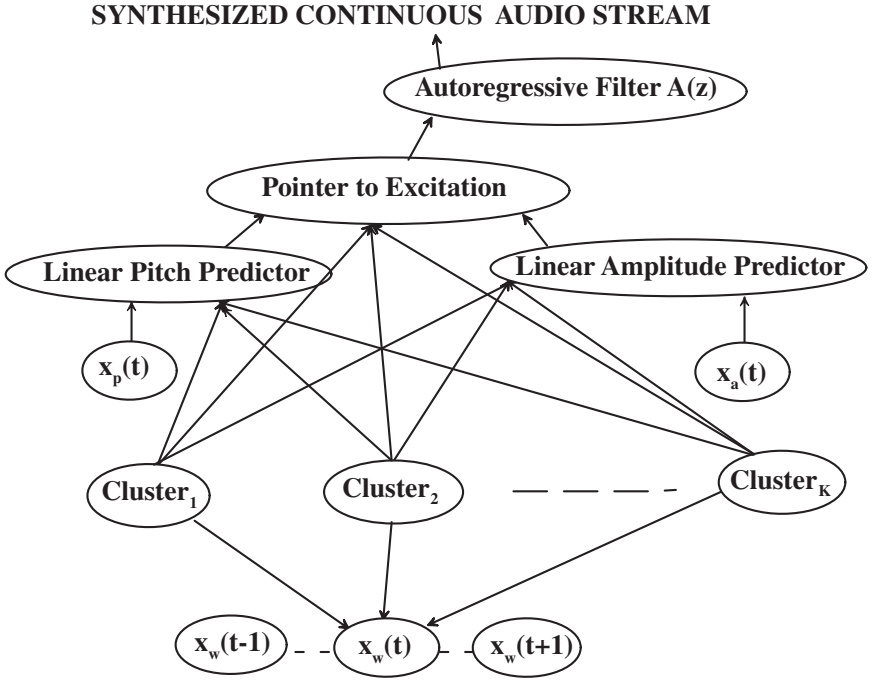


Figure 6-2: Cluster-weighted linear predictive coding in graph form: three separate output models predict pitch and amplitude and implement the autoregressive LPC filter.

same framework (fig. 6-2). The first expert predicts the filter coefficients a_i of an auto-regressive filter and the filter output $\hat{y}$. For the purpose of estimating this expert we construct the state vector $\mathbf{x}_f$, describing the fast dynamics (3.5), from lagged instances of y,

$$x_{i,f}[n] = y[n-i], \quad d = 1, ..., I \quad . \tag{6.10}$$

I denotes the order of approximation, i.e. the number of filter tabs. The filter output is then

$$\hat{y}[n] = \frac{\sum_{k=1}^{K} \mathbf{a}_k^T \mathbf{x}_f[n] \; p(\mathbf{x}_s[n]|c_k) \; p(c_k)}{\sum_{k=1}^{K} p(\mathbf{x}_s[n]|c_k) \; p(c_k)} \quad . \tag{6.11}$$

where $\mathbf{x}_f[n]$ is updated after each time step n according to (6.10) with

$$x_{1,f}[n] = \hat{y}[n-1] + e[n] \quad . \tag{6.12}$$

The estimation of the $\mathbf{a}_k$ is identical to the non-auto-regressive case (section 3.2).

The second expert is concerned with the prediction of the model gain. Since the linear all-pole filter can't be used to control the gain of the predictor, an expert is needed which scales the excitation function in such a way that the filter outputs the desired level. The expert is a simple scalar-valued locally-linear predictor that estimates the scaling factor

$v[n]$, given the input $\mathbf{x}_v$.

$$v[n] = \frac{\sum_{k=1}^{K} \mathbf{a}_{k,v}^T \mathbf{x}_v[n]\ p(\mathbf{x}_s[n]|c_k)\ p(c_k)}{\sum_{k=1}^{K} p(\mathbf{x}_s[n]|c_k)\ p(c_k)} \quad . \tag{6.13}$$

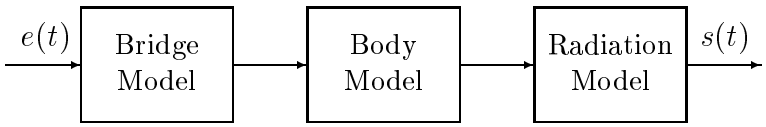


Figure 6-3: LPC model of a violin

The third expert is concerned with the excitation function $e[n]$. The expert is predicting the pitch of the periodic excitation as well as properties of the noise component of the excitation function. The pitch-expert is taken to be a locally linear predictor as above,

$$\mathrm{p}[n] = \frac{\sum_{k=1}^{K} \mathbf{a}_{k,p}^T \mathbf{x}_p[n]\ p(\mathbf{x}_p[n]|c_k)\ p(c_k)}{\sum_{k=1}^{K} p(\mathbf{x}_p[n]|c_k)\ p(c_k)} \quad . \tag{6.14}$$

Similar estimators can be designed to predict the properties of the noise floor.[2]

For synthesis, we superimpose an impulse train at the period of the predicted pitch $p[n]$ and an appropriate noise floor. The excitation is scaled by the scaling/volume factor $\mathrm{v}[n]$ and then used as the driving signal for the auto-regressive filter $1/A[z]$. The filter coefficients $a[i]$ are constructed as a weighted sum of all the individual filters. We assume that one cluster predominates the overall filter, in which case the stability of the individual filters is sufficient for the mixture-coefficients to be stable. Alternatively, a winner-takes-all strategy for the filter coefficients can be used, in which case only the coefficients of the cluster that is most likely given the input vector are used.

6.3 Application: An excitation-filter model of the violin

Like the vocal tract, a violin can be interpreted in terms of an excitation and a filter structure (fig. 6-3). The excitation can be taken to be a simple impulse train, a combination of an impulse train plus a noise term, or a sampled excitation function. Section 6.1 explained how to recover the excitation function from a linear filter model. Subsequently this excitation function is convolved with linear filters representing the bridge, the resonating body of the violin, and the sound radiation into the room. As in the speech model, these filters can be combined into a single all-pole filter $1/A[z]$.

The violin model is different from the speech model[3] in that the violin is a stationary

[2] Research that is yet to be done.

[3] ... and one might argue LPC is the wrong approach for modeling a violin. In fact this technique has not been pursued beyond initial attempts.

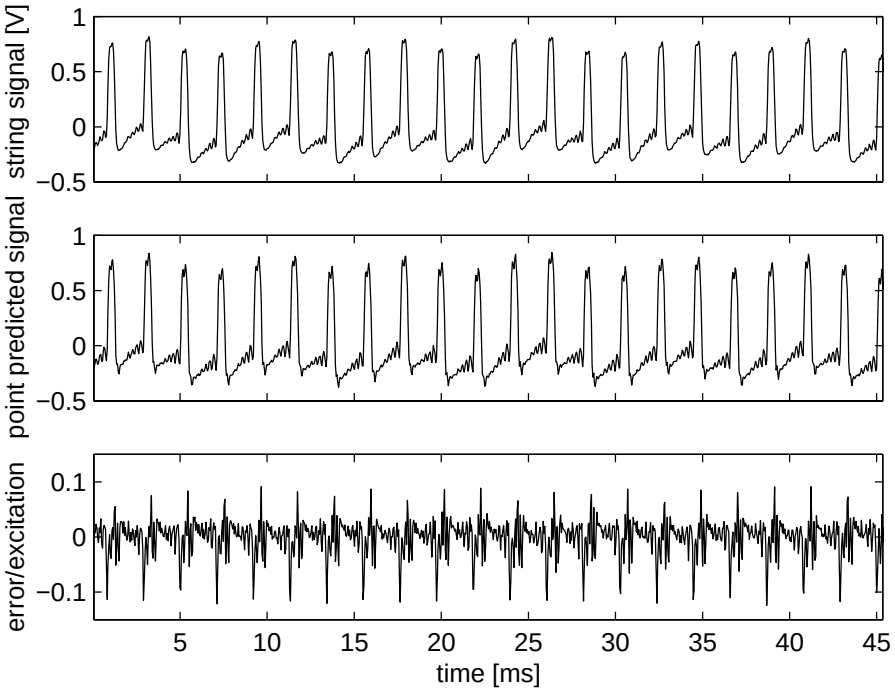


Figure 6-4: Cluster-weighted LPC model built on data recorded from a violin string signal. *Top*: training signal; *center:* point predicted signal; *bottom:* error signal / excitation function for a synthesis system. The model uses a linear filter with 10 tabs, five clusters, and a two-dimensional feature space.

device and hence the filter coefficients shouldn't need to change. In reality different filters end up modeling different excitation functions, which to some extent destroys the analogy between the elements of the model and the actual physical mechanisms [Moo78].

Unfortunately LPC really only shifts the problem of modeling a complicated nonlinear signal from modeling the true signal to estimating the excitation function. The excitation function is freed from the linear equalization, but still contains the interesting elements of the signal. It is therefore as difficult to predict as the sound pressure signal (Fig.6-4).

Chapter 7

Frequency domain estimation

7.1 Frequency domain representations of signals and systems

Complex input-output systems are conveniently represented as graphs of block elements (fig. 7-1) called *transfer functions*. Transfer functions are easy to understand and to manipulate. Every block consists of free input variables and output variables dependent on the inputs. The relationship between the input and output variables can be explicit, that is in the form $\mathbf{y}(t) = \mathbf{f}(\mathbf{u}(t))$, or implicit, for example in the form of a differential equation [Mey92].

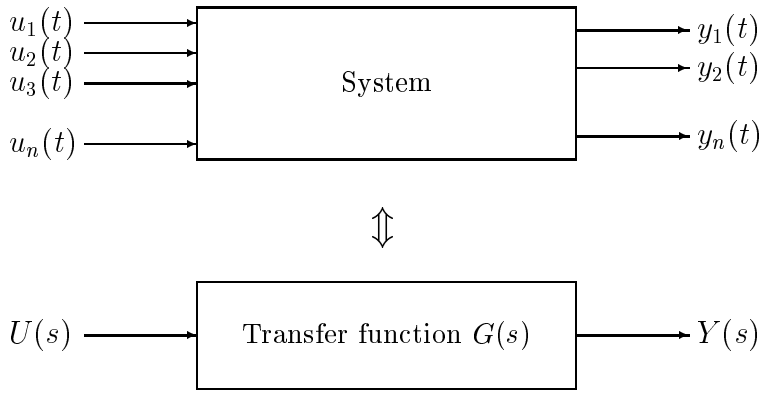


Figure 7-1: Dynamic system and Laplace transform [Mey92]

A particularly interesting subclass of transfer functions are referred to as linear time-invariant (LTI) systems. A system is linear, if the transfer function satisfies

$$f(a_1 u_1 + a_2 u_2) = a_1 f(u_1) + a_2 f(u_2) \tag{7.1}$$

Important examples for linear systems are $y(t) = \int_0^t u(\tau) d\tau$ and $y(t) = \frac{d}{dt} u(t)$. Examples for nonlinear systems are $y(t) = u(t)^2$ and $y(t) = e^u(t)$. The most general representation

of a linear transfer function is a linear differential equation

$$y^n + a_{n-1}y^{n-1} + ... + a_1 y' + a_0 y = f(u) \tag{7.2}$$

If the coefficients a_i in 7.2 are independent of time, the transfer function is said to be time invariant. The condition for time invariance is

$$y(t) = f(u(t)) \Rightarrow y(t - t_0) = f(u(t - t_0)) \quad . \tag{7.3}$$

If the system is time-continuous the Laplace transform $\mathcal{L}$ of the linear differential equation results in a much simpler algebraic expression. For example,

$$\begin{array}{rcll} T\dot{y} + y & = & ku(t), \quad y(0) = y_0 & (7.4) \\ & \Updownarrow & \mathcal{L} & \\ sTY(s) & - & Ty(0) + Y(s) = kU(s) & . \end{array}$$

This expression is easily solved for $Y(s)$ or $G(s)$

$$\begin{array}{rcl} Y(s) & = & \dfrac{kU(s)}{1+sT} + \dfrac{Ty(0)}{1+sT} \\ & = & G(s)U(s) + G_0(s)Ty(0) \\ G(s) & = & \dfrac{Y(s)}{U(s)} \end{array} \tag{7.5}$$

$G(s)$ is the Laplace transform of the impulse response of the system (fig. 7-1). For LTI systems, the impulse response and its transform are equivalent descriptions.

The frequency response of the system equals $G(s)$ for $s = i\omega$. If all the poles of $G(s)$ are in the left half of the complex plane, $G(i\omega)$ also equals the Fourier transform of the impulse response $g(t)$. In this case the frequency response describes the relationship between a sinusoidal component at the input port and a sinusoidal component at the output port of the system. The decomposition $G(i\omega) = \mathrm{Re}\{G(i\omega)\} + \mathrm{Im}\{G(i\omega)\}$ is used for a variety of useful visualization and characterization techniques, e.g. for logarithmic magnitude and phase diagrams of the transfer function.

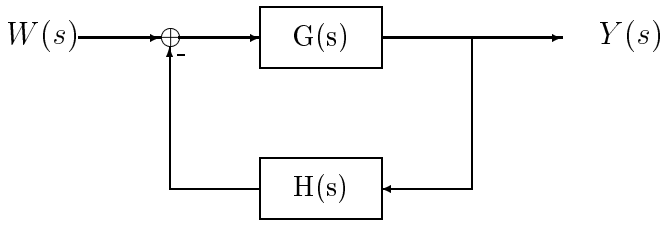


Figure 7-2: Feedback system in the Laplace domain [Mey92, p.100].

The representation as a transfer function is crucial for evaluation of the stability of

a system. A system is stable if and only if the system response $\mathbf{y}(t)$ is bounded for any bounded input $\mathbf{u}(t)$. The translation of this condition in the frequency domain yields the following result: a linear system with a rational transfer function $G(s)$ is stable if and only if all poles of the transfer function have a negative real part. Most nontrivial engineering applications of control systems involve a feedback loop as in fig. 7-2. The stability of the closed-loop system largely depends on the properties of the open-loop transfer function $G_0(s)\hat{=}G(s)H(s)$. The Nyquist-criterium [OW83, P.717][Mey92] is usually used to examine feedback systems for their absolute stability, but also for a number of properties summarized as relative stability.

7.1.1 Volterra expansion

The output of a linear operator and system is a linear combination of all the inputs of the same frequency (equ. 7.2). In other words, spectral components don't interact with each other but are simply scaled and phase shifted. For a nonlinear operator, this restriction doesn't hold, but components are allowed to interact with a component of a different frequency or a component from a different input port. If the input $x(t)$ is given by

$$x(t) = \sum_{n=1}^{N} c_n \; x_n(t) \tag{7.6}$$

we obtain for the response of a second order operator

$$\begin{aligned} y(t) &= \mathbf{T}_2[x(t)] \\ &= \mathbf{T}_2[\sum_{n=1}^{N} c_n x_n(t)] \\ &= \sum_{k=1}^{N}\sum_{n=1}^{N} \mathbf{T}_2\{c_k x_k(t), c_n x_n(t)\} \\ &= \sum_{k=1}^{N}\sum_{n=1}^{N} c_k c_n \mathbf{T}_2\{x_k(t), x_n(t)\} \end{aligned} \tag{7.7}$$

If for example

$$y^{\dagger}(t) = x^2(t) \quad , \tag{7.8}$$

the response of the system is

$$y^{\dagger}(t) = \sum_{k=1}^{N}\sum_{n=1}^{N} c_k c_n x_k(t) x_n(t) \quad , \tag{7.9}$$

and hence $\mathbf{T}_2^{\dagger}$ is

$$\mathbf{T}_2^{\dagger}\{x_k(t), x_n(t)\} = x_k(t)x_n(t) \quad . \tag{7.10}$$

If $\mathbf{T}_2$ describes a time invariant system it turns into the second order Volterra operator

$$y(t) = \mathbf{H}_2[x(t)] \tag{7.11}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2 \quad .$$

h_2 is called the second-order Volterra kernel and can be interpreted as a two dimensional impulse response. Let's reconsider system (7.8) assuming that $x(t)$ is a superposition of two sinusoidal components $x_1(t)$ and $x_2(t)$ and that $f_2 = 2f_1$.

$$\begin{aligned} x(t) &= a_1\sin(2\pi f_1 t+\phi_1) + a_2 sin(2\pi f_2 t+\phi_2) \\ &= Re\{\underline{c}_1[\cos(2\pi f_1 t) + i\cdot\sin(2\pi f_1 t)] + \underline{c}_2[\cos(2\pi f_2 t) + i\cdot\sin(2\pi f_2 t)]\} \\ &= Re\{\underline{c}_1\underline{x}_1(t) + \underline{c}_2\underline{x}_2(t)\} \end{aligned} \tag{7.12}$$

where the $\underline{c}_i$ are complex valued coefficients. Then

$$\begin{aligned} \underline{y}^{\dagger}(t) &= \sum_{k=1}^{2}\sum_{n=1}^{2}\underline{c}_k\underline{c}_n\underline{x}_k(t)\underline{x}_n(t) \\ &= \underline{c}'_{11}\underline{x}_{11}(t) + \underline{c}'_{12}\underline{x}_{12}(t) + \underline{c}'_{21}\underline{x}_{21}(t) + \underline{c}'_{22}\underline{x}_{22}(t) \end{aligned} \tag{7.13}$$

with $\underline{x}_{ij}(t) = \underline{x}_i(t)\underline{x}_j(t)$. Since the mixed terms have the same frequency and $f_2 = 2f_1$, this superposition of terms can be rewritten as

$$\underline{y}^{\dagger}(t) = c''_0 + c''_1\underline{x}_1(t) + c''_2\underline{x}_2(t) + c''_3\underline{x}_3(t) + c''_4\underline{x}_4(t)$$

with $\underline{x}_j(t)$ representing the sinusoidal basis term of frequency $j\cdot f_1$.

The second-order operator is sufficient to handle second-order nonlinearities as in equ. 7.8. In order to deal with arbitrary nonlinearities we can generalize $\mathbf{T}$ into the pth-order operator $\mathbf{T}_p$. The impulse response of $\mathbf{T}_p$ is given by

$$\begin{aligned} y_p(t) &= \mathbf{T}_p[x(t)] = \mathbf{T}_p[\sum_{n=1}^{N} c_n x_n(t)] \\ &= \sum_{n_1=1}^{N}\ldots\sum_{n_p=1}^{N}\mathbf{T}_p\{c_{n_1}x_{n_1}(t),\ldots,c_{n_p}x_{n_p}(t)\} \\ &= \sum_{n_1=1}^{N}\ldots\sum_{n_p=1}^{N} c_{n_1}\ldots c_{n_p}\mathbf{T}_p\{x_{n_1}(t),\ldots,x_{n_p}(t)\} \end{aligned} \tag{7.14}$$

$\mathbf{T}_p$ is called a p-linear operator, since it is linear in each argument when all the other arguments are held fixed [Sch89]. If $\mathbf{T}_p[\cdot]$ is time-invariant, it is called the *pth*-order Volterra operator $\mathbf{H}_p[\cdot]$. We can relate $y_p(t)$ to the input using

$$\begin{aligned} y_p(t) &= \mathbf{H}_p[x(t)] \\ &= \int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty} h_p(\tau_1,\ldots,\tau_p)x(t-\tau_1)\ldots x(t-\tau_p)d\tau_1\ldots d\tau_p \end{aligned} \tag{7.15}$$

The number of terms in the approximation is exponential in p. Yet, given a specific problem or a specific nonlinearity a low dimensional approximation may be perfectly valid (section 7.3).

7.2 Cluster-weighted complex estimation

In this section we consider the approximation of a nonlinear complex-valued input-output function $\mathcal{C}^K \rightarrow \mathcal{C}^L$, given a set of complex valued input-output measurements $\{\underline{\mathbf{y}}, \underline{\mathbf{x}}\}_{n=1}^N$, where the $\underline{\mathbf{x}}_n$ are K-dimensional vector observations and $\underline{\mathbf{y}}_n$ are L-dimensional observations. There are two approaches to this problem:

- The complex valued $\underline{x}_j$ are represented as $\underline{x}_j = x_{\mathrm{Re},j} + i \cdot x_{\mathrm{Im},j}$ and the complex valued $\underline{y}_j$ are represented as $\underline{y}_j = y_{\mathrm{Re},j} + i \cdot y_{\mathrm{Im},j}$. Real and imaginary parts are treated as independent variables. Hence the problem becomes that of a real-valued function $\mathcal{R}^{2K} \rightarrow \mathcal{R}^{2L}$.
- The complex function is approximated using truly complex arithmetic. In the case of a linear fit, this means complex-valued coefficients and complex valued basis functions. In the case of a neural network, this means complex activation functions and complex back-propagation for training [GK92][Hay96][Sma97].

The advantage of the first approach clearly is that we need not change the architecture at all. However, it has been shown that LMS training is faster and more stable in the second case [Hay96, p.826]. Also, we conceptually prefer to keep the truly complex representation since for many applications it provides meaningful insight into the problem. We present a fully complex CWM model but point out that the transparency of the CWM architecture allows for combinations of complex and real valued data. For example, the global weighting can be based on a real valued input $\mathbf{x}_s$ while the local dynamics are described in the complex plane $\underline{\mathbf{x}}_f$. For example, it may make perfect sense to weight the model based on the magnitude of the input, while the local models have complex valued in inputs and outputs.

The change to complex arithmetic simply translates into a redefinition of the inner product,

$$\begin{aligned} \langle \underline{\mathbf{z}}_1 \underline{\mathbf{z}}_2 \rangle &= \overline{\underline{\mathbf{z}}}_1^T \underline{\mathbf{z}}_2 \\ &= \sum_i \overline{\underline{z}}_{1,i} \; \underline{z}_{2,i} \end{aligned} \tag{7.16}$$

where $\overline{\underline{z}}$ is the complex conjugate of $\underline{z}$ [Mag00]. This means that all the matrix transposes are replaced by the conjugate complex transposes. The covariance matrices on a complex support are hermitsch.

In the fully complex approach, we redefine the Gaussian distributions on a complex support (section 3.1).

$$p(\underline{\mathbf{x}}|c_k) = \frac{|\underline{\mathbf{P}}_k^{-1}|^{1/2}}{(2\pi)^{D/2}} e^{-(\overline{\underline{\mathbf{x}}}-\overline{\underline{\mathbf{m}}}_k)^T \cdot \underline{\mathbf{P}}_k^{-1} \cdot (\underline{\mathbf{x}}-\underline{\mathbf{m}}_k)/2} \quad , \tag{7.17}$$

and

$$p(\underline{\mathbf{y}}|\underline{\mathbf{x}}, c_k) = \frac{|\underline{\mathbf{P}}_{m,y}^{-1}|^{1/2}}{(2\pi)^{Dy/2}} e^{-(\overline{\underline{\mathbf{y}}}-\overline{\underline{\mathbf{f}}}(\underline{\mathbf{x}},\underline{\mathbf{a}}_k))^T \cdot \underline{\mathbf{P}}_{m,y}^{-1} \cdot (\underline{\mathbf{y}}-\underline{\mathbf{f}}(\underline{\mathbf{x}},\underline{\mathbf{a}}_k))/2} \quad . \tag{7.18}$$

We furthermore develop the update for locally linear complex predictors. Both basis functions and coefficients are complex valued,

$$\begin{aligned} \underline{f}_k(\underline{\mathbf{x}}) &= \overline{\underline{\mathbf{a}}}^T \cdot \underline{\mathbf{x}} \\ &= \sum_{d=0}^{D} \overline{\underline{a}}_d \; \underline{x}_d \quad . \end{aligned} \tag{7.19}$$

The complex-valued error is defined as

$$\underline{e}_n = \hat{\underline{y}}_n - \underline{y}_n = \mathrm{Re}\{\hat{\underline{y}}_n\} - \mathrm{Re}\{\underline{y}_n\} + i \; \mathrm{Im}\{\hat{\underline{y}}_n\} - i \; \mathrm{Im}\{\underline{y}_n\} \tag{7.20}$$

and the complex error function as

$$E[\underline{e} \; \overline{\underline{e}}] = E[e_{\mathrm{Re}}^2 + e_{\mathrm{Im}}^2] = E[e_{\mathrm{Re}}^2] + E[e_{\mathrm{Im}}^2] \tag{7.21}$$

From this we derive the complex gradient

$$\begin{aligned} \nabla_{\underline{a}} &= \frac{\partial}{\partial \underline{a}} \left[f_{\mathrm{Re}}(\underline{\mathbf{x}}, \underline{a}) + i f_{\mathrm{Im}}(\underline{\mathbf{x}}, \underline{a}) \right] \cdot \nabla_{\underline{a}} \underline{a} \\ &= \frac{\partial}{\partial \underline{a}} \underline{f}(\underline{\mathbf{x}}, \underline{a}) \cdot \begin{bmatrix} 1 \\ i \end{bmatrix} \quad . \end{aligned} \tag{7.22}$$

It is easily verifiable that the update rule for the complex-valued basis functions is (equ. 3.23)

$$0 = \left\langle \left[\overline{\underline{y}} - \overline{\underline{f}}(\mathbf{x}, \underline{a}_k) \right] \; \frac{\partial \underline{f}(\underline{\mathbf{x}}, \underline{a}_k)}{\partial \underline{a}_k} \right\rangle_k$$

where $\langle \cdot \rangle_k$ denotes the cluster weighted expectation with respect to the Gaussian cluster distributions on the complex support. Hence the update rule for the matrix of coefficients $\underline{A}$ in the vector valued case is

$$\underline{A}_k = \underline{\mathbf{B}}_k^{-1} \cdot \underline{\mathbf{C}}_k \quad , \tag{7.23}$$

with

$$\begin{aligned} [\underline{\mathbf{B}}_k]_{ij} &= \langle \overline{\underline{x}}_i \cdot \underline{x}_j \rangle_k \\ &= \frac{1}{N \; p(c_k)} \sum_{n=1}^{N} \overline{\underline{x}}_i \cdot \underline{x}_j \; p(c_k | \underline{\mathbf{y}}_n, \underline{\mathbf{x}}_n) \\ [\underline{\mathbf{C}}_k]_{i,j} &= \langle \overline{\underline{y}}_i \cdot \underline{x}_j \rangle_k \\ &= \frac{1}{N \; p(c_k)} \sum_{n=1}^{N} \overline{\underline{y}}_i \cdot \underline{x}_j \; p(c_k | \underline{\mathbf{y}}_n, \underline{\mathbf{x}}_n) \quad . \end{aligned} \tag{7.24}$$

In similar fashion the updates for the remaining cluster parameters are:

$$
\begin{aligned}
p(c_k|\mathbf{y},\mathbf{x}) &= \frac{p(\underline{\mathbf{y}},\underline{\mathbf{x}}|c_k)\ p(c_k)}{\sum_{l=1}^{K} p(\underline{\mathbf{y}},\underline{\mathbf{x}}|c_l)\ p(c_l)} \\
p(c_k) &= \frac{1}{N}\sum_{n=1}^{N} p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n) \\
\langle \underline{\mathbf{m}} \rangle &= \frac{\sum_{n=1}^{N} \underline{\mathbf{x}}_n\ p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)}{\sum_{n=1}^{N} p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)} \\
\langle \underline{\mathbf{P}}_{i,j} \rangle &= \frac{\sum_{n=1}^{N} \overline{\underline{\mathbf{x}}}_{i,n}\ \underline{\mathbf{x}}_{j,n}\ p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)}{\sum_{n=1}^{N} p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)} \\
\langle \underline{\mathbf{P}}_{y,i,j} \rangle &= \frac{\sum_{n=1}^{N} \hat{\underline{\mathbf{y}}}_{i,n}\ \underline{\mathbf{y}}_{y,j,n}\ p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)}{\sum_{n=1}^{N} p(c_k|\underline{\mathbf{y}}_n,\underline{\mathbf{x}}_n)}
\end{aligned}
\tag{7.25}
$$

Figure 7-3: Device under test (DUT) characterized by its transfer function.

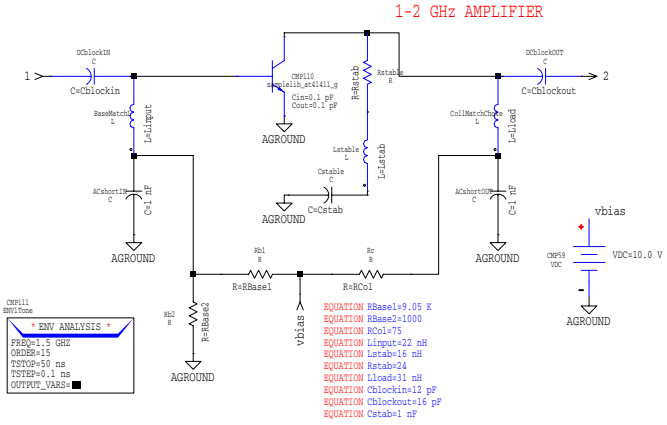


Figure 7-4: Device under test (DUT) for the extended S-parameter approach.

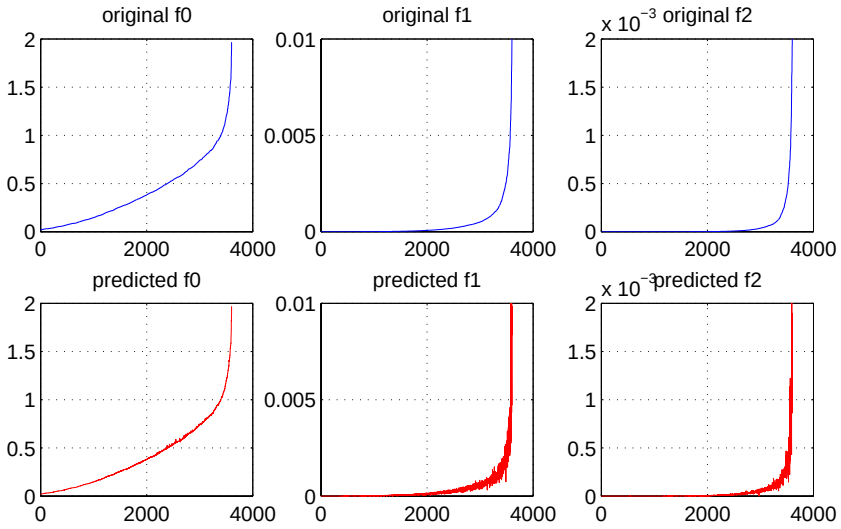


Figure 7-5: Approximated data (local third-order polynomials) from a simulated amplifier: $b_2(1), b_2(2), b_2(3)$. The x-axis gives the test event, the y-axis indicates the energy in the component. Original and predicted output data are sorted individually by the energy of the particular component. The noise (non-monotonicity) in the predicted f_1 and f_2 plot indicates a slight prediction error.

7.3 Application: nonlinear device characterization

Microwave device technology has evolved into application domains where its emulation and characterization with linear techniques are no longer satisfactory. Although nonlinear effects appear in many systems to such an extent that failure or success of ambitious design goals depend on them, the measurement technology as well as the data representation in state of the art analyzer systems still relies on the assumption of linearity. As a result, most of the interesting and useful nonlinear behavior of microwave devices is either missed or neglected [SCDG99].

Nonlinearity is difficult to model as well as to characterize; while physical models [FGN92] of devices and circuits require impractical amounts of computation, reliable electrical models are device-specific and tedious to design [MGOP+97]. Motivated by successful inference approaches to system characterization in other engineering domains, we characterize a device based on analysis of empirical data collected from the device under test (DUT).

Van den Bossche [VB94a][VB94b] introduced the nonlinear S-parameter equivalent for weakly nonlinear multi-port devices. A linear multi-port network is completely specified by its scattering and reflection parameters $S(\omega)_{i,j}$, where i refers to an input and j to an output signal, and the device load can be described by linear superposition of input signal components. However, transfer and reflection coefficients of nonlinear devices can only be specified as functions of all input frequency components. Nonlinear interaction

between frequency components can be modeled by polynomials beyond the first order linear terms. Such a model characterizes the harmonic responses at the output ports due to nonlinear coupling between harmonic components at the input ports, up to an arbitrary order of approximation. Although Volterra approximations perform well on the local scale [VB94a], their accuracy degrades over a larger dynamic range and a wider frequency span. Our algorithm overcomes this problem, as it optimally allocates a series of local models describing different input-output behavior.

Starting from a set of measurements taken from fig. 7-3 we identify the effect of combinations of input signals a_i on the output signals b_j. Restricting input and output to harmonic signal components, we denote the input component associated with the fundamental f_1, and the harmonics $f_2, f_3...$ by $a_i(1), a_i(2), ..$ respectively. We designate the corresponding output components by $b_i(1), b_i(2), ...$

The S-parameter approach for linear microwave devices is extended into nonlinear transmission kernels H_n using the Volterra theory [VB94b, Sch89]. $H_{n,ji_1i_2..i_n(f_1,f_2,..f_n)}$ describes the n-th order effect of frequency components f_k at the input port i_k on the frequency components $f_1+f_2+..+f_n$ at output port j, where conjugate complex components are denoted by a negative f_i [VB94b]. Instead of restricting the model to the mixed terms of the Volterra series we keep all the mixing terms between real and complex components up to the desired order O. The increase in the number of terms is compensated by a gain in symmetry that facilitates the parameter search.

Thus the local model is

$$\mathbf{y} = \sum_{e_1+e_2+..+e_D<=O} a \cdot x_1^{e1} \cdot x_2^{e_2} \cdot ... \cdot x_D^{e_d} \quad . \tag{7.26}$$

The order of the local model is traded for the complexity of the global architecture (fig. 7-6). While the polynomial expansion is very efficient within a limited input domain of interest, wide ranges of frequency and amplitude are best captured by splitting the application domain into sub-domains of influence. Fig.(7-6) illustrates how local and global complexity contribute to the approximation result in the S-parameter approach. The data is taken from a device with a single input component at 1.5 GHz and 4 harmonic output components at 1.5, 3, 4.5 and 6 GHz. It is approximated by a combination of varying numbers of clusters and varying polynomial orders. The purely linear approximation (7-6a) is unable to capture the data characteristics and the fifth-order polynomial model (7-6b) still performs purely. The approximation with five linear models (7-6c) does well, while the approximation with only three second-order polynomials is practically perfect (7-6c).

The model was also tested on data obtained from realistic simulations of an amplifier (fig. 7-4) with two input ports (input 1 and 2 at port 1) and a single output port 2 [VB94b]. There was one input component (1.5 GHz) at input-port 1 and three components (1.5, 3 and 4.5 GHz) at input port 2, causing output components at 1.5, 3 and 4.5 GHz. The model predicts a three-dimensional complex output vector, given a four-dimensional complex input vector. Fig. 7-5 shows the approximation results from a particular simulation. The relative RMS error was at worst 0.001% for the fundamental and 0.5% for the second harmonic; on some of the test sets a significantly better performance was achieved. Local third-order approximations performed best, which was expected given that the data ranges from the fundamental to the second harmonic.

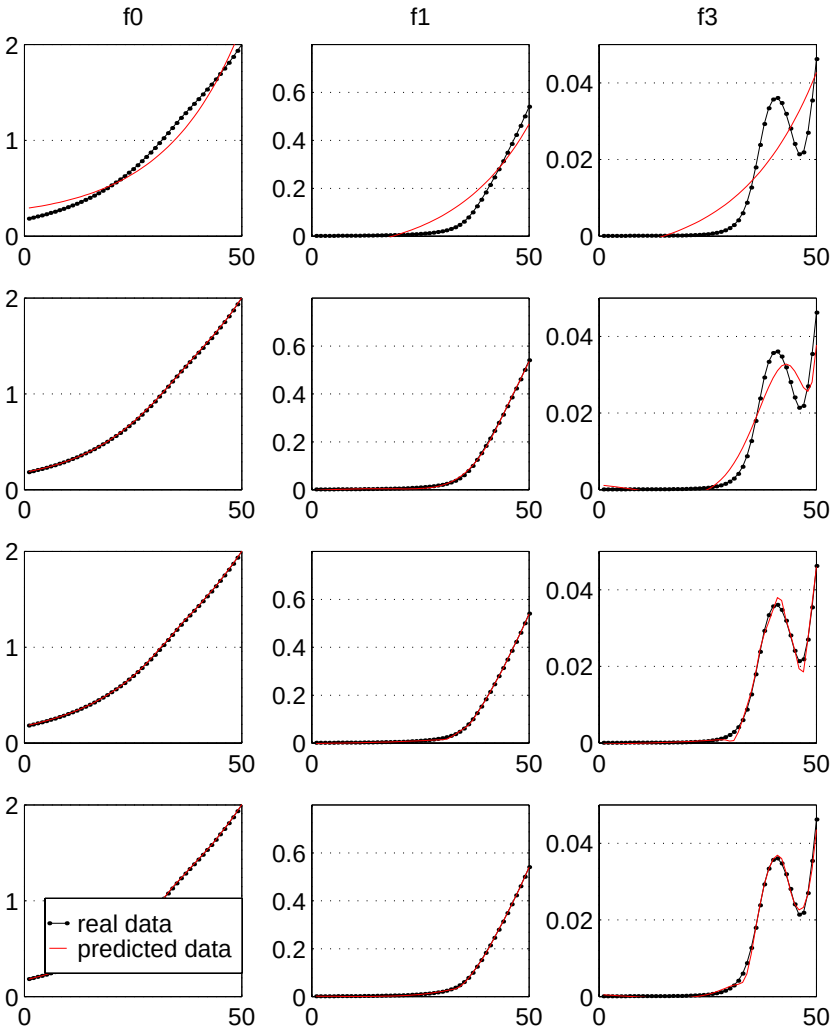


Figure 7-6: Approximation of three (out of four) frequency components. The rows represent the approximation results of (a) a linear model, (b) a fifth-order polynomial, (c) five-cluster/local-linear model, (d) three-cluster/local-quadratic model.

Chapter 8

Sampling

Global sampling has been used for speech synthesis as well as for musical synthesis. It is also referred to as *wavetable synthesis* (section 9.1.3) and relates directly to overlap-add synthesis techniques [CS86, CM89, VMT91]. Sampling reuses recorded pieces of a time series (mostly audio signals) to synthesize a new time series given some control parameters. The new signal has a different shape than the recorded material; however, it is similar in terms of its timbral properties. Patches of data representing different timbre states are overlapped and added. Sampling is a pitch synchronous technique and capable of handling strongly harmonic sound sources. However, its benefit compared to other popular synthesis techniques lies in its potential to model noisy signals.

Sampling replays digitized sounds given a limited number of control parameters. The sequences of audio material are slightly adjusted with respect to pitch and note duration; that is, they are resampled to match a control pitch and are terminated appropriately with respect to the end-note command. They are possibly looped to allow for a long sustain and can be interpolated to match a given attack volume. Sampling relies on some fundamental concepts and signal processing tools, but also on a variety of tricks and ad hoc methods that work well, but lack a deeper meaning. Here we will review the former and mention some of the latter, as long as they have relevance to cluster-weighted sampling. Massie [Mas98] gives an exhaustive review of the algorithms related to sampling.

8.1 Algorithmic pieces and related work

Pitch shifting

Given that pitch is continuous on many instrument families as well as in speech, it is impossible to sample enough examples of an instrument to cover all the possible pitches. However, it is possible to correct slight pitch deviations by pitch-correcting given sound material with respect to the target pitch. Out of the many known techniques for pitch correction [Mas98], software re-sampling stands out as a technique that is not limited in accuracy and achievable targets and doesn't require special hardware.

Re-sampling uses the virtual reconstruction of the band-limited analog signal from the digitized samples. If an analog signal is played back at a speed different from the

recording speed, we hear the pitch shifted[1]. In the analogous case of a digital signal, we need to play samples in between digitized samples or leave out samples from the available data, depending on whether we want to achieve a lower or higher pitch. However, these in-between samples are not readily available and they vary depending on where *in between* they fall. As a first order approximation, one can use a linear interpolation between the nearest samples to the right and left. The result is an understandable signal, but one with significant artifacts due to aliasing and quantification noise. These negative effects can be reduced by up-sampling the signal before its conversion and then down-sampling it again. However, the clean approach to the problem uses the sampling theorem directly and in its pure form.

Given an infinite length digitized signal, we can reconstruct the analog signal as long as the original signal was band-limited to the Nyquist frequency, that is, half the sampling frequency. The time continuous values are perfectly estimated as an infinite sum over `sinc` functions weighted by the digital samples:

$$\begin{aligned} \hat{s}(t) &= \sum_{n=-N}^{N} s(n \cdot T_s)\ h_s(t - n \cdot T_s) \qquad (8.1) \\ h_s &= \min\{F_s/F_s'\} \cdot \mathrm{sinc}(\min\{F_s, F_s'\} \cdot t) \\ \mathrm{sinc}(x) &= \frac{\sin(x)}{x} \end{aligned}$$

where F_s is the sampling frequency of the sequence, F_s' is the target sampling frequency and N is the number of filter coefficients [SP84]. Evidently, N can't be infinite in practice, since we are dealing with finite signals and we need to keep the amount of computation per sample reasonable. However, even a tenth-order filter ($N = 10$) yields good enough sound quality.

If the signal is down-sampled, e.g. pitch is increased, the signal needs to be low-pass filtered with respect to the new Nyquist frequency, since components may be shifted out of the covered spectrum. This is taken care of by the term $\min\{F_s/F_s'\}$ in equ. 8.1. Note that the re-sampling filter functions as a low-pass filter as well. Hence, no additional computation is needed to handle the low-pass filtering.

Sample pitch and target pitch should not differ too much, since large pitch shifts cause the synthesized signal to be perceptively different from the original. One reason for this is that the formant structure and resonance frequencies of the sound source are altered along with the shifted pitch. For example, a harmonic component remains strong/weak although it has been shifted in a register where it should be weak/strong. This effect is particularly strong for speech signals; it is considerable with musical instruments such as the piano or the violin. Hence re-sampling shouldn't be used for shifts bigger than semi-tones, but can easily be used to compensate for shifts due to vibrato. Algorithms that pitch shift the signal without altering the spectral envelope have been introduced in [Len89] and discussed in [BJ95].

[1] The length of the sequence is changed at the same rate, but this doesn't matter for our purpose.

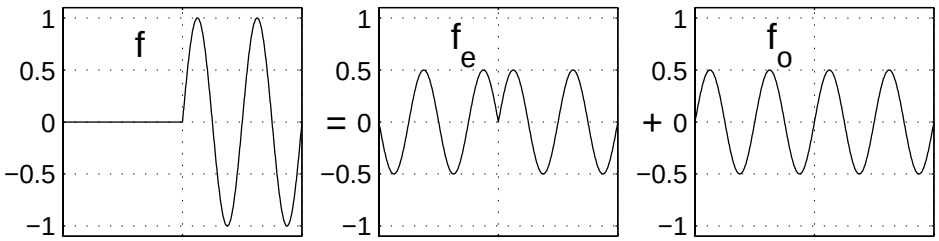


Figure 8-1: Decomposition of a function into and even and odd part

Looping

Sustained notes require looping of the same sample sequence to remain in the same timbre region for an arbitrary duration. In order for transitions to be inaudible, the juncture has to be perfectly smooth and the phases of the signal partials need to be preserved. The former requirement can be achieved by playing the same sequence first forward and then backward in time.

Any real valued signal can be decomposed into an even and an odd part with respect to a mirror point (fig. 8-1).

$$s(t) = s_e(t) + s_o(t) \tag{8.2}$$

The odd and even part are extracted from the signal as

$$\begin{aligned} s_o(t) &= \frac{1}{2}s(t) - \frac{1}{2}s(-t) \\ s_e(t) &= \frac{1}{2}s(t) + \frac{1}{2}s(-t) \end{aligned} \tag{8.3}$$

We decompose the old signal into odd and even parts at the juncture and then replay it, superposing the odd part and the inverted even part. Figure 8-1 illustrates how these elements fit together. This operation assures continuity of all derivatives at the juncture, but not continuity of the signal partials. To assure the latter we need to carefully pick the transition point, which is discussed in the next subsection.

Sequencing

When the timbral characteristics of the sound change, we need to switch sampling sequences. Unfortunately, there isn't a right or best approach to move from one sequence to another (as for many DSP problems in sound processing and computer music), but only a number of heuristic methods that work more or less well. Assuming both sequences have been resampled at the same pitch, we want to achieve a maximum of continuity by picking the best transition point. A measure of smoothness that works well is the inner product between the signals. The two sequences are overlapped in such a way that the correlation

in the overlap-window is maximal. Given $s_1[n]$ and $s_2[n]$ we find the correct offset T

$$T = \text{argmax}_T \sum_{n=1}^{N} s_1[n] \cdot s_2[n+T] \tag{8.4}$$

where N is the length of the correlation window, which should exceed the period of the sequence.

One could be afraid of the computational expense of this procedure. However, the search for the optimal solution can be optimized using additional tricks. For example, consider a violin signal at an early stage of its creation. Say the signal has been picked-up by a dynamic transducer system. The signal will then be phase coherent; the Helmholtz motion is preserved and we see a fairly regular rectangular function (fig. 8-3a). Since the signal has a very clear maximum per period, we only need to correlate signal one and two around this first estimate of a transition point.

Cross-fading

No matter how well two sequences of sound are pieced together, remaining artifacts need to be eliminated by yet another powerful technique to enforce smoothness: cross-fading. Cross-fading is both simple and intuitive. The ending sequence is slowly decreased in volume, while the starting sequence is increased in volume, in such a way that the two scaling factors add up to unity at any moment in time.

$$x[N+n] = x_1[N_1 - N_f + n] \cdot (1 - A[n]) + x_2[n] \cdot A[n] \tag{8.5}$$

where x is the final signal, N is the starting point of the cross-fade, x_1 is the ending sequence, N_1 is the length of the ending sequence, N_f is the length of the fading interval and x_2 is the starting sequence. The fading coefficients $A[n]$ describe a linear ramp, or a squared sinusoidal function, which in addition to being continuous is continuous in all the odd derivatives as well:

$$A[n] = \sin^2\left(\frac{\pi n}{2N_f}\right) \tag{8.6}$$

Halfway through the fade the two signals are equally strong. Some authors suggest to pause after this fist part, before starting the second part of the fade ([Mas98]).

Fading works because the sum of two sinusoidal components of equal frequency but different phase results in a third sinusoid of the same frequency but yet another phase. Since the starting and ending phase are not the same, a phase stretch occurs during the fade. If an appropriately long fading interval and fairly well-matched samples are used, this stretch can be kept small enough to be inaudible.

8.2 Cluster-weighted sampling

Global sampling has been a successful synthesis technique for instruments with a low dimensional control space, such as the piano [Mas98]. However, the technique is less appropriate for instruments with a continuous and complex control space, such as the

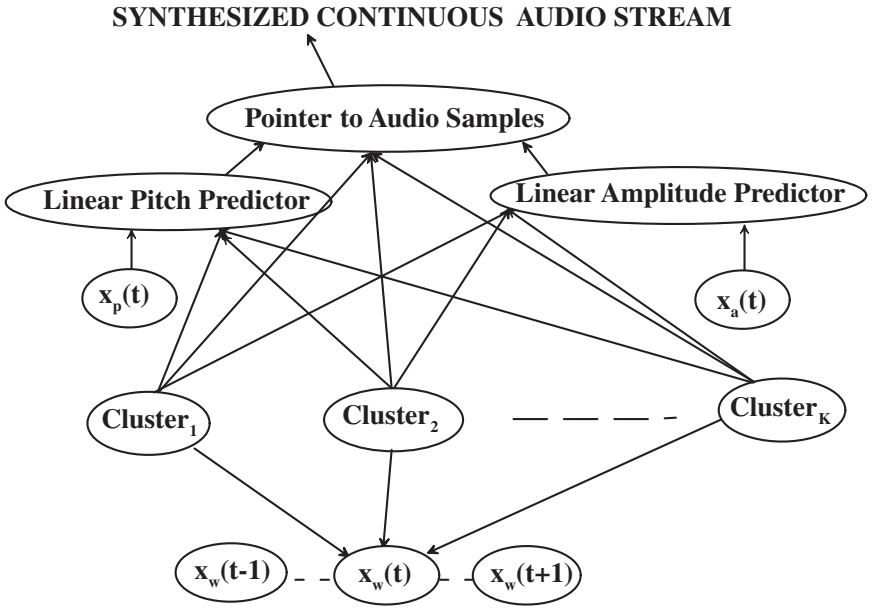


Figure 8-2: Cluster-weighted sampling in a graphical representation. The output models, including a pointer into samples-space, a pitch predictor and an amplitude predictor.

violin. For the latter family of instruments, as well as for speech, the amount of data required to cover all possible playing situations is prohibitive, since control possibilities are essentially unlimited. Cluster-weighted sampling (CWS) overcomes this problem through efficient parameterization of the available sample material. CWS learns how to select the appropriate samples, but also how to predict the parameters needed to reassemble the sound from the raw material.

CWS clusters have multiple output models (experts) covering sample selection, amplitude prediction and pitch prediction. The first expert is a pointer into sample space. The cluster that most likely generated a perceptive or physical control data $\mathbf{x}_s$ takes over and its sequence of samples stands in for the particular playing situation. The cluster is replayed until another cluster becomes more likely and hence takes over with its own samples. The maximum likelihood principle is applied during training as well as replay. Unlike most other local models we have seen so far, the sample selection model uses a *winner-takes-all* approach as opposed to a weighted average approach. The reason for this choice is that sampled sounds do not add up constructively, but would interfere randomly and create artifacts.

During training the sequence is picked to represent a cluster that has the highest posterior probability with respect to this cluster. We assume a sequence to be perceptively represented by its instantaneous spectral characteristics, packaged in the vector $\mathbf{x}$. $\mathbf{x}$ can be interpreted as a pointer into timbre space. The definition and research of timbre has almost become a scientific field of its own, which we don't want to get into here. However, in our case, a relative measure of timbre is sufficient since we only need to define the

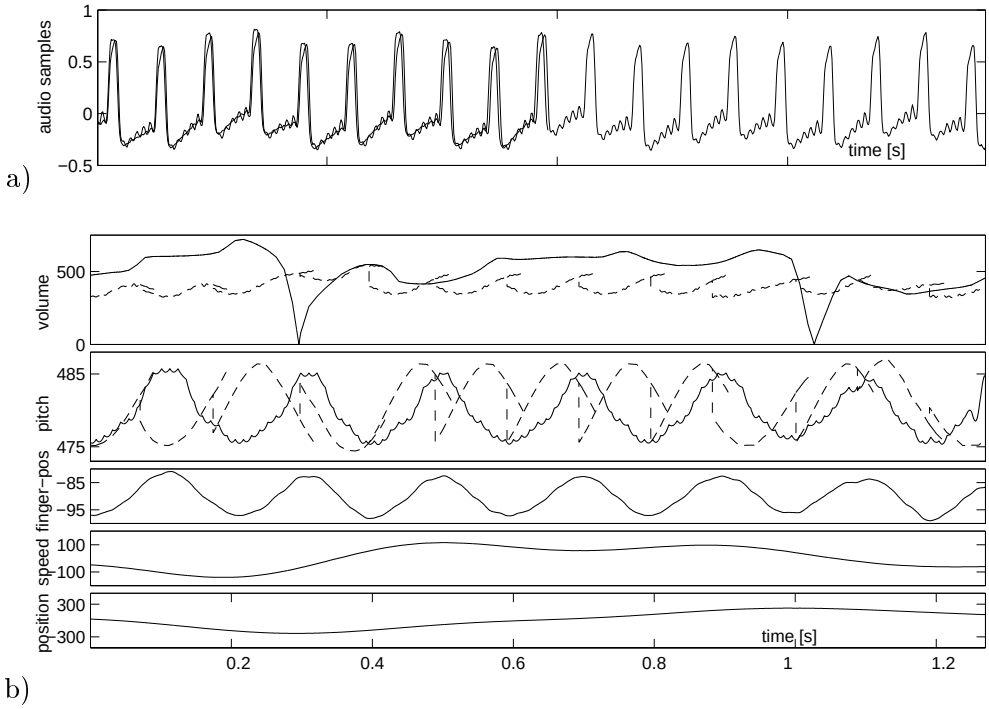


Figure 8-3: Cluster-weighted sampling: a) overlapping samples of the string signal. b) input-output model, *from the bottom*: bow position; bow speed; finger position; predicted out-of-sample amplitude (solid) and given sampled amplitudes (dashed); predicted out-of-samples pitch (solid) and given sampled pitch (dashed); the doubled dashed lines indicate overlapping sample windows: the old window is slowly faded out while the new window is faded in, in such a way that the total weight of data adds up to unity at any given moment.

closeness of two sounds, not their absolute timbral properties. Closeness can be defined in terms of the Euclidean norm in a feature vector space. In the simplest approach, we choose the magnitude of the harmonics as the timbral characteristics and the components of vector $\mathbf{x}$. A more elaborate approach summarizes the spectral information in a measure of brightness [Wes79]. The Brightness $\mathcal{B}$ is defined as

$$\mathcal{B} = \frac{k}{K} = \frac{\sum_{i=2}^{K/2} f_i \cdot A_i}{\sum_{i=2}^{K/2} A_i} \tag{8.7}$$

where the A_k are the coefficients of the short-term Fourier transform and K is the size of the transform. This measure is closely related to the spectral centroid

$$\mathcal{C} \quad = \quad \frac{2k}{K} \tag{8.8}$$

$$\sum_{k'=2}^{k} |A_{k'}| \quad = \quad \sum_{k'=k+1}^{K/2} |A_{k'}| \quad .$$

Given a target vector $\mathbf{y}_{\mathcal{S}}$ that has been extracted from the sequence $\mathcal{S}$, the sequence that has most likely been generated by a particular cluster c_k, is picked to represent this cluster.

$$\mathcal{S} = \text{argmax}_{\mathcal{S}} p(\mathbf{y}_{\mathcal{S}} \mid c_k) \tag{8.9}$$

where $p(\mathbf{y}_{\mathcal{S}} \mid c_k)$ is the usual Gaussian distribution. For prediction, this process is inverted. The cluster c_k is chosen that best represents the current input or state of the system. The representative sequence of this cluster stands in for the predicted samples,

$$\mathcal{S} = \mathcal{S}_{c_k} | c_k = \text{argmax}_{c_k} p(c_k|\mathbf{x}) \quad . \tag{8.10}$$

The second output model is a pitch predictor. Given some control input $\mathbf{x}_{\text{pitch}}$ a local linear model predicts the appropriate pitch at any moment in time.

$$\text{pitch}(t) = \frac{\sum_{k=1}^{M} \mathbf{a}_{\text{pitch}}^T \mathbf{x}_{\text{pitch}} p(\mathbf{x}_s|c_k)\ p(c_k)}{\sum_{k=1}^{M} p(\mathbf{x}_s|c_k)\ p(c_k)} \tag{8.11}$$

The third output model predicts the instantaneous scaling factor (volume) given the control vector $\mathbf{x}_{\text{vol}}$. These are once again simple locally linear predictors.

$$\text{vol}(t) = \frac{\sum_{k=1}^{M} \mathbf{a}_{\text{vol}}^T \mathbf{x}_{\text{vol}} p(\mathbf{x}_s|c_k)\ p(c_k)}{\sum_{k=1}^{M} p(\mathbf{x}_s|c_k)\ p(c_k)} \tag{8.12}$$

The samples selected for synthesis almost certainly won't match the desired pitch and volume. Therefore the selected sequence is resampled with respect to the predicted target pitch and rescaled with respect to the target volume. Re-sampling is done in real time according to expression (8.1). Likewise, we rescale the sequences with respect to the predicted target volume, which is as simple as

$$s[n] = s_r[n] \frac{v_t[n]}{v_r[n]} \tag{8.13}$$

where s is the final output, s_r is the prerecorded sample, v_t is the instantaneous target volume and v_r is the instantaneous prerecorded volume.

CWS uses high-level information about the timbral properties of the modeled sound. We need pitch, volume, and timbre to instantaneously label, parameterize, and correct the sample material. These properties may seem difficult to ascertain. Pitch tracking, for example, has not been solved in full generality, although, there are a number of algorithms that work well in most cases [Met96] [Lar98] [QM98]. For many of these techniques, a little bit of extra information, such as a rough *a priori* estimate of the pitch window, helps a great deal. Often we do have this extra bit of information, since we pick or explicitly record the sequences that we then use for modeling. Likewise there is a number of slightly different methods to extract the instantaneous volume of a signal. One approach uses the instantaneous energy of the signal, another one uses the peak value of the signal within the

vicinity of the observation. Yet another approach uses the instantaneous variance of the signal which tends to be closer to the perceived energy than the signal energy. As opposed to the instantaneous volume, loudness takes into account the psycho-acoustic properties of the human ear to evaluate the perceptional effect of a signal.

An important aspect of CWS is the sequencing of pieces of audio, when cluster sequences have to be looped or transitions from one cluster/sequence to another have to happen. We choose to match samples by minimizing the least square error between the old and the new samples as demonstrated in equation 8.4. Additionally, we fade out the old sound and fade in the new sound using a Hanning window overlap-add as in equation 8.5.

We can increase the resolution of our fit allowing for non-integer alignment of sounds. Since the audio sequences are resampled anyway, this does not increase the computational load of the overall algorithm. The success of the overlap-add depends on the length of the permissible fading interval and on the character of the sound. Fig. 8-3 shows the overlap of two highly phase-coherent pieces of the string signal of a violin describing a Helmholtz motion. For the Helmholtz motion, harmonic partials line up nicely with the fundamental so that discontinuities are a minor problem. However, the sound signal loses its phase coherence after the filtering through the bridge and through the resonant body of the instrument. Alignment becomes more difficult then.

Cluster-weighted sampling is applied and tailored to musical synthesis in Part III of this thesis.

Part III

Data-Driven Modeling of Musical Instruments

> *Geschmack* ist das Beurteilungsvermögen eines Gegenstandes oder einer Vorstellungsart durch ein Wohlgefallen, oder Mißfallen, *ohne alles Interesse*. Der Gegenstand eines solchen Wohlgefallens heißt *schön*.
> I. Kant, *Kritik der Urteilskraft*, B17.[2]

In his third critique, the *Critique of Judgment*, Kant defines and explains aesthetical judgment, e.g. he explains the concepts of beauty and taste. Following philosophers like Aristotle, Kant is not the first to decompose knowledge into three disciplines, i.e. rational reasoning, moral behavior, and aesthetical judgment. The author of this thesis is intrigued that the division of his own thesis into an initial part with theoretical concepts, a second part concerning practical implementation, and a third part on aesthetical goals parallels the themes of the three critiques.

This thesis work has all along been driven by the prospect of building a digital copy of a violin: a digital Stradivarius. Motivated by insights from dynamic systems theory, it has been our goal to reconstruct the dynamic behavior of the violin to the extent that a player would be willing to switch from an acoustic to a digital instrument. Although the developed machine-learning and signal-processing algorithms apply to problems in many scientific and engineering domains, the challenging application of digitally replicating violins has remained our benchmark test.

The violin is in many ways an ideal test device. (1) As a physical object, it is well defined in that it has a clear input space, the player's actions, as well as a clear output signal, the sound pressure. Once we are able to predict a system as complex and well engineered as the violin, we believe we'll be able to model many other devices as well. (2) A digital model of the violin comes with a simple but compelling error metric, which can be summarized in the statement that the model is as good as it sounds. From a player's perspective, the situation is a little more complex since the quality of an instrument is also measured in terms of its ease of use and its projection qualities. However, in playing the model, the skilled violinist easily evaluates these criteria. (3) It is, on the other hand, extremely challenging to quantify the difference between great and not so great sound. Establishing a valid and general theory regarding the tone quality of an instrument is probably as difficult as the artificial synthesis of sound itself. (4) Lastly, the violin is an object that people care about. Throughout its long history, violin makers have tried to improve its design, composers have written beautiful music for it, violinists have studied and performed it, and listeners have enjoyed its playing. Even those who dislike violin music or today's concert practice still recognize the aura of perfection and harmony around a master instrument.

[2] *Taste* is the ability to judge an object, or a way of presenting it, by means of a liking or disliking *devoid of all interest*. The object of such a liking is called *beautiful*.
I. Kant. *Critique of Judgment*. English translation by W.S. Pluhar.

Chapter 9

Related work in musical synthesis

9.1 Synthesis algorithms

9.1.1 Physical modeling

In many ways, physical modeling is the most fundamental and most intuitive way of emulating the behavior of an instrument on a digital machine. Physical modeling reconstructs the global behavior of the instrument by means of simulating the mechanical properties of the instrument.

Physical models retain the natural expressiveness of the acoustic instrument. The most important aspect of this feature is the ability of building models that extrapolate to any playing technique of an instrument. The number of different things one can do with a violin is virtually infinite, even if one doesn't consider the atypical *smashing the instrument over a chair*. Since a physical model implements the dynamic response in terms of first-principle equations, the model has an answer for any of those inputs.

The numeric techniques of approximating physical models on digital machines range from simple finite difference models to digital waveguide models. The latter methodology is important since it overcomes computational limitations of most other physical modeling approaches. The former methodology is interesting because of its transparent structure. Finite difference models approximate the underlying differential equation of the dynamic system. The easiest approximation uses Euler's method, which will be demonstrated here for a bowed string model.

Finite difference models discretize both time and space. The first time and space derivatives are approximated as [Smi98]

$$\dot{y} \approx \frac{y(t,x) - y(t-T,X)}{T} \tag{9.1}$$

and

$$y' \approx \frac{y(t,x) - y(t,x-X)}{X} \tag{9.2}$$

where X is the length of a spatial element and T is a discrete time step.

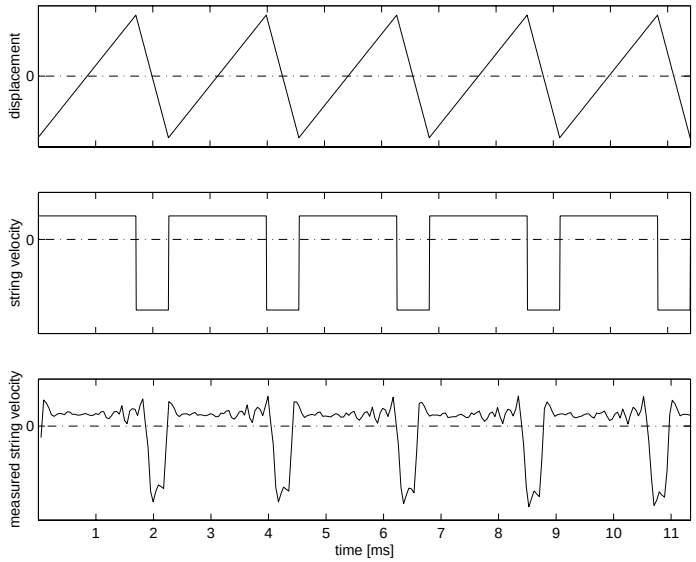


Figure 9-1: Helmholtz motion. *top:* ideal Helmholtz motion, string displacement. *center:* ideal Helmholtz motion string displacement. *bottom:* violin string signal measured with a dynamic pickup showing a low-pass filtered rectangular function.

Taking the derivatives of these two expressions we obtain the second order derivatives,

$$\begin{aligned} \ddot{y} &\approx \frac{y(t+T,x) - 2y(t,x) + y(t-T,X)}{T^2} \\ y'' &\approx \frac{y(t,x+X) - 2y(t,x) + y(t,x-X)}{X^2} \quad , \end{aligned} \tag{9.3}$$

shifting everything forward in time in order to avoid delay errors.

The wave equation of the ideal string is given by

$$Ky'' = \epsilon\ddot{y} \tag{9.4}$$

where K denotes the string tension, ϵ denotes the linear mass density and y is the string displacement as a function of x. We cut a string of finite length into N discrete pieces, and assume the supports at both ends are rigid. Plugging the approximation for the derivatives into (9.4) gives

$$K\frac{y(t,x+X) - 2y(t,x) + y(t,x-X)}{X^2} = \epsilon\frac{y(t+T,x) - 2y(t,x) + y(t-T,x)}{T^2} \quad . \tag{9.5}$$

Solving for $y(t+T,x)$ we find the finite difference update for each element at time $t+T$,

$$y(t+T,x) = \frac{KT^2}{\epsilon X^2}[y(t,x+X) - 2y(t,x) + y(t,x-X)] + 2y(t,x) - y(t-T,x). \tag{9.6}$$

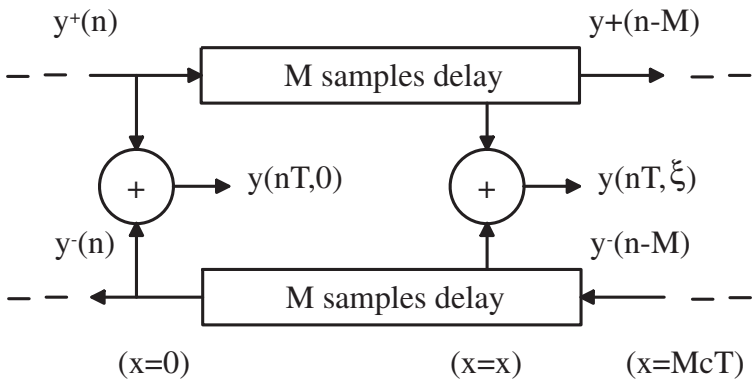


Figure 9-2: Delay line model after Smith [Smi98, p.429].

which simplifies to

$$y(n+1,m) = y(n,m+1) + y(n,m-1) - y(n-1,m) \quad . \tag{9.7}$$

using $T = 1$, $X = (\sqrt{(}K/\epsilon))T$, $t = nT$ and $x = mX$. This is the generic update for the free vibrating string. In order to create a realistic system, we need to add a dissipation term. The restoring force in the vibrating string is proportional to the forth derivative and yields the dissipative wave equation:

$$\epsilon\ddot{y} = Ky'' = \kappa y'''' \tag{9.8}$$

with

$$\kappa = Y\pi r^4/4 \tag{9.9}$$

where Y is Young's modulus and r is the string radius.

The plugged string is practically as simple as the free string: one of the string elements is pulled away from its rest position by a fixed displacement and then released. The string is free again. The boundary conditions for the bowed string are more complicated. They involve the stick-slip interaction between bow and string, which leads to the famous Helmholtz motion of a bowed string (fig. 9-1). The bow sticks to the string when the vertical relative force is lower than the frictional force between bow and string. In this case the bow enforces its velocity on the string. When the restoration force of the displaced string becomes too strong, the string slips and the bow acts as a damping force. This repeated pattern causes the string to travel in a circular square wave around its full length.

The finite difference implementation has two update rules depending on whether the bow is in the sticking or the slipping state. If the bow is in the sticking state, the velocity of the element that is touched by the bow is set to the velocity of the bow. If the pulling force becomes stronger than the friction force $F_f = f \cdot P$, where f is the friction coefficient and P is the bow pressure, the system changes into slipping mode. The bow-node is now

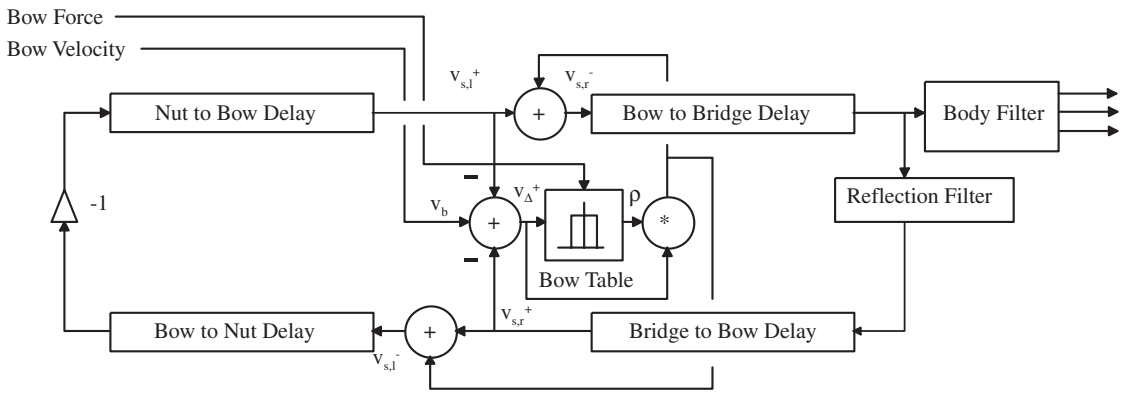


Figure 9-3: Violin waveguide model after Smith [Smi98, P.464].

updated, taking into account the frictional force caused by the bow. If the relative speed between bow and string becomes smaller than a sticking threshold, the system switches back into sticking mode.

For the purpose of explaining waveguide models we go back to wave equation (9.4). As can be easily verified, the solution satisfies

$$y(x,t) = y_r(t - c/x) + y_l(t + x/c) \quad , \tag{9.10}$$

where y_r is a wave component traveling to the right and y_l is a wave component traveling to the left with speed $c = \sqrt{K/\epsilon}$ [dA47, Smi98]. For a digital implementation, this equation needs to be sampled in time and space. The two components are traveling along a delay line and are simply added together at the pickup point. This is the origin and motivation for digital waveguide models.

Fig. 9-2 illustrates the component waves traveling in opposite directions, using a velocity wave representation. Fig. 9-3 illustrates the implementation of a violin waveguide model. The string delay line pairs contain the left-traveling and right-traveling waves going into and coming from the bow. The nut-end (finger-end) and bridge-end of the string are implemented as delay elements modeling the losses and delays at either end of the string. To first order these elements are taken to be rigid reflection elements that equal -1. The numerical waveguide implementation represents delay line and reflection elements in a single filter element. The string velocity at any point equals the sum of left-going and right-going velocity elements and the bow string interaction is again implemented for the cases of sticking and slipping (see [Smi98] for details). Waveguide models can be shown to be equivalent to finite difference models in the loss-less case. In addition to being computationally more efficient, they also provide more flexibility in terms of including second order effects into the equations.

As was mentioned before, physical models provide a lot of flexibility because we can change the model parameters and still obtain reasonable physical behavior. We can create instruments that obey physical laws but have no equivalent in reality. However, in turn, it is very difficult to find the right parameters to emulate a given instrument. While it is

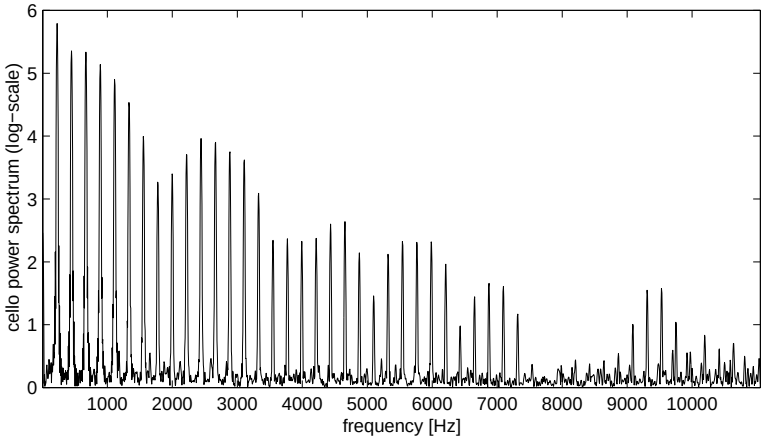


Figure 9-4: Typical short term spectrum of a cello sound.

possible to infer some filter parameters from observed data, there is no general systematic search to find all the model coefficients from sampled data. The filter parameters of the bridge and resonating body are hidden in the sound of an acoustic instrument. However, there is no immediate connection between bow-string interaction parameters and a recorded violin signal. Also, given a basic model of a violin there is no systematic search for the parameters that distinguish a Guaneri from a Stradivarius instrument other than trying out combinations of parameters in a very high-dimensional space.

9.1.2 Sinusoidal analysis/synthesis

Physical modeling uses a bottom-up approach. We start with first principles and we hope to achieve accurate sounds by getting the physical details right. In contrast, sinusoidal analysis/synthesis uses a top-down approach. It relies on the predominantly harmonic structure of most musical instruments and physical sound sources. For example, almost all the energy of a violin is contained in its harmonic partials and hence the signal can be efficiently represented as a finite sum of harmonic sine waves with slowly time-varying frequencies and amplitudes (fig. 9-4). The same is true for speech, although the contribution of unvoiced sound elements is higher. Unvoiced sounds don't have a harmonic structure but can be approximated by a sum over sine waves without coherent phase structures. Alternatively noise can be added in form of a stochastic process.

Additive synthesis parameterizes a quasi-periodic sound signal in terms of

$$s(n) = \sum_{k=1}^{K} A_k \cos(k\omega n + \Phi_k) \quad , \tag{9.11}$$

where n is a discrete time index, A_k and Φ_k are amplitude and phase of the partials k and K is the number of partials. This representation is both modeling assumption and synthesis instruction. The model finds peaks of the power spectrum and estimates

the instantaneous frequencies and amplitudes of the underlying sine wave, neglecting the noise in between partials. To synthesize sound, amplitude and frequency-modulated sine waves are summed together, interpolating frequencies and amplitudes in between analysis frames.

The model parameters are typically inferred from a short-term Fourier transform (STFT) applied to the audio signal at time intervals on the order of 10 ms. The sample windows can be as long as 100 ms, depending on the register and the character of the signal or music. The sample sequences used for the STFT need to be windowed, e.g. with a Hanning or Kaiser window, in order to avoid artifacts. Zero padding the edges of the window helps achieving a smooth frequency domain signal.

In the simplest approximation the partial amplitudes are taken to be the amplitudes of the bins, and the frequencies are taken to be the center frequencies of these bins. This approach is known as *phase vocoding*. It has been used particularly for pitch and timescale modifications of audio signals. In these applications, either the spectral content remains the same and time is stretched (timescale modification), or the frequencies are modified and time remains the same (pitch shifting). The phase vocoder can be implemented in a time continuous way as a filter bank that maps any incoming component into a transformed outgoing component.

The phase vocoder has serious limitations regarding sound quality. For example, it is assumed that only a single partial falls into the path-band of a filter. Also, considerable artifacts are created when partials of natural sounds move in and out of a filter bank or a STFT bin. Furthermore, consider a 440 Hz signal and a 512 point STFT. The phase vocoder frequency estimate for this signal will be 430.6 Hz and the amplitude estimate will be off by 8%, due to the fact that the peak is not centered but close to the border of the frequency bins. Fortunately, there are much more accurate estimates for frequencies and energies of partials using either adjacent STFT frames or adjacent frequency bins to estimate a center bin. This leads to the explicit modeling of sound partials rather than the modeling of a filter bank. The model representation is driven by the idea that a quasi-periodic sound is composed by a limited number of harmonic partials and possibly additional noise components.

Very accurate frequency estimates can be obtained by looking at the phase increments in between Fourier frames separated by a single time step [BP93]. If

$$\begin{aligned} \phi(k,n) &= \arctan\left\{\frac{\mathrm{Im}\{X[k,n]\}}{\mathrm{Re}\{X[k,n]\}}\right\} \\ \phi(k,n-1) &= \arctan\left\{\frac{\mathrm{Im}\{X[k,n-1]\}}{\mathrm{Re}\{X[k,n-1]\}}\right\} \end{aligned} \tag{9.12}$$

are the phases of the k-th bin at time n and $n-1$, the estimator for the instantaneous frequency at time n is

$$f(k,n) = \frac{\omega(k,n)}{2\pi} = \frac{1}{2\pi}[\phi(k,n) - \phi(k,n-1)] \tag{9.13}$$

Alternatively, precise frequencies can be estimated from coefficients, adjacent to the

carrier bin, using a second order polynomial fit of the spectral curve [SS87]

$$F(z) = a(z-p)^2 + b \quad . \tag{9.14}$$

If k is the index of the carrier bin, we set $z(k) = 0$, $z(k-1) = -1$, and $z(k+1) = 1$. $F(-1) = \alpha$, $F(0) = \beta$, and $F(1) = \gamma$, where α, β, and γ are the amplitudes of the correspondent bins. The LMS fit yields,

$$\begin{aligned} p &= \frac{1}{2}\frac{\alpha-\gamma}{\alpha-2\beta+\gamma} \\ k' &= k+p \\ f^* &= \frac{f_s k^*}{N} \end{aligned} \tag{9.15}$$

where k' is a real valued frequency bin, i.e. it is a real value in $[k-1 \quad k+1]$, f^* is the corresponding peak frequency, f_s is the sampling frequency and N is the number of coefficients. The estimated peak height is

$$F(p) = F(k^*) = \beta - \frac{1}{4}(\alpha-\gamma)\ p \quad . \tag{9.16}$$

Given the spectral parameters, the reconstruction of the original sound relies on the notion of frame-to-frame peak matching. Partial components in natural sounds such as speech and music tend to change continuously. For the purpose of analysis, the continuity assumption helps identifying deterministic as opposed to random components. For synthesis, this continuity needs to be preserved in order to recreate the sound accurately and to avoid annoying artifacts. Unfortunately, there is no fundamental strategy of identifying the beginning and the ending of a partial, but a number of heuristics that can be applied. One tries to maximize the overall continuity in the system while at the same time avoiding unrealistic, that is, too big steps in frequency. The reconstruction algorithms are commonly referred to as the *death* and *birth* algorithm, since they deal with a rapid change in number and location of spectral peaks [MQ85, MQ86].

Many quasi-periodic sounds have significant noise components. Speech sounds consist of voiced and unvoiced syllables. Violin output is largely harmonic, but can be very noisy during the bow change and for certain bowing techniques, e.g. spiccato. It has been shown that these components can be taken care of by adding the noise that was not captured by the sinusoidal model [MQ86, QM98]. The procedure is as follows: a sinusoidal analysis and re-synthesis is performed; the resulting sound is subtracted from the original, and the spectrum of the difference signal corresponds to the missing noise. Since it is mainly noise, it can be modeled as a stochastic process and added on top of the sinusoidal signal.

For this approach to work and for the difference signal to be meaningful, the phases of the reconstructed sinusoids need to match the phase of the original signal. This can be achieved using a cubic polynomial to interpolate in between frames. The approach maximizes smoothness of phase increments given the boundary conditions of starting and ending frequencies and phases [MQ86, QM98].

In a slightly different approach, known as *Spectral Modeling Synthesis* (SMS), Serra and Smith [Ser89, SS90] compute the residual component in the magnitude frequency

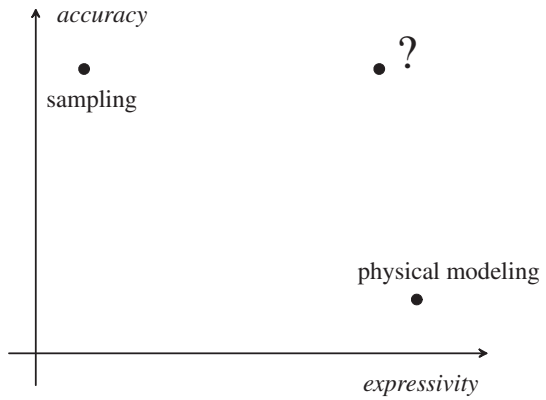


Figure 9-5: Synthesis trade-offs after D. Massie [Mas98]. In reality the diagram is populated with more than the two examples of existing techniques. However, sampling and physical modeling nicely describe the edges of the space.

domain. This approach avoids phase matching, since the magnitude spectrum ignores phase completely. In exchange, the STFT of the sinusoidal components is computed and compared to the original spectrum. The residual is transformed back into the time domain using an inverse FFT and then added onto the sinusoids. SMS claims to reconstruct the full signal, including deterministic and stochastic parts, but its perceptual performance suffers from a lack of homogeneity between the two components. The blending of stochastic and deterministic components appears to be not as seamless as one would wish.

FM synthesis is a variant of spectral modeling [Cho73]. A complex spectrum, meaning a superposition of many sinusoids, is created by means of modulating a single sinusoid. Although FM synthesis does not provide explicit control over the parameters of each signal component, it has been the predominant synthesis technique for most of the early electronic synthesizers, mostly due to its computational efficiency (section 9.1.4).

9.1.3 Wavetable synthesis

Wavetable synthesis (section 8.1) has become the predominant technique in modern commercial synthesizers. Dana Massie [Mas98] points out that the main reason for this overwhelming predominance is the ease with which libraries of sounds can be built. Similar to photography, which allows archiving of pictures of arbitrary scenes very quickly, sampling simply records a given audio scene and adds it to the archive. While other popular techniques, e.g. physical modeling, require tedious model adjustment and parameter tuning for every new instrument considered, sampling requires not much more than a recording device, the acoustic instrument, and a player.

Wavetable synthesis is the paradigm example for synthesis technology that is characterized by high sound accuracy, but very little flexibility and expressive freedom (fig. 9-5). It sounds good, because it simply reuses recorded and digitized sound. Yet it is not expressive since there is no meaningful way to change the samples based on changed control

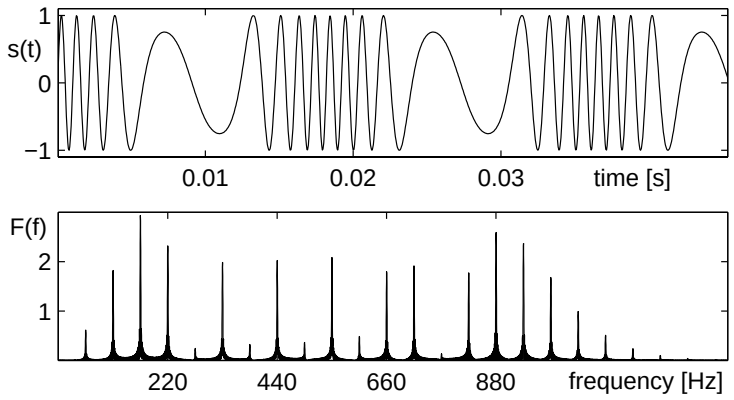


Figure 9-6: *top:* FM modulated signal (equ.9.17 using $A = 1$, $f_C = 440Hz$, $I = 10.0$, and $f_M = 55$. *bottom:* power spectrum of the signal (Bessel-function of the first kind).

parameters. The method works well for instruments with low-dimensional control space, such as pianos. Pianos and organs alike are *left alone* in between the activation of the key and the release. Hence the space of sounds is limited and can easily be covered by a reasonable range of sampled sequences.

While early implementations of Wavetable synthesis suffered from heavy interpolation of a few samples, this is less and less of an issue since memory has become cheap. Nowadays, any single key of the piano can be sampled many times. Additionally, modern MIDI sampler offer a choice between different kinds of pianos, ranging from Bösendorfer and Steinway grands to honky-donk saloon pianos.

9.1.4 More synthesis techniques...

FM-synthesis(FM) has been the predominant commercial synthesis technique in the 70s and 80s.[1] The success of FM has been due to the fact that one could create very complex sounds at very little computational expense. Unlike physical modeling or other DSP approaches, FM is not derived from physical laws nor is it derived from signal processing ideas close to the physical system [Roa95].

Simple FM or **Chowning FM** uses a carrier oscillator modulated by a modulator oscillator.

$$s(t) = A\sin(2\pi f_C t + I\sin(2\pi f_M t)) \tag{9.17}$$

where A is the carrier amplitude, f_C is the carrier frequency, I is the index of modulation, and f_M is the modulating frequency. If $\frac{f_C}{f_M}$ is an integer ratio, this formula generates a harmonic spectrum. If the frequency ratio is rational an inharmonic spectrum is created. The amount of frequency modulation, i.e. the number of sidebands, is controlled by

[1] John Chowning from Stanford University is commonly known as the inventor of FM. Yamaha licensed his patent in the mid 70s [Cho73] and developed the synthesizers GS1 and DX7. The commercial success of the latter model remains unbeaten.

the index of modulation I, i.e. the amplitude of the modulating oscillator. The power spectrum of a periodically modulated oscillator is described by the Bessel functions of the first kind $J_n(I)$, where I is the index of modulation and n indicates the sideband (fig. 9-6). Therefore the signal can also be represented in terms of partials,

$$s(t) = \sum_{n=-\infty}^{n=+\infty} J_n(I) \cdot \sin(2\pi[f_C \pm n \cdot f_M]\ t) \quad . \tag{9.18}$$

The rest is a bag of specific tricks, each of which has its justification in a specific sound effect or instrument. **Multiple Carrier FM** superposes multiple carriers modulated by a single modulating frequency to generate spectra with a complex formant structure. For **Multiple-Modulator FM**, a single carrier oscillator is modulated by multiple oscillators, either in parallel, e.g. the modulating signals are added, or in series, e.g. each modulating oscillator is the argument of the oscillator one level up. **Feedback FM** is good for generating a controlled increase of partials and smooth partial amplitudes [Roa95].

Like FM, **Waveshaping Synthesis** is a fairly adhoc technique to construct complex, dynamic spectra at little computational expense. It sends an arbitrary input signal through a distortion filter. The filter has the form of a lookup table that maps every time signal sample x into a unique output sample w. The function W is called the shaping function. The amplitude of the driving signal X is used to select domains of the shaping function, rather than to fix the amplitude. Therefore Waveshaping Synthesis requires an additional amplitude normalization module to control the dynamics independently from the spectrum [Ris69, Arf79, LeB79].

Granular synthesis primarily targets the creation of new sounds. As a tool for composers, it provides the means for referencing and reusing given sounds in a new context, but it is not meant to literally recreate a given source in the digital medium. The fundamental building blocks of Granular Synthesis are the so-called Sonic Grains, that is, pieces of sound weighted by a window envelope. Many different grains, overlapping or not, form a complex sound. It has been shown that given arbitrarily simple grains, the granular representation can approximate any arbitrary signal [Roa95, Gab46, Gab47, Bas80, Bas85]. The grain envelopes vary from Gaussian shapes to piece-wise linear windows and they vary in length. A major concern is the parameterization of the grains, in order for composers to be able to use them efficiently. **Fourier/Wavelet Grids** use the STFT or wavelet transform to establish a grid of grains corresponding to spectral bands and different points in time. While the STFT typically uses fixed window length, bandwidth and frame rates, wavelet transforms find these parameters dynamically. Perfect-reconstruction wavelet-packets are designed to decompose a signal in frequency and time, varying window length and bandwidth given some entropy measure (section 4.1.2) [Sai94]. **Pitch Synchronous Granular Synthesis** analyzes recorded sound with respect to pitch and spectral content. For synthesis, a pulse train at the right pitch drives a linear filter which models the spectral content of the original sound (section 6) [DPR91]. **Quasi synchronous Granular Synthesis** uses short grains characterized by a carrier frequency and an envelope with variable pauses in between. The carrier frequency determines the pitch of the sound, while the fast changing envelope of the grains creates the sideband [OHR91]. **Asynchronous Granular Synthesis** allocates grains statistically over a defined period of time, creating

so called clouds. Clouds are parameterized by starting time, duration, grain length, density of grains, bandwidth or pitch of the cloud, amplitude envelope of the cloud, and grain waveform [Roa91, Roa95].

In his dissertation Metois introduces a new synthesis technique which he calls **Psymbesis** (for *Pitch Synchronous Embedding Synthesis*) [Met96]. Much like the author of this thesis, Metois was inspired by the embedding theorem and the prospect of developing a synthesis method that would have the generality and flexibility of a general nonlinear model [Cas92]. Metois overcomes the intrinsic stability issues of nonlinear recursive predictors by setting up a framework that forces the signal to follow a recorded pitch-synchronous period. He defines a vector of perceptual control parameters, including pitch, loudness, and brightness, clusters this control space, and assigns periods of sound to each cluster. Each cluster period is resampled with respect to a reference pitch and is characterized by a mean and a variance of each sample. For synthesis, the chosen period is represented in a low-dimensional lag-space rotating around on the cycle. Depending on the sample variance of the output, samples are slowly pulled back to the mean values ensuring that the transition between different sampled periods happens smoothly. The periods are resampled at the desired pitch and adjusted for the desired loudness [Met96].

Recently researchers have attacked the problem of combining different synthesis methods. From figure 9-5 we learn that it is very difficult to develop synthesis methods that provide good sound quality along with expressive freedom. Since most techniques tend to favor one or the other aspect, it can be beneficial to dynamically switch between techniques depending on the specific needs. Peeters and Rodet combine the benefits of sinusoidal additive synthesis (S/A) and of Time-Domain Overlap-Add (TD-OLA) and TD-Pitch-Synchronous OLA (TD-PSOLA) to what they call **SINOLA** [PR99]. S/A is very accurate for signals that can be considered stationary over some low number of fundamental periods and that have predominantly harmonic content. TD-PSOLA is preferable for non-stationary and non-harmonic sounds. One of the key issues is how to find and define the switching points. [PR99] defines a separate feature vector from a local spectral analysis, based on which method A, method B, or both at the same time are used. The other issue concerns the blending of different approaches in such a way that transitions don't cause artifacts and that the sound appears to be coming from the same source.

9.2 Connectionism and musical applications

Connectionism is a young discipline, mostly because one needs fast computers to solve large optimization problems such as the parameter search for an Artificial Neural Network (ANN). However, ever since connectionist networks have been around, there have been attempts to use the newly invented tools for applications in music. Given that these algorithms were in part inspired by the design of the human brain, it appeared natural to use their power to solve some of the many problems in music cognition. As unexplained as the predictive power of ANN's originally was, equally unexplained were cognitive phenomena such as analysis, evaluation, and creation of music.

Todd and Loy [TL91] reprint some of the early work in the category of music cognition and connectionism. Most of the contributions in the volume conclude with the statement that the proposed networks were really promising but that there hasn't been sufficient

experimental data around to verify findings and theory experimentally. More recent work includes the contribution of Hörnel, who has been working on tonal analysis, harmony and synthesis of high-level musical structure [HR96, HD96, HM98]. In [HR96, HM98] a system is presented that predicts harmonic structure given a melody, hence basically acting like a baroque figured base player harmonizing a melodic line. The system is also able to recognize and classify different harmonization styles based on the concept of expectation and surprise. The authors use a combination of symbolic analysis, trained neural networks, and simple rule-based algorithms, such as *don't use parallel fifths*. In [HD96, HM98] a system is presented that produces melodic variations in the style of baroque composers. A system of neural networks working in parallel and at different timescales harmonizes a choral melody and then improvises variations on any of the choral voices.

Increasingly, pattern recognition techniques are being used for classification of music, style, and instruments. Brown [Bro99] recognizes instruments using Gaussian mixture models. Martin [Mar99] extends the idea into a system that classifies sounds with respect to a number of different instruments and instrument families. He is able to teach machines to classify as accurately as human listeners as long as a single instrument is recorded. Scheirer [Sch00] worked on the notion of similarity of sounds. His system compares arbitrary music in terms of perceptual properties. He defines measures of closeness and evaluates them with state of the art pattern recognition technology. The latter problem has also become a key topic for a number of start-up companies.

Connectionist approaches to musical synthesis are not as common. Wessel et al. presented a synthesis model derived from an ANN that is surprisingly close to our approach [WDW98]. The fundamental elements of their approach are a psycho-acoustic high-level characterization of sound, a sinusoidal analysis-synthesis, and a predictor function. A database of recorded sounds is analyzed and parameterized with respect to pitch, loudness, and brightness as well as to spectral information in the form of frequencies and amplitudes. The former psycho-acoustic parameters serve as inputs to the feed-forward network, whereas the latter parameters serve as outputs. The goal is to train the network to represent a specific instrument. When the model is given arbitrary psycho-acoustic control data, the network maps this control to the sound of the instrument it has been trained with. The framework is tested with an ANN using one hidden layer and with a memory-based network. It was found that the ANN model provides smoother output, while the memory-based models are more flexible and easier to use and modify in a creative context [WDW98].

9.3 Synthesis evaluation

The quality and similarity of musical sounds are difficult to evaluate since both properties depend on the characteristics of the human ear and the taste of the listener. Unfortunately, the cognitive process of hearing is highly nonlinear and dependent on a huge number of external parameters and context. Two completely different time domain signals may result in exactly the same perceptual experience, while two signals that visually appear very similar may turn out to be very different acoustically. Both phenomena can be easily demonstrated in little experiments. If one mirrors all the samples of a digitized piece of music at the zero-axis, the correlation function between the original and the mirrored

signal is maximally negative. However, due to the fact that the ear has no ability of distinguishing the absolute phase of a signal we will by no means be able to tell the two signals apart. Likewise, two signals that correlate strongly in the time domain may widely differ in some frequency components and noise properties. Another compelling thought regarding the nonlinearity of musical perception is the often-quoted insight that a music lover does not experience twice as much joy listening to two of her favorite songs at the same time [Met96].

In addition to these relatively obvious problems regarding the representation and comparability of sounds, there are more subtle psycho-acoustical phenomena which make it difficult to understand, compare, and classify sound. The ear essentially works like a short-term spectrum analyzer that detects frequency components and events in a sound scene. However, the resolution with which components are resolved is limited in the frequency and time domains. Only sound events that are sufficiently distant in time and frequency are resolved as separate events, while those that are too close smear out into a single event. The minimal frequency difference required to resolve two components is called the critical band. The phenomenon that an event becomes inaudible due to the presence of a stronger neighboring event is called masking. Masking is difficult to model, but has many practical applications in audio-signal processing, the most prominent application being lossy compression of audio signals. Lossy compression transmits only those frequency bands that are actually perceived, while perceptually irrelevant components are omitted [Wat95].

In musical synthesis as in lossy compression, only the perceptually relevant output of an instrument is of interest. We need all the audible effects and sound components to be adequately represented, but do not care about the rest. However, even within the reduced space of audible sound components, the relevance of different aspects is difficult to assess and certainly can't be simply taken to equal the energy of the component. For example, a tiny bit of noise during the bow change can make a big difference, even though a lot of harmonic energy is present in the signal at the same time.

The characterization of musical timbre has been the topic of extensive research [Sch77, Bla73, Wes79, RD82, Hou97]. *Timbre* is the term most often used to summarize sound properties that are not easily described in terms of the common quantitative measures such as loudness, pitch, and duration. Unfortunately, the concept is hardly defined scientifically but has different meanings for different people [Mar99]. In fact, the American Standard Association has explicitly defined the term in the negative.[2] Timbre certainly relates to the spectral characteristics of a sound or an instrument, that is, the overtone series. However, the steady state spectrum of a musical note played on almost any instrument provides very little discrimination for human as well as for machine listeners [Mar99]. The overall signal envelopes and especially the attack characteristics of a note are much more

[2] "[Timbre is] that attribute of auditory sensation in terms of which a listener can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar[...]. Timbre depends primarily upon the spectrum of the stimulus, but it also depends upon the waveform, the sound pressure, the frequency location of the spectrum, and the temporal characteristics of the stimulus [Ass60]." Note that much of this definition is meaningless or redundant. Clearly "timbre" has to depend on the waveform, which is just about all a human or a machine can know about a sound, and it has to depend on the sound pressure, otherwise we wouldn't hear anything.

relevant for the discrimination and characterization of sounds. The problem of classifying sounds in terms of timbre is close to that of asserting the similarity or quality of one sound signal relative to a model signal. In both cases, a measure of similarity needs to be defined that is largely dependent on the psycho-acoustic properties of the ear and brain.

Scientifically sounder than the concept of *timbre* is the concept of *perceptual sound dimensionality.* The attempt to identify perceptual dimensions beyond pitch, loudness, and duration led to multidimensional scaling (MDS) [Gre78, Wes79, HKCH97]. In this methodology, firstly features are defined which are likely to be relevant for the human perception of sounds. Secondly, subjects are exposed to a series of tones and are asked to rate the similarity and dissimilarity of each pair. A computer algorithm then finds a low-dimensional subspace of the feature space that best describes the listener experience. In addition, features are weighted to accommodate the dissimilarity rating. According to Hajda, the results of different MDS experiments are rather inconsistent [HKCH97]. The feature selection appears to be highly dependent on the kind of sound source, the instrument family and the context. The only criteria that is consistently identified as a principal dimension is the average spectral centroid. Wessel [Wes79] showed how this dimension strongly correlates with the perception of brightness.

Wessel's observation has been confirmed and extended by Beauchamp [Bea82, Bea93]. The latter suggests that sounds will be perceived as equal as long as the spectral centroid and the intensity (that is, the indicating loudness) match. Beauchamp uses these features as the fundamental objective and evaluation methods for his work in analysis and synthesis.

Chapter 10

The digital stradivarius

10.1 General concept and chronology

10.1.1 Background

In the early 1990's, physicist Neil Gershenfeld, cellist Yo-Yo Ma, and composer Tod Machover, joined their disparate expertises to design, perform, and compose music for what they called the *Hypercello*. The device they conceived of preserved the traditional functionality of a cello: the sound-generating mechanism was acoustic, it was played like a cello, and it sounded like a cello (fig. 10-1). However, the instrument was enhanced by a multitude of sensor devices that tracked the action of the player in real time [PG97]. The cellist not only played the instrument in the way he was trained, but generated information describing how he actually went about playing. This information was interpreted by software and mapped to computer sounds that were either presampled or generated in real time. Hence, the role of the cellist was closer to the traditional role of a conductor who controls an entire orchestra (section 10.6) [Mac92].

Subsequently Machover and Gershenfeld collaborated on more *Hyperstring* projects, including a *Hyperviolin* project with violinist Ani Kavafian (fig. 10-2) and a *Hyperviola* project with violist Kim Kashkashian. In all these collaborations, a traditional instrument was enhanced with sensor technology and computer-generated music. Coming out of these projects, Gershenfeld was a little frustrated that he had to keep the acoustic (electric) instruments to generate beautiful violin, viola or cello sounds [Ger00a]. Despite very fast machines, no computer algorithm was available to replace the beauty of the original instruments.

Gershenfeld considered violins to be analog computers, i.e. machines that take some input, do computation, and output a signal. He furthermore argued that modern digital computers had essentially become faster than nature in processing information. Gershenfeld concluded that it should be possible to replace the analog processor by a digital machine and hence create a digital violin. This was the birth of the digital Stradivarius project.

Figure 10-1: Yo-Yo Ma playing *Begin Again Again...* by Tod Machover (Hypercello by Machover and Gershenfeld), Tanglewood, August 1991.

10.1.2 The goal

It is our goal to predict and synthesize sound from physical control parameters. Our approach is data-driven: the model is learned from observations of a given acoustical instrument. During training, a network learns the mapping from physical gesture input to audio parameters. During synthesis, the network generates appropriate audio, given new input. Along with the software synthesis engine, we have designed a hardware interface that feels and sounds like the acoustic instrument with the final goal that the original and the model become indistinguishable.

Our approach to musical synthesis lies conceptually in between physical modeling and wavetable synthesis. We do not model the instrument in terms of first-principle governing equations, but extract the essence of these equations. The goal is to build a model that, from the point of view of an independent listener or player, appears to obey the same physical laws as the acoustic instrument. On the other hand, we choose to represent the audio data as sequences of samples and in the form of a spectral decomposition which positions our approach close to conventional synthesis techniques such as global sampling and sinusoidal analysis/synthesis.

In the physical modeling approach, one replicates the physical mechanisms of the acoustical instrument from first principles [Woo92, Smi92, Smi98]. The models are directly derived from fundamental acoustic research and have initially been scientifically motivated. Only later were the numerical implementations turned into synthesis engines. Physical models have the amazing property that they can be used to create new instruments based on physical principles without having to build them mechanically. The flexibility is endless, but there are also serious problems with this approach.

Gershenfeld showed how a simple finite element approximation of a violin is limited

Figure 10-2: Ani Kavafian and the Boston Modern Orchestra Project, conducted by Gil Rose, performing *Forever and Ever* by Tod Machover (Hyperviolin by Machover and Paradiso [PG97]), Jordan Hall, Boston, June 1997.

in terms of speed [SG00]. Assuming ten body modes per body axis and ten finite-element nodes per cycle, we get 10^4 nodes per violin plate and on the order of 10^5 nodes per instrument. If we multiply this by a CD-quality sample rate of 44 kHz, we end up with roughly ten giga-instructions per second needed to run a model in real time. This is still an order of magnitude more operations per second than what today's fastest machines can handle. In addition, there is no systematic way of allocating the parameters of a physical model. Although there have been attempts to find coefficients of waveguide filters from observed data, it is hard to extract differences as subtle as those between two master violins [Smi92].

Physical modeling provides a lot of flexibility, but it is difficult to achieve good sound quality. Global sampling is extreme in the opposite way [Mas98]. Since the original digital samples are preserved, the sound quality is as good as the recording technology, but there is no notion of controlling and shaping the music. This is fine for keyboard instruments where the control is low-dimensional and where there is no continuous control other than the eventual release of the note. In the case of a violin, however, the technique fails because there are more than a few degrees of freedom in the control space and there are an unlimited number of possible playing situations. Our approach is in fact best described as a *Physics Sampler*. We infer our model from recorded data but hope to create a model that has the flexibility of a physical model.

We are also convinced that the artistic and expressive possibilities of a violin are intrinsically related to the very specific input device consisting of the bow, the strings and the fingerboard. We believe that no keyboard can ever recreate the expressive space of an acoustic violin simply because the kind of control available on a keyboard is inadequate to shape the sound of a string instrument. Therefore we keep the interface as it is and

design unobtrusive recording sensor hardware for the violin as well as a sensor interface for synthesis only.

We choose to predict and model sound based on physical input data to the model. This approach is new in the field of musical synthesis. More common is the approach taken by Metois and Wessel et al. who reconstruct a waveform from a given perceptual description in terms of pitch, loudness and brightness [Met96, WDW98]. Although this is not our primary intent, our system contains the solution to this problem as well. Using pitch and loudness as input to the model rather than an output, our system simplifies considerably. The only elements to be predicted are spectral parameters (sinusoidal synthesis) or sampling parameters (cluster-weighted sampling). In fact, this is a considerably simpler task than the highly nonlinear mapping from bowing and finger data to a useful sound representation.

10.1.3 Defining the measurements

Our sound synthesis algorithm is based on measurements of physical inputs of the violin. Defining and carefully executing these measurements was assumed crucial for the success of the model. The search for optimal measurements was guided by three fundamental considerations:

1. Assuming unlimited access to all input parameters, we want to define a hierarchy of important parameters. We need to find out which parameters are most relevant for the physical behavior of the instrument both from a musician's and acoustician's point of view.

2. Different parameters require different physical measurement technologies. Some parameters may not be measurable at all. We need to find the optimal technology and balance the measurement effort with the importance of the measurement.

3. Associated with a measurement is a signal to noise ratio (SNR) which indicates the statistical quality of the measured value relative to the true value. Depending on the SNR, the measurement may be worthwhile or not.

The **left hand** of a violinist shortens the length of the vibrating string, which causes the wavelength of the vibration and the perceived pitch to change. Clearly we need to track the position of the fingers on the different strings and the following sections introduce a way of making this measurement.

In addition violinists adjust the pressure of their left hand fingers on the strings. The quality and timbre of the violin sound changes depending on the pressure between the finger and the string. A different finger pressure can also affect pitch. Unfortunately, the left hand finger pressure is even more difficult to measure than the position in direction of the fingerboard. Since we have no measurement technology available, we neglected the pressure of the left-hand finger.

The violin bow is held by the **right hand** of the player. It interacts with the string in a stick-slip motion, which causes the string to resonate as described in the famous Helmholtz motion (fig. 9-1). A Helmholtz resonance is only established if three major bowing parameters, bow speed, bow pressure, and bow position relative to the bridge, are

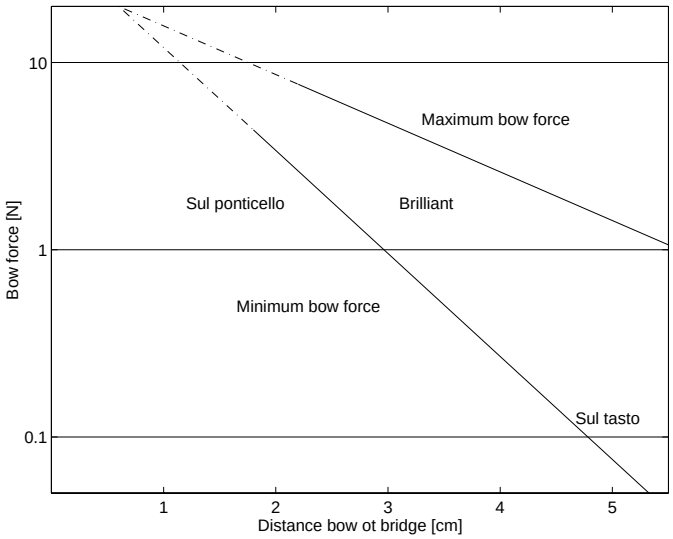


Figure 10-3: Range of bowing force versus bow-to-bridge distance for a cello, given a bow speed of 20 cm/s (after [FR97, p.279] and [Sch73]).

kept within a certain window (fig. 10-3). The intensity of the sound is roughly proportional to the bow speed, to bow pressure, and to the inverse of the bow-bridge distance [Cre84, FR97]. With decreasing distance between bridge and bow, more pressure and less bow speed is required to achieve the same sound intensity. The system of three parameters is over-determined, in that there is more then one combination of parameters that results in the same intensity. The following chapter will introduce the technology that was used to measure these parameters.

Also important from a player's perspective are the angle between the plane described by bow-bar and hair and the strings as well as the angle between the bow-bar and the strings. The former angle adjusts the amount of bow hair touching the string, which affects the overtones of the sound. The latter angle is used to achieve a change of bow position while the bow is in motion. Unfortunately these parameters are difficult to measure. We assume that they are implicitly represented in the measurements mentioned above.

10.1.4 Embedding and musical synthesis

The digital Stradivarius project was originally motivated by results from nonlinear dynamics theory. The embedding theorem (section 2.2.1) claims that *any* nonlinear physical system can be emulated numerically if only we have access to some measurements of the system. For a period of about a year we tried to apply the embedding theorem to reconstruct the state space of the violin and then use this state space to predict the audio signal sample by sample in an iterative approach. For a variety of reasons, the implementation of this promise turned out to be more difficult than expected.

We built specialized hardware to track the violinist's gesture data and the violin audio

signal (section 10.2). Although we are seemingly able to capture the most important and rich input data, our measurements are unlikely to be totally exhaustive. However, the input-output embedding theorem requires that there be a complete account of all the driving signals of the model [Cas92]. Unfortunately, there is little reason to believe that the ability to reconstruct a time series should degrade gracefully with an increasing lack of input information. Rather than getting a slightly worse forecast, we are likely to completely lose the predictability and stability of the system given missing data.

A second serious problem is the difference in timescales between input and output time series. The gesture control of a violin happens at roughly the timescale of a millisecond, which is generally assumed to be the limit of human motor action. This estimate is very optimistic and perceivable human action is rather on the order of 100 ms or even slower. A violin audio signal, however, covers the band between 130 Hz (the open G-string) and 20 kHz (roughly the highest perceivable frequency), which corresponds to a period of 0.05 ms. The nonlinear character of the input-output mapping is obvious from looking at these timescales alone. Unfortunately, relating these very different kinds of signals to each other turns out to be very difficult. Embedded in the same state space, the input hardly changes while the output goes through 100 periods. How could the output signal be conditioned in a meaningful way on the input? Somehow the model has to bridge this discrepancy using a meaningful representation of all the signals.

The use of a reconstructed state space to predict a system output relies on the iterative structure of the embedded space. The reconstruction theorem uses lagged samples of the time series to predict future values. The predicted values then become part of the feature vector for new predictions. Since small prediction errors corrupt future feature vectors, errors propagate and cause the predicted signal to degrade quickly. While the embedding theorem works well as an analysis tool to assess system characteristics, it is rather difficult to apply to the emulation and prediction of these systems.

A last insight regarding the failure of state space reconstruction for synthesis purposes concerns the nature of musical signals. An instrumentalist tries to achieve a certain sound, that is, he/she shapes the perceptual qualities of the music. As the player doesn't perceive the details of the waveform, such as partial phases, he also doesn't control these details, but only shapes the overall statistics of the instrument output. As a consequence the relationship between waveform and control sequence is not deterministic, but intrinsically stochastic. While the spectral content of the sound is predictable given the gesture input, the stochastic components are not. We observe that we need to model the process, not the particular instantiation of the process as recorded in a single sound. While we can predict deterministic aspects of the signal, stochastic behavior needs to be summarized in appropriate statistics such as the power spectrum [SBDH97].[1]

10.1.5 A working hello-world: the arm demo

Given the lesson from the last section we wanted to come up with a representation that would allow us to generate recognizable violin sound. The decision was to build a system that would work but sound bad and then slowly improve from something that was working rather than going for a perfect model right away. Initially we worked with recorded

[1] In the chronology of the project, this work was done from October 1995 to December 1996.

Figure 10-4: The arm demo (summer 1997). A little antenna is hidden under the shirt, which receives signals from the bow.

commercial violin sounds[2] in an analysis/synthesis approach. We used different representations to decompose the sound and put it back together, slowly identifying which elements and parameters were perceptually important and which could be neglected. It was the goal of this procedure to find a representation that could be predicted given the available input data. A sinusoidal decomposition obtained by structuring the data in terms of frames and by applying a STFT to those frames turned out to be a reasonable choice, mostly because this representation can easily abstract from phase information (section 9.1.2).

The discrete short-term Fourier transform is only one possible choice for a spectral analysis of the audio signal, yet it is a rather good one. Different time-frequency transformations such as the Fourier, cosine, or wavelet transform use different kinds of orthogonal basis functions to expand the time domain signal. The author experimented with wavelet transforms but couldn't find any advantage over the Fourier transform in the case of audio analysis [KMG91]. Clearly the sinusoidal basis functions of the Fourier and cosine transforms are very adequate to represent the highly harmonic structure of musical signals.

Using a sinusoidal representation was a breakthrough in that for the first time we were able to synthesize sound in a stable way. We had found a representation that at the time didn't sound great but allowed us to make stepwise improvements on sound quality and functionality of the instrument.

For recording and real-time synthesis we used the *Paradiso* bow, a violin bow enhanced with electronics to measure the bow position in two dimensions and to measure the finger pressure [PG97, Sch96].[3] We also experimented with accelerometers mounted on the

[2] Victoria Mullova's recording of J.S. Bach's violin solo sonatas and partitas.

[3] Joe Paradiso designed the bow for the Hyperviolin project with Ani Kavafian. It was originally put together to last for one concert, but then served the author for a period of three years of daily research.

Figure 10-5: A one string digital violin (fall 1998). The strings are muted. The instrument functions as a sensor input device for the single-string model, ranging over two octaves. A sound clip is available from [WEB].

bow as an alternative to a capacitive measurement of the bow position. Accelerometers measure the acceleration of the bow in three dimensions and hence provide all the necessary information to reconstruct the trajectory of the bow in space.[4] However, there are major drawbacks to this approach. Since we need to integrate the raw sensor values in order to retrieve bow speed and position, the baseline of those parameters is likely to drift over time. Baseline correction is possible, but comes with trade-offs regarding filter time constants.

The real-time implementation of this model was presented in form of the arm demo (fig. 10-4 and [WEB]), which admittedly had a number of serious limitations. Obviously the player couldn't play notes with different pitch, which motivated only a few composers to write music for the instrument...[5] Furthermore, the model had the annoying feature that the sound didn't stop when the player stopped bowing. The reason for this missing feature was that the training data set didn't include silence, which was an important lesson for future incarnations of the model. At the time we added on/off functionality by means of a pressure threshold. There was also considerable latency ($\approx$ 200 ms) between the player action and the audio response. This latency was mostly due to a slow serial connection between the sensor hardware and microprocessor on one end, and the PC running Windows on the other end.[6]

[4] In order to get the position and dynamics of the bow relative to the violin, one needs to do the same measurement for the violin.

[5] Special thanks to David Borden, who was actually prepared to write a piece.

[6] January 1997 to January 1998.

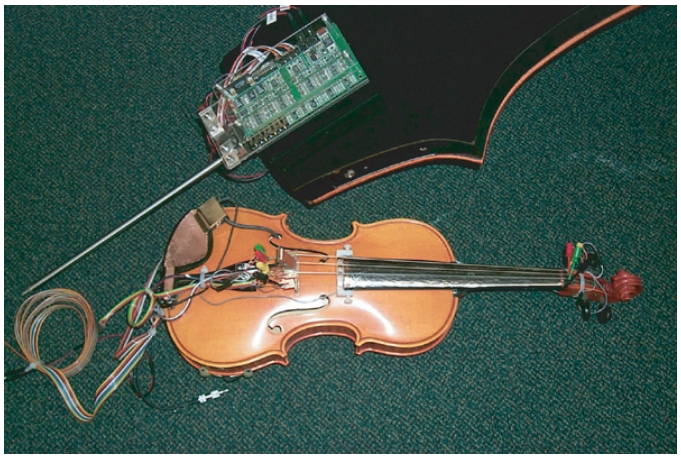


Figure 10-6: Violin with mounted sensors and RAAD cello with the sensor board mounted in the back of the instrument.

10.1.6 A cellist playing a single-string violin

For the next incarnation of our model, we again used the sensor bow from the previous year, but enriched the interface with a sensor that measures the finger position on the A-string of the violin. A number of different sensing approaches were tested including an ultrasonic approach as used in computer touch screens: an ultrasonic impulse is sent along the string, reflected by the finger and received at the end of the string from where it has been sent. Unfortunately the medium turned out to be a bad guide for the ultrasonic wave and the idea had to be abandoned. Likewise, the idea of using a capacitively-coupled voltage divider along the string was abandoned because the capacitive coupling doesn't provide resolution necessary for a precise finger location but rather smears out along the fingerboard.

The idea of using a contact-based voltage-divider along the fingerboard using the string as a pick-up electrode seems rather obvious. However, attempts to implement this for the early hyper-instruments failed because it was impossible to find a electrically resistive material that at the same time was mechanically strong enough to stand the impact of the string on the fingerboard.[7] We decided to compromise on the electrical side and use stainless steel which is available as very thin foil (0.001 inches) but is strong and easily mounted below the strings. The electrical resistance on the stainless steel strip was on the order of 1 Ohm over the length and width of the fingerboard of violin and cello.[8] Due to the rather low resistance of the material, a high current[9] was required to run through the device.

[7] For the same reason, force resistive strips were not an option, at least for the recording instrument [TC99].

[8] The values are very much the same for both instruments since $R \propto \frac{l}{A}$, where l is the length of the piece of material and A is the surface area.

[9] alternating current, 5kHz.

The stainless steel strip was taped onto the fingerboard and the supporting signal-conditioning circuit was proto-boarded on the instrument next to it (fig. 10-5). Since the many cables running to the instrument were only as long as four feet, the action of the musician was spatially limited to a circle around the desktop PC. However, the model provided the functionality of playing a scale over about one and a half octaves. It was trained on violin data, but the Windows GUI included a slider which allowed the user to change the frequency and turn the installation into a cello or even a double bass. Unlike later models which were truly based on recorded data from different instruments of the violin family, this early implementation of a cello was a true hack.[10]

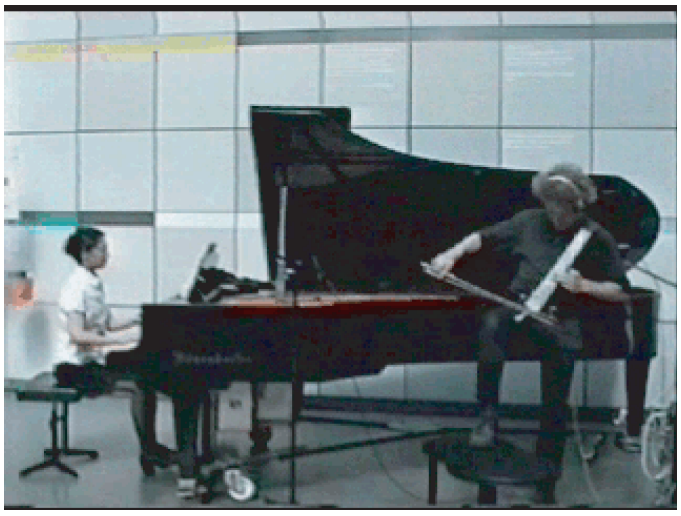


Figure 10-7: The Marching Cello in performance (Summer 2000); Mary Farbood (piano), the author (Marching Cello).

Although this incarnation of the model allowed for notes of different pitch to be played, the limitations of the device were still serious: clearly a single string is rather limiting in terms of what one can achieve with an instrument; the installation was far from being movable and therefore more like an advanced hello-world demo than a usable instrument; the setup was mechanically and electrically rather fragile causing many unexpected problems with bad connections; the sound clearly originated from a violin but the sound quality was still below a beginner instrument.[11] A more specific problem: since the author is trained as a cellist, he plays the violin interface in cello position, which arguably looks weird (fig. 10-5).

The representation used in the single-string violin is purely sinusoidal. Clearly the methodology had to be adjusted to handle pitch changes. A little later we developed

[10] During a demonstration for IBM, the question was posed concerning how long it would take to turn the violin into a cello. The author carelessly replied that would be a five-minute hack... and ended up having to prove his claim. Five minutes later, at the end of the lab tour, the IBM visitors indeed got to hear a cello. A video clip of the demo is available from [WEB].

[11] ...to quote Neil [Ger00a]: "This is recognizably the worst violin in the world."

a preliminary version of cluster-weighted sampling (section 8.2) which uses a wavetable representation of the audio. However, there was no real-time implementation of this representation. We also experimented with an LPC based synthesis model, which conceptually is very interesting (section 6). Unfortunately it doesn't provide as much control from a model-building perspective. In particular the linear filters tend to be unstable. We have not implemented a real-time violin system using LPC (section 6.3).[12]

Figure 10-8: *Take The Money And Run*, Woody Allen, 1969.

10.1.7 The final approach

After this series of implementations we defined the following *final* model. Modeling and synthesis processes are composed by a series of separate steps (figures 10-9 and 10-10). The sequence of modeling steps consists of the following:

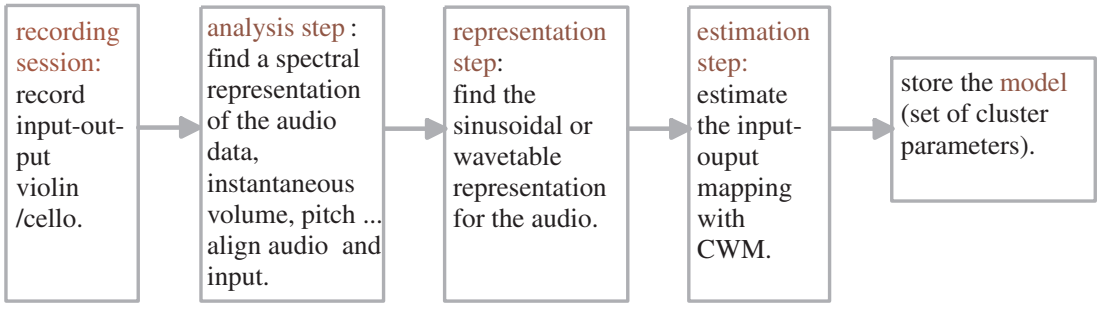


Figure 10-9: Modeling Sequence

[12] See [WEB] for off-line sound examples. February 1998 to January 1999.

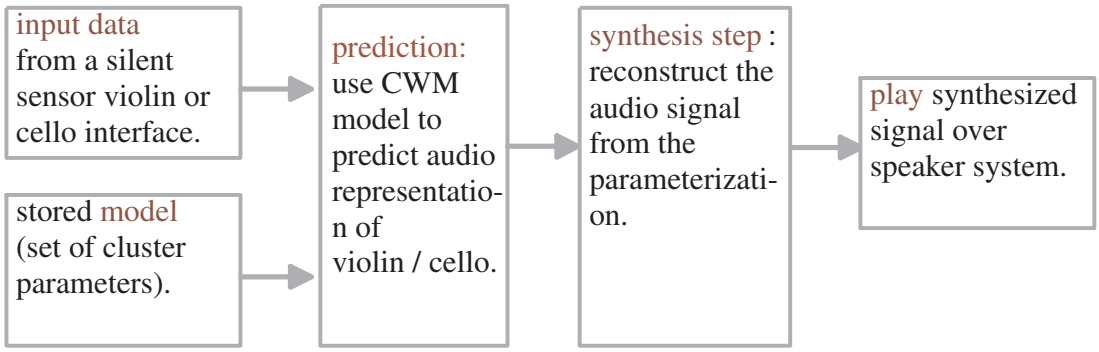


Figure 10-10: Synthesis Sequence

1. We start out with a recording session of violin or cello sounds. The audio signal of a played instrument is recorded along with the physical gesture input to the instrument.

2. The database of sounds is analyzed with respect to perceptual properties such as pitch, loudness, brightness. It is also decomposed into a spectrum of harmonic components.

3. The recorded input data is ordered along with the analyzed audio data in frames that are updated 86 times per second.

4. CWM or CWS is trained based on input-output frame data.

The sequence of real-time synthesis steps (figure 10-10):

1. A player plays the violin/cello sensor interface to deliver gesture control input in real time.

2. The real-time input is calibrated and fed into the CWM model 86 times per second. The model predicts the representation of the audio data it was trained for.

3. From the predicted representation, the audio signal is reconstructed and sent to the sound device.

10.2 Data-collection

Input-output violin data is recorded on 13 channels at 22050 Hz on a PC data acquisition board. The channel assignments are (appendix B),

0-2: three bow position signals.

3: bow pressure.

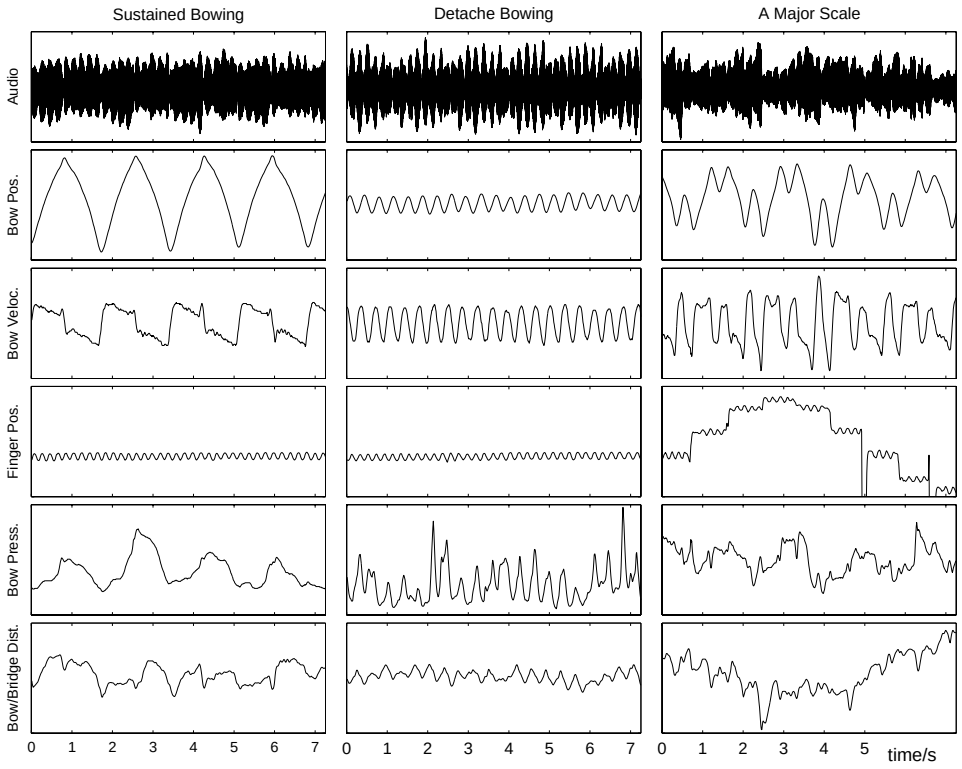


Figure 10-11: Violin input data. *Horizontally:* sustained bowing, détaché bowing, A major scale. *Vertically (from top to bottom):* audio data, bow position, bow velocity, finger position, bow pressure, bow-bridge distance.

4-7: finger position on string I-IV.

8: audio signal recorded from a microphone placed inside the instrument (violin) or a commercial pickup (RAAD electric cello).

9-12: audio signals from string I-IV recorded by a dynamic pickup system.

In this approach channels 0-7 are over-sampled by two orders of magnitude. Since the sampling device is fast enough, this approach is much more convenient than dealing with different sampling rates. Signals 0-7 are down-sampled by the factor 256 ($\approx$ 86 frames per second) after recording. About 1.5 hours of cello data were recorded in 20-second-long sequences. About half an hour of violin data was recorded in the same fashion.

In earlier work a data acquisition card for capture of the slow input data was used together with a PC sound card for a stereo recording of the audio signals. Obviously this reduces the number of audio channels considerably. The biggest problem with this approach is the synchronization between the two recording boards. The temporal delays due to device drivers and operating system are not consistent or systematic in any way.

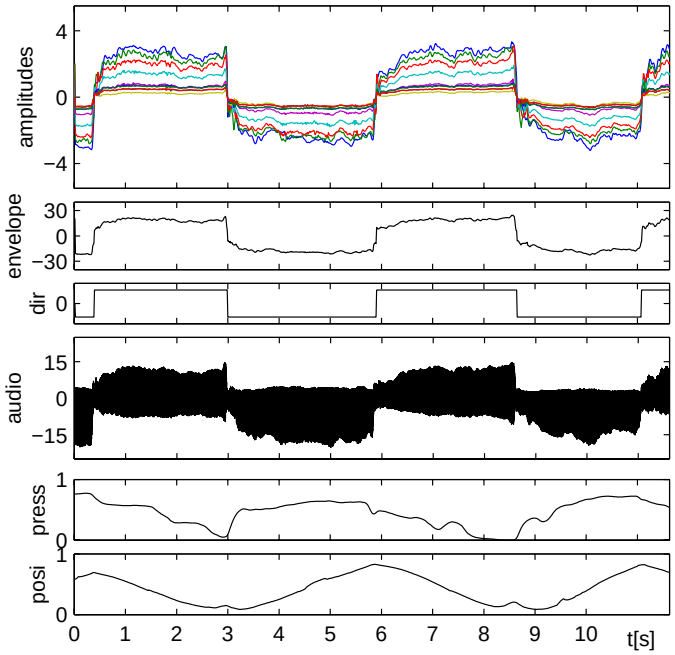


Figure 10-12: Analyzed cello data - sustained bowing. *From bottom to top*: bow position relative to the strings, bow pressure, audio signal, extracted bowing direction, envelope, harmonic amplitudes.

For posterior calibration we used a clapperboard approach. An additional input channel was switched from one to zero about half a second after the recording started. At the same time a microprocessor generating an audible sound signal (9kHz) superimposed on one of the audio channels was turned off. After recording the switching was detected in both data sets and the time series were aligned accordingly.[13]

10.3 Data-analysis

Regardless of the sound representation (sinusoidal, wavetable, or mixed presentation) we extract relevant physical and perceptual information from the collected audio data.[14] The time-synchronized physical input data is used to support this analysis where needed.

1. A **spectral representation** of harmonic components is extracted at a framerate of $22050/256Hz \approx 86Hz$. A Hanning-windowed short-term Fourier transform is used. The finger position input indicates the pitch/register of the sound. Depending on the pitch estimate we pick the size of the sample window ($512 - 4096$ samples). The

[13] See appendix B for technical details.
[14] Time domain audio data is always sampled at 22050kHz in this work.

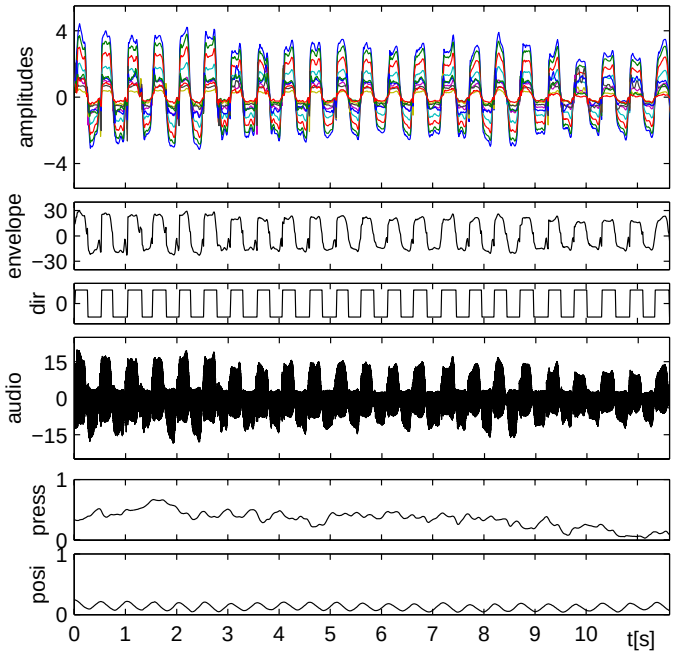


Figure 10-13: Analyzed cello data - détaché bowing. *From bottom to top*: bow position relative to the strings, bow pressure, audio signal, extracted bowing direction, envelope, harmonic amplitudes.

Fourier frame is taken to be roughly two times this size in order for the spectral curve to be smooth.[15]

From the coefficients we pick peaks of the spectrum and assign an estimated amplitude and frequency to each of them using the methodology from section 9.1.2. Given the finger position we identify **25 partials** corresponding to the first harmonic components of the sound. Hence we obtain a vector with 50 components for each frame of sound.[16]

[15] The size of the Fourier frame is adjusted to match a power of two by zero padding the ends.

[16] The precise algorithm is as follows:

(a) Guess a good frequency bin corresponding to the fundamental of the signal, given either prior knowledge of the recorded signal, or given the finger position of the player.

(b) Within a window of about 5 bins around the original pitch guess, find the maximum of the FFT and use the corresponding bin as second guess for the fundamental bin. Given this bin compute the true energy and the true frequency using the methodology described in section 9.1.2 (equations 9.12 – 9.16).

(c) Given the fundamental frequency, guess the harmonic frequencies using $f_n \approx n \cdot f_1$ $(n = 2...25)$. Then repeat (2) for every frequency f_n.

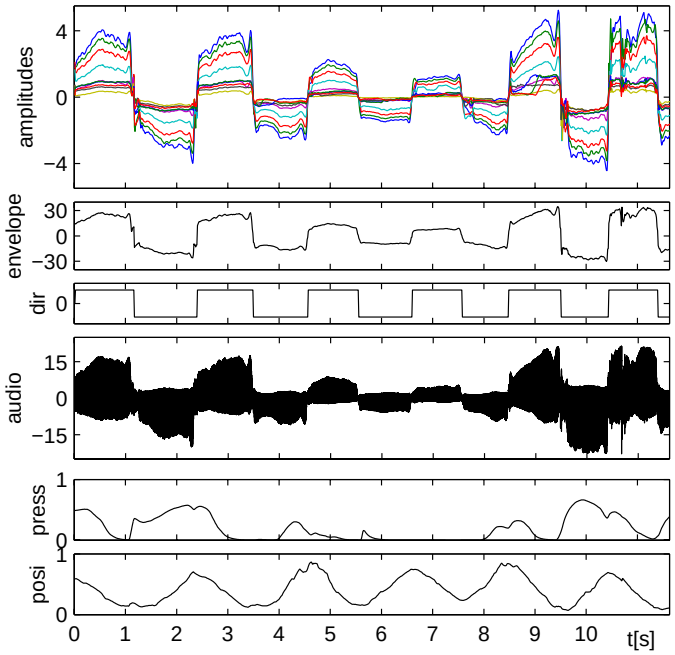


Figure 10-14: Analyzed cello data - dynamic bowing. *From bottom to top*: bow position relative to the strings, bow pressure, audio signal, extracted bowing direction, envelope, harmonic amplitudes.

2. We identify the partial corresponding to the fundamental and use its frequency as our **pitch** estimator P.[17]

3. From the instantaneous spectra and pitch we derive the **spectral centroid** (equ. 8.8) and **brightness** (equ. 8.7).

4. Using a moving average window weighted by a Hamming window (framerate $= 86Hz$, framesize $= 256$ samples) we identify the bias of the audio signal B.[18] Using the same windowing and averaging approach on $|s(t)|$ we find a measure proportional to the root of the signal energy E. We estimate the signal **amplitude** as $A = E - B$.

5. Remembering the sign of the bias B (the direction of the Helmholtz motion, fig. 9-1) and the sign of the bow velocity we detect the **bowing direction** D as well as the location of the **bow changes**.[19]

[17] The author is aware of the fact that the frequency of the fundamental does not necessarily match the pitch of the signal. However, the approximation is reasonably close for the strongly harmonic violin-family instruments.

[18] As can be seen from figures 10-12 – 10-13 the recorded audio signal is biased depending on the bowing direction. The difference is inaudible for all practical purposes, but needs to be considered in the modeling process.

[19] The location of the bow changes is evaluated using the bow position signal, while the bias of the

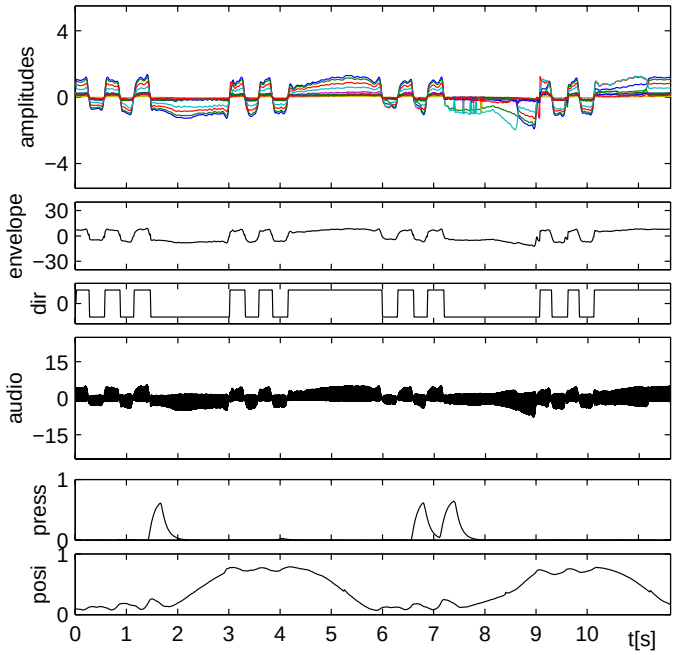


Figure 10-15: Analyzed cello data - mixed bowing pattern. *From bottom to top*: bow position relative to the strings, bow pressure, audio signal, extracted bowing direction, envelope, harmonic amplitudes.

6. Using D we assign negative (up bow) and positive (down bow) signs to harmonic amplitudes and to the instantaneous signal amplitude (fig. 10-12 – 10-14) leading to an **alternating spectral representation**. This little trick forces the signal to pass through zero when a bow change occurs during synthesis. [20]

The spectral components function as indicators of the timbre properties of the recorded sounds (section 9.3).

10.4 Inference model

10.4.1 Representation

Although linear transforms, such as Fourier or wavelet transforms do not change the information content of the data, it makes a considerable difference in which domain we

Helmholtz motion is a more reliable (short-term) estimator for the bowing direction.

[20] The early incarnations of the model (section 10.1) have been built without this little trick. The bow change is sometimes not clearly noticeable.

In learning how to play violin or cello, one spends many years trying to make the bow change as inaudible as possible. The author, on the other hand, spent many years trying to make bow changes as audible and characteristic as possible...

try to predict. Notably, we need a representation that generalizes with respect to the boundary conditions since it is our goal to define a singular valued function that maps any reasonable input to a unique output. The output can vary numerically for identical input, but needs to be reproducible in terms of its perceptual properties. Clearly, a time domain audio signal does not have this property, since two identical gestures may result in completely different waveforms. We believe that the deterministic content of the signal is hidden in the power spectrum, because the player actively shapes the spectral content of the signal.

Figure 10-16: The Marching Cello.

We abstract from the notion of absolute phase. Since phase is not perceived in a typical playing situation, we may pick it arbitrarily as long as we avoid discontinuities in the signal components. We furthermore observe that violin and cello, like most musical instruments, are characterized by slowly varying boundary conditions that map onto a fast audio signal. The nonlinear interaction between bow and string causes the slow gesture input to be turned into the famous Helmholtz motion containing the frequency components of the final audio signal [Cre84]. The slow and fast elements describe two different times scales which, if mixed, confuse each other. Instead, fast and slow dynamics

and the corresponding state variables need to be treated differently. CWM provides the means to implement such distinction: the slowly varying boundary conditions are used to select the domain of operation (cluster) in the configuration space (equ. 3.6), while the fast dynamics are handled by the local models and the associated state variables (equ. 3.5 and 3.8).

Sinusoidal representation

We decompose the signal (sampled at 22050 Hz) into a short-term Fourier transform (STFT). The window length is chosen depending on the register (section 10.3) and the frame rate is $22050/256 \approx 86$ frames/second. From the Fourier coefficients, we pick the 25 bins that contain the first 25 harmonic partials. Each partial is represented by its frequency and its energy. Hence for each frame, we obtain a 50-dimensional vector representing the instantaneous properties of the audio signal.

The frame rate matches the input sample rate, which yields a classic function approximation scheme of $\mathcal{R}^x \rightarrow \mathcal{R}^y$. Frames are considered independent. Continuity is achieved through time-lagged input values.[21] We summarize frequency and energy components of the spectral representation in a single output vector $\mathbf{y}$. Alternatively, we split the model into two separate models with different feature vectors $\mathbf{y}_f$ and $\mathbf{y}_e$, since frequencies and energies rely on different kinds of input information (section 10.4.2).[22]

During synthesis, output frame $\mathbf{y}$ is predicted given the incoming feature vector $\mathbf{x}$. The estimated instantaneous spectral information is converted into a time-domain audio signal using a superposition of sinusoidal components. Since the original analysis is strictly harmonic, we can avoid complicated algorithms to match harmonic components [MQ86]. Instead, all the partials are kept strictly parallel. The ith harmonic of frame k always transitions into the ith harmonic of frame $k+1$. In between frames, frequencies and energies are interpolated linearly.

$$
\begin{aligned}
s[k,n] &= \sum_i e_i[n] \sin(\phi_i[n]) \qquad (10.1)\\
\phi_i[k,n] &= \phi_i[k] + \sum_{n'=1}^{n} \Delta\phi[n'] \\
&= \phi_i[k,n-1] + \frac{2\pi}{f_s}\left[f_i[k] + \frac{n}{L}(f_i[k+1] - f_i[k])\right] \\
e_i[k,n] &= e_i[k] + \frac{n}{L}(e_i[k+1] - e_i[k])
\end{aligned}
$$

where L is the number of samples per frame ($L = 256$), k is the last frame, n is the sample count starting with the last frame, and e_i, ϕ_i and f_i are the energy, the instantaneous phase and the frequency of the ith partial [SS90, SCDG99].

Only a few ad hoc measures are necessary to avoid model artifacts. Since the model

[21] The experimental results in this section use this method rather than the hidden Markov approach. The latter approach was avoided in order to keep the complexity of the model at a minimum.

[22] The earliest incarnation of the model (section 10-4) uses a joint output vector and CWM model for frequencies and amplitudes, while subsequent models use independent predictors for the two subsets.

is designed to generate continuous sinusoidal components, sound is likely to be generated even though the bow does not touch the string or string-like support. In fact one of the particularities of the hardware used is that bowing input can be generated without physical contact. The problem can be fixed by implementing a switching mechanism. For example, we define a minimum bow pressure that needs to be achieved in order for the volume to be nonzero. Alternatively we require the fingerboard to be touched for the volume to be nonzero.[23]

Wavetable representation

We decompose the training audio signal ($f_s = 22050Hz$) into frames at framerate = 86.1Hz. Each frame describes physical and instantaneous properties of the audio signal. We extract (section 10.3)

- the instantaneous decomposition into harmonic components as an indicator of the instantaneous timbral properties (sections 10.3 and 10.4.1).
- the instantaneous volume of the signal, using the method of the short-term relative signal energy (chapter 8 and section 10.3).
- the instantaneous pitch of the signal. Given the harmonic decomposition, pitch equals the frequency of the fundamental.

Since sample selection, scaling, and pitch prediction require different input features, we choose to build three independent models. The first model is trained to choose the most appropriate sequence of samples given feature vector $\mathbf{x}_s$ (equ. 8.10). The second model learns to predict volume given feature vector $\mathbf{x}_v$ (equ. 8.12) and the third model learns to predict pitch given feature vector $\mathbf{x}_p$ (equ. 8.11).

At synthesis time, we find the most likely sequence of samples given the current vector $\mathbf{x}_s$, resample the sequence with respect to the predicted pitch, and rescale it with respect to the predicted volume (equ. 8.13). Sequences of samples are patched according to the methodology in Chapter 8. Pitch and volume are linearly interpolated in between frames (equ. 10.1).

Mixed representation

The strength of the sinusoidal representation lies in its flexibility and robustness with respect to new player input. Since phase is constrained to be continuous by construction, artifacts in the generated audio are minimal. The disadvantage of this approach is the lack of noise components in the sound. By definition the harmonic decomposition disregards noise in the training audio signal as well as in the predicted signal. This is particularly problematic for the attack part of the violin sound, which is when the noise components are strong and characteristic. In general the sinusoidal representation and prediction handles sustained sound very well, but fails in dealing with transitory sounds.

[23] The cleaner solution adds *silent violin* data to the training set. In the final real-time model, this was enough to assure that the instrument stopped when not bowed.

The wavetable approach, on the other hand, reproduces the full timbre of the sound, including the noise components. The downside of this faithfulness to the original is less flexibility with respect to unknown input. Also, CWS is more vulnerable in terms of undesired artifacts.

We combine the advantages of both methods by having a stored sample be played immediately after the bow change, while having the sinusoidal method model the sustained part of a note. The synthesis program detects a bow change from a sign change of the bow velocity. When the bow change occurs, we start a recorded note onset sequence, pitch adjusted and scaled with respect to the desired volume. After about half a second,[24] the sample sequence fades out and the sinusoidal model takes over at the same rate using

$$x[N+n] = w[N_w - N_f + n] \cdot (1 - A[n]) + s[n] \cdot A[n] \tag{10.2}$$

where x is the final signal, N is the starting point of the cross-fade, w is the wavetable sequence, N_w is the length of the wavetable sequence, N_f is the length of the fading interval, and x_s is the sinusoidal sequence. The fading coefficients $A[n]$ describe a linear ramp.

This approach preserves the noisy and characteristic violin attack without compromising the sustained part of the note. The resulting attack is over-realistically aggressive [WEB].

10.4.2 Feature selection

The available raw input data covers

- the bow position relative to the strings b_x;
- the bow position relative to the bridge b_y;
- the pressure between the player's forefinger and the bow p;
- the finger position on each of the four strings $f_{1...4}$.

Given this raw input, we infer

- the bow velocity v from b_x, using a moving average filter and a differentiator;[25]
- the inverse of the bow-bridge distance b'_y;
- the bow change c from a change of sign of v;
- pitch $p_{1...4}$ from $f_{1...4}$. Since the fingerboard measurement is perfectly linear, pitch can be calibrated with two calibration points only (appendix B).

[24] In our experiments this time was fixed at 0.5s, but it could be taken to be a function of the inputs.

[25] Estimating velocity from position is difficult, especially if the position signal is noisy. In general we are facing a trade-off between latency and noise-level in the estimator. We use a time window of three frames to estimate the differential position signal.

Early on we also experimented with a Kalman filter, but found that, for our application, the response time of the filter is too severe.

In order to keep the non-linear mapping as smooth and simple as possible, we aim to construct feature vectors that are as close to the needs of the physical system as possible. Cremer [Cre84] finds that the maximum displacement of the string z and the transverse force at the bridge F are proportional to $\frac{v}{b_y}$. The relationship can be written as

$$\begin{aligned} z &= \frac{1}{8f}\frac{v}{x_y} \\ F &= \mu c \frac{v}{x_y} \end{aligned} \tag{10.3}$$

where f is the frequency, μ is the mass per unit length, and c is the wave speed [FR97]. We compute z from the input data and add it to the vector of features. In general we accommodate physical features at the right polynomial order.

Since past input conditions the current state of the instrument, the input vector is augmented with past input data. By adding time-lagged input samples, we balance the need to include the past input and the burden of a big input space. While the model scales linearly in the output dimension, it is sensitive to a large input space, which favors over-fitting.

Given these considerations, a typical feature vector[26] to predict spectral energies or the total volume is

$$\mathbf{x}_v[k] = \begin{bmatrix} v[k] \\ v[k-3] \\ v[k-6] \\ b_x[k] \\ p[k] \\ p_i[k] \end{bmatrix} \quad . \tag{10.5}$$

A typical feature vector to predict pitch or spectral frequencies is

$$\mathbf{x}_f[k] = \begin{bmatrix} f[k] \\ f[k-3] \end{bmatrix} \quad . \tag{10.6}$$

A typical feature vector to select samples is

$$\mathbf{x}_s[k] = \begin{bmatrix} v[k] \\ v[k-3] \\ b_x[k] \\ f[k] \end{bmatrix} \quad . \tag{10.7}$$

[26] The dimensionality of the following vectors is at the upper end of what is recommendable. Some experiments where done with a subset of the given vectors, e.g.

$$\mathbf{x}_v[k] = \begin{bmatrix} v[k] \\ v[k-3] \\ p_i[k] \end{bmatrix} \quad . \tag{10.4}$$

10.5 Experimental Results

Using an off-line approach, we synthesize violin sounds from sampled input data. Audio examples based on the sinusoidal as well as wavetable representation are available from [WEB].

Real-time synthesis from real-time player data is just about the truest out-of-sample test imaginable for our problem. Off-line out-of-sample testing, even when done carefully, is constantly in danger of falling into statistical traps. Repeatedly using the same training and test data almost inevitably weakens the notion of out-of-sample testing (data snooping), since accumulated knowledge about the data helps with the search for model hyper-parameters [STH99, Whi00]. Also, using test and training data from the same data collection session avoids potential problems with sensor calibration or with player-dependent models. A real-time instrument has to face these issues, since the collection of input data is, by definition, happening at a different time than the collection of training data. In addition, the interface and model may be played by any available musician trained on the particular instrument.

We use a mixed synthesis approach (section 10.4.1), which plays parameterized samples at the note onset and additive synthesis for the note sustain. Video clips of real-time synthesis with input data generated by the Marching Cello are available from [WEB].

10.5.1 Discussion

Mechanical and acoustical feedback

On an acoustic instrument, the player gets both mechanical and acoustical feedback. Touching the strings with the fingers and the bow gives the player a feeling of contact resistance and mechanical impedance. This feedback is important, since the establishment of the Helmholtz motion is subject to multi-dimensional physical constraints. Only when bow speed, pressure, and distance from the bridge are within a certain window, the string is forced into a steady state motion [Cre84, FR97]. The accomplished player feels the establishment of the Helmholtz motion without even hearing the sound. Apart from this fundamental feedback, there are a lot of accidental little mechanical and visual hints that help the player find the right left-hand finger position or bow position.

The acoustic feedback is even more important. It is widely accepted that playing in tune is not a matter of remembering the correct spots on the fingerboard, but of constantly adjusting the finger position to achieve the correct pitch given audio feedback. This is particularly important for position changes of the left hand. The player hears the pitch changing and accelerates and decelerates his arm to come to a halt just at the right position. This is the reason why a good violinist, when given a viola to play for the first time, almost instantly plays in tune, while a mediocre violist[27] will not, even though he has played the instrument for a long time. The violinist hasn't learned the geometry of his instrument but the ability to react in a control loop with the audio.

Our system provides very little mechanical feedback, however, the acoustic feedback loop is closed again. The musician hears the sound coming out of the speakers and

[27] I apologize to all violists. Of course, the example works the other way around as well.

can adjust his technique. In the long run, the player learns how to play the digital instrument and adjusts for possible differences between the acoustic instrument and the digital model.[28]

Temporal delays

The real-time model deals with a number of sources for time delays. We explicitly use the notion of time delay to estimate the bow velocity. Since velocity is the time derivative of position, it can't be perfectly causal. By the time we know that the velocity has changed, the change is already history. Our particular estimator, a moving average/difference filter (section 10.4.2 and appendix B), trades off the time delay with the noise level of the estimator. The longer the filter window, the cleaner is the velocity estimator. However, the delay increases linearly with the window size. Using large windows also smears out useful features of the signal. The final delay is estimated to be about 30 ms.

A second delay is introduced with time-lagged inputs in the feature vector. We try to assure continuity of the model by including aspects of the past in the current state vector. Again, we need to trade continuity with latencies of the instrument response.

A third delay is due to the implementation of model computation and communication between the subsystems. On the input side, we were able to reduce latencies considerably using an ethernet connection between the micro-controller on the sensor board and the PC. Unlike serial port connections, the UDP packet-based communication allows for a constant and instantaneous delivery of sensor data. The delay due to computation almost by definition equals one frame of input data ($\approx$ 10 ms).

The delay between availability of new audio samples and their digital-to-analog conversion on the sound card is more difficult to assess. High-end sound cards along with optimized software (Microsoft DirectSound 7.0) have been reported to achieve delays lower than 20 ms.

Sound quality

A major concern is the evaluation of prediction accuracy and sound quality, both for cross-validation during training and for the evaluation of the final violin model. The training algorithm used (Expectation-Maximization [DLR77][GSM99]) has an implicit error metric, known as the Kullbach-Leibler distance, or relative entropy between the actual data distribution and the approximation. The algorithm converges to an optimal solution given the initial conditions and the model parameters. However, the complex violin model, including its preprocessing and post-processing steps, has a large number of implicit and explicit model parameters, yielding a quasi-infinite number of parameter combinations each of which results in different synthesis results. We believe that no machine-listening technology can replace the judgment of a human listener. We represent sound in terms of its major spectral components and assume that the closeness of two sounds relates to the distance in terms of these perceptual parameters. Yet the final audio model output is compared to the original sound material through the ear of a human listener.

[28] Before demonstrations, the author would typically spend half the night building a new model and the other half of the night practicing the new model.

We are able to synthesize sounds that are arbitrary close to the original recordings as long as a limited data-space is considered.[29] Pitch is much easier to estimate than the sound envelope, mostly because the mapping from bowing patterns to energy is very irregular and noisy. Problematic are models that aim at covering the full violin/cello input space. The explosion of training data is scary. The cello register ranges from C2 to A5, which corresponds to roughly four octaves of notes ($4 \cdot 12 = 48$ notes). At the same time an enormous variety of bowing styles is available to the cellist.[30] Ideally the model should be trained on the cross product of these two spaces. Since we cannot hope to record and deal with that much data, the author chose to first record all pitches but with limited variability in terms of bowing (40 second of data per note). In a second recording session, an extensive number of bowing styles were recorded at a fixed pitch. Assuming that bowing and fingering are independent to some extent, these two data sets were combined to build a model that would know how to predict the different bow strokes for all pitches. Original and synthesized sound samples are available from [WEB].

In terms of real-time synthesis, we achieve a recognizable cello attack as well as a dynamic response with respect to the bow-bridge distance and with respect to bow speed. The real-time cello covers the full register of the acoustic instrument. The functionality of double stops has not been implemented at this point, although this extension would only require a minor software change. Since the sensor interface supports four strings separately, a multiple-strings-at-a-time system would simply run four separate cello models in parallel. *Open string* notes need to be fingered with the current interface, since the sensor technology is indifferent to which string is bowed. The bow model is generic, in that it can be used with any string and any number of strings at the same time. The fingerboard has been repartitioned to provide a spot for the open string as well. In fact we are free to map the fingerboard to pitch in any way we want (section 10.6.3 and appendix B). The model could be improved regarding responsiveness. Fast playing is difficult compared to the acoustic instrument, since the model takes time to build up sound. Work on reducing the latencies should be helpful in this matter.[31]

10.6 Artistic extensions and applications

The Hypercello project (section 10.1.1) was a first attempt to overcome the limits of instruments that haven't changed over the past hundreds of years. As engineers, we are a little bit embarrassed by how little the advancement of science and technology has been able to contribute to devices such as violins. The truth is that we are not only unable

[29] For example, 10 to 40 seconds of sound can be estimated in such a way that the test set, taken from similar input patterns, sounds nearly indistinguishable from the original. Likewise, 5 min of data with the same pitch are tractable.

[30] Gershenfeld compiled the following list: legato, détaché, martelé, sautillé, spiccato, jeté, staccato, staccato volante, viotti stroke, arpeggio, tremolo, sul ponticello, sul tasto, col legno, ondulé [Ger96].

[31] The Marching Cello video clip of *Stars and Stripes Forever* [WEB] runs a model with the following parameters:

Amplitudes: 30 min of training data, three input parameters (velocity, lagged velocity, bow-bridge distance), 15 clusters, local linear approximation.

Frequencies: 8 min of training data, one input parameter (finger position), 2 clusters, local second-order approximation.

to surpass Stradivarius, who worked three hundred years ago, but even worse, we don't know why his instruments outperform the ones built today. In imitating the performance of instruments from observation, our modeling approach has the conceptual potential to get arbitrarily close to a master instrument. A violin by Stradivarius should be as easy or difficult to model as a student violin. Assuming success, the limitations of our model are the limitations of the instrument from which the data originates.

Reconstructing and synthesizing the output of an actual instrument means accessing its internal states. This information can be used in many ways, for example, for mappings of completely genuine computer music. However, full control over the sound generation also allows us to map different known interfaces onto different known instrument outputs. The violin interface becomes a universal controller rather than an instrument that computes violin sound only, just like a keyboard can be an interface to instruments ranging from harpsichords to analog synthesizers.

The hyper-string idea is concerned with instruments for accomplished players who want to extend the expressive means of their instrument. Along a different axis, it may be just as valid to reduce the freedom of control of an instrument [Ger99b] in exchange for easier use (section 10.6.4).

10.6.1 A universal violin controller interface

While the piano keyboard has long been accepted as a universal interface to many different types of instruments, the violin bow has never made it beyond the violin family instruments. The piano keyboard has been used as an interface for all kinds of sound-generating mechanisms, be it hammers that hit or pluck strings, or air tubes that connect to organ pipes. More recently this collection has been, or enlarged by all kinds of electronic and digital instruments. Typically, the key selects pitch while the activation of a key controls note onset, volume, and release. The notion of continuous control conflicts with the keyboard, since the key itself only allows for very brief interaction with the mechanics of the acoustical sound generation.

The violin bow is a perfect example of a continuous sound generator and controller. It provides the means for unlimited musical expression. Any note may develop into the most amazing, complex, and unexpected sound. At the same time, a skilled player is able to create the illusion of a perfectly sustained sound that doesn't stop when the bowing direction changes. Despite these unique features and possibilities, until recently, the haired bow has not been used in a context unrelated to the violin-family sound. In part this restriction is due to the immense skill that is required to overcome the demands of a high-dimensional device.[32] The question is whether there is a fundamental trade-off between control degrees of freedom and the expressive capacity of the instrument. In other words, we would like to know if we could enjoy the richness of a violin-like instrument without investing 20 years of our lives practicing to reach that point.

The violin bow has been designed for the very specific reason of exciting a resonating string in a continuous fashion. Unbeknownst to its creators in the 17th century, it was meant to generate the famous Helmholtz motion of the string which is responsible for the

[32] Gershenfeld [Ger99b] compares the violin bow to a computer mouse. Not surprisingly he identifies six times the number of things one can do with a violin bow than with a mouse.

characteristic sound of bowed string instrument. Only recently have there been technological efforts to extend this successful symbioses of bow and string. For example, Trueman and Cook designed an instrument called BoSSAA (Bowed-Sensor-Speaker-Array) which integrates a bowing and finger sensing mechanism and an array of speakers distributed on a sphere. The purpose of this instrument is to recreate the special characteristics of the acoustic instrument in addition to sound properties [TC99].

In this section we want to consider the violin bow as a universal control interface for musical instruments. Rather than conceiving of a completely new instrument, we map a known interface (the violin interface) onto an instrument such as the cello or a trombone, which also has continuous pitch. The mapping is done through careful calibration of the device or probabilistic modeling within the framework of cluster-weighted modeling.

10.6.2 Marching Cello

The Marching Cello is a novel controller device for a cello that can be played standing up and can be carried around while being played (fig. 10-16). In the movie *Take the Money and Run* (1969), young Virgil, played by Woody Allen, gets to play cello in the local marching band. Carrying a chair and a cello he desperately runs after his fellow marching musicians, puts down the chair, plays for a couple of seconds, then gets up again and runs after his colleagues. Needless to say, the experience is frustrating.[33]

Our Marching Cello solves Virgil's problem. The Marching Cello is played like a cello and generates cello sound. Although it is strapped onto the player's body, it preserves the cello geometry and feel. The raw sensor data is wirelessly transmitted (range $\approx$ 100 feet) to a receiver with a lightweight Ethernet connection which forwards the data over the local area network to a PC computing the sound. The sound is played over a loudspeaker system [WEB].

10.6.3 A new violin MIDI controller

Many electric string instruments (plucked and bowed) provide the functionality of converting the electric audio signal into MIDI control data, which then can be reconverted into arbitrary sounds by a MIDI compatible synthesizer.[34] The converter technology is entirely based on information contained in the audio signal. Inevitably, there are accuracy problems with this technology. By definition pitch is only defined for sound sequences longer than one period. Hence there is a delay between fingering a note and knowing its pitch. For fast playing or noisy sound, pitch often can't be resolved at all.

The developed fingerboard sensing technology allows retrieval of the player's finger position directly from the sensing of the finger. Hence there are no issues with latency or noisiness of the pitch-tracking technology. Unlike commercial MIDI violins, which generate a minimum of acoustic energy and sound pressure, this MIDI converter needs no acoustic energy at all. Therefore a cello that is fingered without use of the right hand for plucking or bowing, can be mapped to MIDI sounds.

[33] A video clip of the scene is available from [WEB].
[34] ZETA violin, Yamaha silent string instruments.

In addition to the finger position, our instrumentation indicates the precise bow position on two axes as well as bow pressure. This information can be represented in the MIDI protocol or it can be processed and mapped into high-level control data, which in turn may be used with a MIDI synthesizer. Used in this manner the interface becomes a hyperinstrument very similar to the early work on the Hypercello [Ger99b]. The instrumented RAAD cello was used in this sense in a performance of *Meteor Music* for Hypercello and 16 Simple Things by Tod Machover.[35]

10.6.4 Flexible musical instruments: CD-player versus Strad

A Stradivarius violin and a CD-player very much represent opposite limits in the space of musical interfaces and devices. A Stradivarius gives the ultimate range of expressive freedom to a skilled player. Every detail and musical gesture are literally in the hands of the musician and the beauty of the music relies on the magnificence of the instrument as much as it relies on the skill and the creativity of the player. Unfortunately, in exchange for this expressive freedom, one has to invest almost a lifetime to learn the technical skills required for mastering the instrument.

On the other hand, CD-players and similar mass media require no skill at all, but still produce beautiful music. Any of these media, however, leave the user completely passive apart from providing a choice of tracks. There is no notion of expressiveness in a CD player whatsoever.

The space of musical devices is empty in between these two extremes. Digital sound synthesis along with new interfaces should help filling the space with interfaces that trade off expressiveness and usability. Not many people can afford the luxury of studying an instrument to the point of total mastery. Regardless, many people feel compelled to play an instrument because they love music. Instruments that provide a more flexible trade-off between expressiveness and ease of playing should be very appropriate for amateur musicians. For example, a violinist with limited proficiency may prefer playing a violin with discretized and well-tempered pitch to playing constantly out of tune. The decomposition of sound generation into sensing, representing, and predicting, as demonstrated above, provides the means to implement such an instrument.

[35] MIT Media Lab, October 1999.

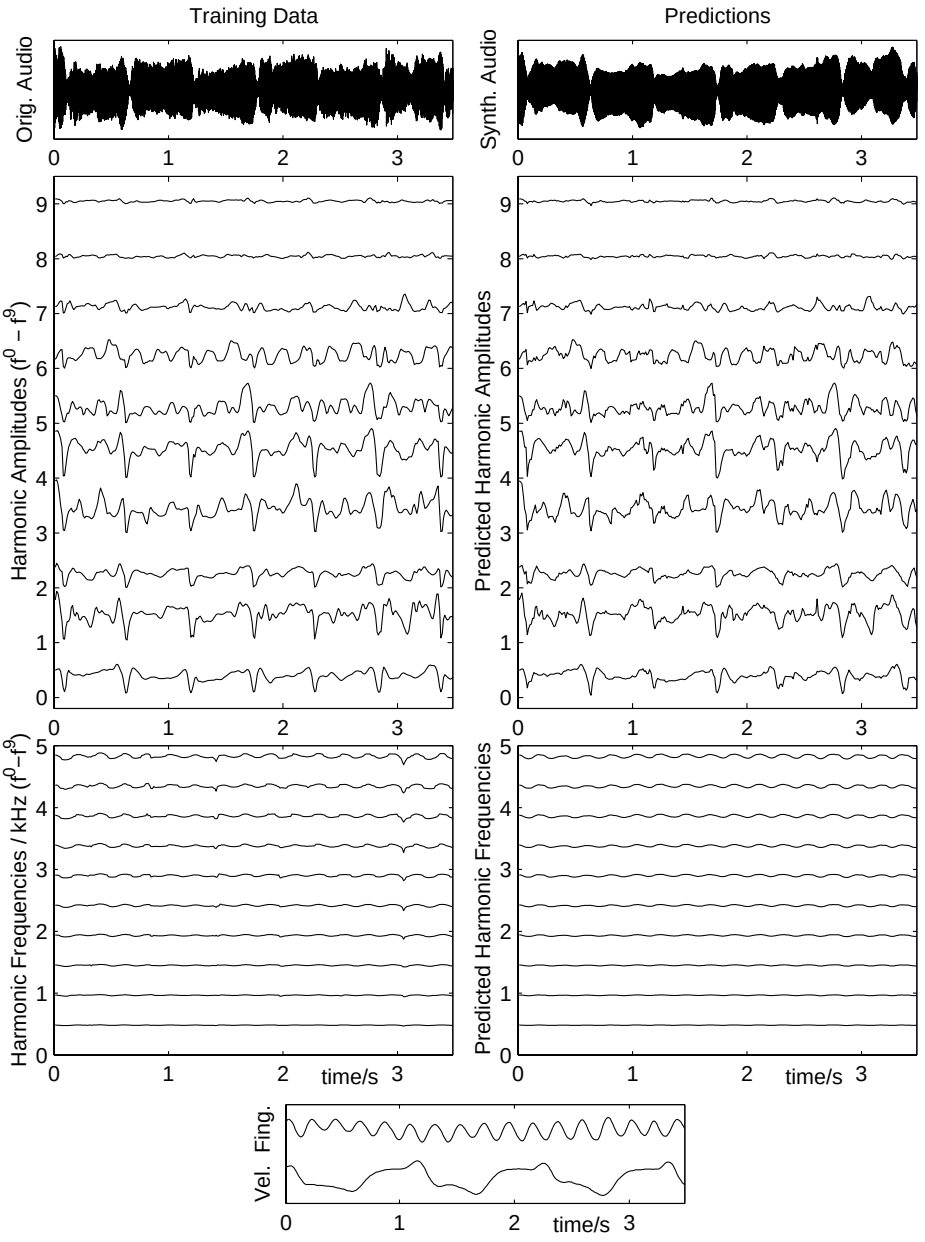


Figure 10-17: Comparison of original and predicted spectral violin data. *Bottom:* Input sensor measurements, showing the bow velocity (Vel.) and the player's finger position (Fing.). *Left:* Harmonic structure of the training data and the corresponding audio time series. *Right:* Harmonic structure and the estimated audio time series (15 clusters, locally linear models, 10s (860 points) of training data).

Chapter 11

Conclusions

11.1 Contributions

Cluster-weighted modeling is a contribution to probabilistic inference networks, more precisely Gaussian mixture models. CWM expands input-output data in a joint probability density which can be used to derive conditional probabilities and non-random estimators such as function approximation schemes. CWM differentiates itself from earlier architectures in a number of ways:

1. CWM uses the EM algorithm to allocate Gaussian basis functions and a LMS estimator (matrix inversion) to fit the local models. This fast and transparent parameter search finds the locally optimal parameters and converges to the globally best model reachable from the initial conditions.

2. CWM seamlessly and transparently integrates discrete and continuous, real and complex, deterministic and stochastic variables. The probabilistic framework allows for the evaluation of the data density in terms of unconditional and conditional probabilities, it allows for the derivation of non-random estimators and it allows for error and uncertainty estimation along with self-consistency checks.

3. CWM is general enough to be applicable to many different estimation and classification problems, and at the same time it is easy and quick to use. Unlike unconstrained Bayesian networks, the basic architecture does not change, which makes it instantly applicable to a large range of engineering applications.

4. CWM is a flexible and carefully designed architecture which can easily be extended if needed. Examples for such extensions that have been presented here include an online implementation, a cluster-weighted hidden Markov structure, and function approximation under constraints.

Linear systems theory and linear signal processing provide a multitude of useful tools, but many applications have reached limits where the assumption of linearity doesn't make sense any more. By seamlessly incorporating linear techniques in the CWM framework, we have pushed the concept of mixture models toward novel algorithms and application

domains and have created powerful nonlinear tools. We have developed CWM implementations for basic nonlinear classification and estimation problems and have created novel synthesis algorithms within the framework.

1. Cluster-weighted Kalman filtering was presented to recursively estimate the coefficients of locally linear models. The notion of having multiple recursive, linear estimators that are selectively updated based on the state of the system is a powerful idea that should be useful for many fields and applications.

2. Cluster-weighted complex-valued estimation and cluster-weighted Volterra modeling were developed to estimated complex valued functions. Real-valued linear algebra has been replaced by the complex-valued equivalent to make CWM applicable for important application domains, e.g. nonlinear device characterization.

3. Cluster-weighted linear predictive coding was developed to unify the successful excitation/filter models in a coherent framework.

4. Likewise, cluster-weighted sampling extends and unifies the idea of a global sampler, creating a tool that is flexible enough to handle rapidly changing sounds with continuous control. While global wavetable synthesis suffers from a lack of expressiveness and flexibility, CWS retains the benefits of good sound quality, and, in addition, is applicable to systems and instruments with continuous control.

Inference-based synthesis of acoustic instruments is a novel approach to mimic synthesis of existing sounds. Our approach makes use of well-known techniques for the analysis and synthesis of sounds, but is also clearly distinct from earlier work:

1. Our synthesis is based on physical input data, such as the bowing gesture data of a violinist. While there are earlier attempts to map perceptual time series data into audio data, the idea of deriving a quasi-physical model from physical input-output data is new.

2. Our approach is data-driven, i.e. all the model parameters are inferred from recorded data in a joint input-output space. While inference in music is not new, the specific application of reconstructing sound from scratch, based on an inference model, is.

3. Cluster-weighted sampling extends the idea of global wavetable synthesis into arbitrary synthesis applications with arbitrary interface and input space. In particular instruments with after-touch, such as violin-family instruments, become tractable with this new technique.

4. New instrumentation has been developed that extends earlier technology for violin bow sensing. A fingerboard sensor was designed that resists the impact of the string while giving a precise position measurement. Compared to earlier instrumentation, sensors and circuitry are packaged in a more flexible way, allowing for the instrumentation to be used in different devices ranging from recording devices to synthesis controllers as unusual as the Marching Cello.

11.2 Future Work

Naturally, our violin/cello model leaves room for improvements. The sound quality should be closer to that of a master acoustic instrument, the model should extrapolate better, and the instrument should be more responsive. Apart from these improvements, there are also fundamentally different, and novel, takes on the problem of building a digital violin.

Early in this thesis, we claimed that physical modeling of musical instruments suffers from two major problems: it is both computation and model-estimation bound. The former problem is taken care of by additional time. Waveguide models already run in real time; soon there will be no speed limitations for other implementations of first-principle physical models as well. However, the estimation of physical models remains a problem that should be worthwhile to attack.

In the course of this work, the author has built up a certain frustration with digital sound and with models that clearly lack the liveliness and flexibility of an acoustical instrument even after careful tuning. The most careful tweaking of hyper-parameters of an inference model can't really preserve the spectacular responsiveness and the often surprising features of the physical object. By definition the model is only as good as the data that it is inferred from. As of now, we do not handle extrapolation explicitly. Our model is general only in so far as we provide the means of estimating any instrument, but not in that we find general rules for instrument design.

The benefit of a physical model is precisely its generality. If the governing equations are implemented correctly, the response to any player action is that of the acoustic instrument. In order to recover this freedom, yet at the same time retain the benefit of inferring parameters, future research should consider data-driven physical modeling. Based on some numerical physical primitives, the estimator would find the optimal allocation of those primitives with respect to some training data. Primitives should be high-level to start with, but could increasingly become less sophisticated and increase in number in order to accommodate the subtleties of an acoustic instrument. It should be interesting to develop a cluster-weighted modeling hybrid that allocates simple physical units in a meaningful way to create a globally-powerful physical model.

The author would like to see applications of his instruments, explicitly or implicitly mentioned in this thesis, developed. These instruments and new hybrids should be used for performances, be it in a classical concert setting or in an installation/performance setting. The possibilities of combining the new technology with other digital technology and with traditional acoustic objects are endless. Also, there are many conceivable applications in musical education, which would be worthwhile to pursue. Music students should have access to instruments that give feedback and make suggestions for possible improvements on their playing rather than just passively sounding bad. The learning process could certainly be sped up and the pain associated with learning a difficult instrument could be reduced. A digital violin interface and synthesis algorithm are good starting points for a sophisticated interactive learning program.

Appendix A

Three views of the EM algorithm

The machine learning community shares the appreciation of the power of EM for training networks, but also is collectively humbled by how it works. Many times the author of this thesis had the impression that *he finally got it*, just to be disturbed again a minute later by some aspect of this *magic* algorithm. We present three views and approaches that prove the convergence of the EM algorithm. In the order presented, the three approaches increase in generality and conceptual beauty, but also in terms of theoretical and terminological overhead.

A.1 Gradient approach

The first idea is the one closest to the CWM notation, the least abstract and in many ways the most intuitive approach. We want to show that any iteration of Expectation and Maximization step only increases the data-likelihood.

Consider the log-likelihood L of the data-set [Ger99a]. We differentiate L with respect to the cluster positions $\mathbf{m}_k$:

$$\begin{aligned}
\nabla_{\mathbf{m}_k} L &= \nabla_{\mathbf{m}_k} \log \prod_{n=1}^{N} p(\mathbf{y}_n, \mathbf{x}_n) \qquad \text{(A.1)} \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{m}_k} \log p(\mathbf{y}_n, \mathbf{x}_n) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} \nabla_{\mathbf{m}_k} p(\mathbf{y}_n, \mathbf{x}_N) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \cdot \mathbf{C}_k^{-1} \cdot (\mathbf{x}_n - \mathbf{m}_k) \\
&= \sum_{n=1}^{N} p(c_k | \mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_k^{-1} \cdot \mathbf{x}_n - \sum_{n=1}^{N} p(c_k | \mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_k^{-1} \cdot \mathbf{m}_k
\end{aligned}$$

$$= \quad Np(c_k) \cdot \mathbf{C}_k^{-1} \cdot \underbrace{\left[\left(\frac{1}{Np(c_k)} \sum_{n=1}^{N} \mathbf{x}_n p(c_k|\mathbf{y}_n, \mathbf{x}_n) \right) - \mathbf{m}_k \right]}_{\delta \mathbf{m}_k}$$

Now consider the maximization update of the cluster means in Section 3.2, equation 3.17) and notice that it equals the term in bracket in equ. A.2. We rewrite the change in the cluster means as $\delta \mathbf{m}_k$ and obtain

$$\delta \mathbf{m}_k = \frac{\mathbf{C}_k}{Np(c_k)} \cdot \nabla_{\mathbf{m}_k} L \quad . \tag{A.2}$$

From this we see that the mean moves in the direction of the log-likelihood scaled by the covariance matrix and likelihood of the cluster $(1/(Np(c_k)))$. Since the covariance matrix is always positive definite, the update of the mean increases the likelihood of the data set at each maximization step.

A similar reasoning proves the convergence with respect to other parameters in the model, such as the coefficients of the local models:

$$\begin{aligned}
\nabla_{a_{i,k}} L &= \nabla_{a_{i,k}} \log \prod_{n=1}^{N} p(\mathbf{y}_n, \mathbf{x}_n) \qquad (A.3) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{y}_n, \mathbf{x}_n)} p(\mathbf{y}_n, \mathbf{x}_n, c_k) \cdot \mathbf{C}_{y,k}^{-1} \cdot [\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k)(-f_i(\mathbf{x}_n))] \\
&= \sum_{n=1}^{N} p(c_k|\mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_k^{-1} \cdot \mathbf{y}_n(-f_i(\mathbf{x}_n)) \\
&\quad - \sum_{n=1}^{N} p(c_k|\mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_{y,k}^{-1} \cdot \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k)(-f_i(\mathbf{x}_n)) \\
&= Np(c_k) \cdot \mathbf{C}_k^{-1} \cdot \\
&\quad \underbrace{\left[\left(\frac{1}{Np(c_k)} \sum_{n=1}^{N} \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k) f_i(\mathbf{x}_n) p(c_k|\mathbf{y}_n, \mathbf{x}_n) \right) - \left(\frac{1}{Np(c_k)} \sum_{n=1}^{N} \mathbf{y}_n f_i(\mathbf{x}_n) \right) \right]}_{\nu}
\end{aligned}$$

The maximization step finds the parameters $\mathbf{a}_k$, for which ν is zero. Since

$$\begin{aligned}
\nabla_{a_{i,k}}^2 L &= -\sum_{n=1}^{N} p(c_k|\mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_{y,k}^{-1} \frac{\partial f}{\partial a_{i,k}} \cdot \mathbf{f}(\mathbf{x}_n, \mathbf{a}_k)(-f_i(\mathbf{x}_n)) \\
&= \sum_{n=1}^{N} p(c_k|\mathbf{y}_n, \mathbf{x}_n) \cdot \mathbf{C}_{y,k}^{-1} \cdot f_i(\mathbf{x}_n)^2 \\
&\geq 0
\end{aligned}$$

is strictly positive, L is concave and the point where the first derivative of L vanishes is an absolute minimum.

A.2 Information theoretic approach

The information theoretic approach establishes a strong analogy between EM and statistical mechanics [NH93, Jor98b]. The problem is posed based on the distinction between observed and hidden variables. Suppose we observe some phenomenon but assume that it is dependent on some underlying variables or states which we don't know. The observed data, i.e. the known states, are denoted Z. The hidden data are denoted Y. The joint probability of both Z and Y, forming the complete set of data X, is parameterized by the set of parameters Θ, the cluster parameters, giving $P(Z, Y|\Theta)$ and a marginal probability for Z $p(Z|\Theta) = \sum_Y P(Y, Z|\Theta)$. We wish to find the parameterization of the joint distribution that maximizes the log-likelihood $L(\Theta) = \log P(Z|\Theta)$.

In the *E-step* of the EM algorithm we update the conditional distribution of hidden data, given known data and parameters: $P_t(Y) = P(Y|Z, \Theta_{t-1})$. In the *M-step* we re-compute Θ maximizing $E_{P_t}[\log P(Y, Z|\Theta)]$ [NH93]. The following convergence proof closely follows [Jor98b].

We need to maximize the log-likelihood $\log P(Z|\Theta)$, with respect to Θ. The task would be easy if the hidden nodes were observed, since they are not we can only work with an average $\log P(z, y|\Theta)$. Introducing an average function $Q(Y|Z)$ and using Jensen's inequality we get the following bound,

$$\begin{aligned} \log p(Z|\Theta) &= \log \sum_Z p(Y, Z|\Theta) \qquad \text{(A.4)} \\ &= \log \sum_Y Q(Y|Z) \cdot \frac{Y, Z|\Theta}{Q(Y|Z)} \\ &\geq \sum_Y Q(Y|Z) \log \frac{p(Y, Z|\Theta)}{Q(Y|Z)} \\ &= \sum_Y Q(Y|Z) \log p(Y, Z|\Theta) - \sum_Y Q(Y|Z) \log Q(Y|Z) \qquad \text{(A.5)} \end{aligned}$$

where the right hand depends on the choice of $Q(Y|Z)$ and the parameters Θ.

In the E-step we maximize expression (A.4) with respect to $Q(Z|Y)$, using $Q(Y|Z) = p(Y|Z, \Theta)$. This gives $p(Z|\Theta)$ on the right hand side. In the M-step we maximize with respect to the parameters Θ, maximizing the first term in (A.4).

Neal and Hinton make a similar argument [NH93]. They suggest that E and M steps are both increasing the function $F(P, \Theta)$, which is defined as

$$\begin{aligned} F(P, \Theta) &= E_P[\log P(Y, Z|\Theta)] + E_p[\log P(Y)]) \qquad \text{(A.6)} \\ &= E_P[\log P(Y, Z|\Theta)] + H(P) \end{aligned}$$

This expression corresponds to the physical free energy in statistical physics: the first term evaluates to the energy in the system, while the second one equals the entropy of the system. The free energy of a physical system can only increase, which is the property of EM we want to demonstrate.

A.3 Differential-geometric approach

A.3.1 The curved space of exponential distributions

Amari et al. [AKN92, Ama95] enriched the field of machine learning with an information geometric description of probability and estimation theory. Part of Amari's groundbreaking work is a representation and explanation of EM in a curved space described by differential geometry. In this approach sets of probability distributions become manifolds in a high dimensional parameter and data space and instances of networks are represented as points on those manifolds.

Let's consider a network characterized by a set of parameters $\mathbf{\Theta} = (\Theta_1, ..., \Theta_n)$. Then all the possible networks characterized by Θ form an n-dimensional manifold of networks. If the network is probabilistic these manifolds stand for probability distributions. A subspace of the manifold describes those distributions that can be reached by a given network architecture. Once again we distinguish between known and hidden variables. Using this distinction we define a manifold S that describes all the probability distributions over the hidden and observed variables. The data manifold D, a sub-manifold of S, contains only those distributions that are compatible with the observed data. In other words, when S is collapsed into D, the dimensions describing the observed data are fixed at the observed points. Another sub-manifold of S, the manifold M, contains all those distributions that are achievable given the parameterization of the network. The estimation process then aims at finding the distribution in M, that is closest to D relative to some metric. The metric will be the Kullbach-Leibler distance (KL) between the two distributions. It is the objective of this section to demonstrate the E-step and M-step in S and the convergence of EM to a local minimum of KL [Ama95]. Fig. A-1 illustrates the basic elements of this approach.

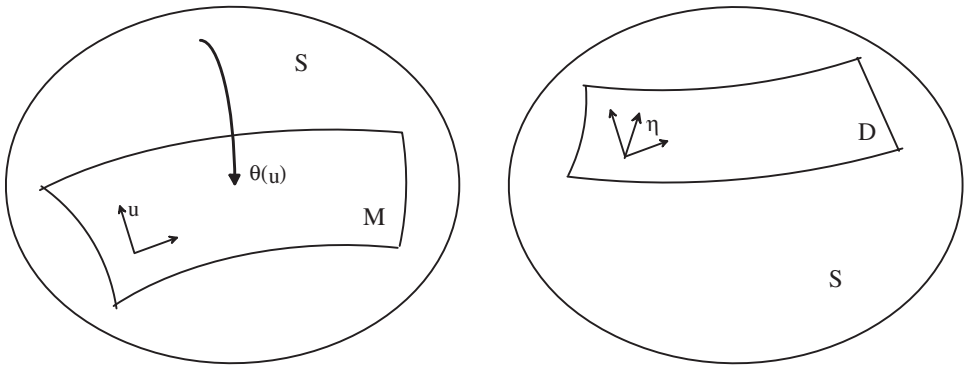


Figure A-1: Curved data-manifold D and model-manifold M in S.

A key concept in information geometry (and machine learning in general) is the family of exponential distributions, which includes all probability distributions that can be

parameterized as

$$p(x,\Theta) = \exp\left\{\sum_{i=1}^{n} \Theta_i r_i(x) + k(x) - \Psi(\Theta)\right\} \quad . \tag{A.7}$$

Simple examples are the normal distribution $p(x;m,\sigma^2) = 1/\sqrt{2\pi\sigma^2}exp\{-\frac{(x-m)^2}{2\sigma^2}\}$ and discrete distributions with $\mathbf{p} = (p_0, p_1, ..., p_n)$ and $p_i = \text{Prob}\{x = i\}$. The parameters m and σ can be easily matched with the parameters in equ. A.7 [Ama95]. The coefficients of the discrete distribution can be parameterized as

$$\begin{aligned} \Theta_i &= \log\frac{p_i}{p_0}, i = 1, ..., n \\ r_i &= \delta_i(x), i = 1, ..., n \quad . \end{aligned} \tag{A.8}$$

We are most interested in mixture distributions with hidden variables such as

$$p(y,z) = \sum_{i=0}^{I} \delta_i(z) p_i \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\{-\frac{(x_i - m_i)^2}{2\sigma_i^2}\} \quad . \tag{A.9}$$

where z is hidden and $\mathbf{x}$ is observed.

$$\begin{array}{llllll} r_{11} = x, & & \Theta_{11} = \frac{\mu_0}{\sigma_0^2} \\ r_{12} = x^2, & & \Theta_{12} = -\frac{1}{2\sigma_0^2} \\ r_{2i} = \delta_i(z), & & \Theta_{2i} = \log\frac{p_i\sigma_0}{p_0\sigma_i} - (\frac{\mu_i^2}{2\sigma_i^2} - \frac{\mu_0^2}{2\sigma_0^2}) \\ r_{3i} = x\delta_i(z), & & \Theta_{3i} = \frac{\mu_i}{\sigma_i^2} - \frac{\mu_0}{\sigma_0^2} \\ r_{4i} = x^2\delta_i(z), & & \Theta_{4i} = -(\frac{1}{2\sigma_i^2} - \frac{1}{2\sigma_0^2}) \end{array} \tag{A.10}$$

maps into a member of the exponential family. Mixtures without the term $\delta(z)$ are not part of the exponential family.

Usually we are dealing with more than one observation, which is why we are looking at the product space of all the observations $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_T$. Assumed an exponential distribution the joint distribution is given by

$$\begin{aligned} p(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_T; \Theta) &= \prod_{t=1}^{T} p(\mathbf{r}_t; \Theta) \\ &= \exp\Big\{(\sum \mathbf{r}_t) \cdot \Theta - T\phi(\Theta)\Big\} \quad , \end{aligned} \tag{A.11}$$

which is also an exponential distribution.

The observation of the data set corresponds to a point $\hat{\eta} = \mathbf{r}$ in S, which most likely is not part of the model manifold M. The best model to choose is the one that is as close to $\hat{\eta}$ as possible, i.e. the one that maximizes the Likelihood of M. This in turn means minimizing the Kullback-Leibler $K(\hat{\Theta}||\Theta(u))$ divergence between the observed point $\hat{\Theta}$ and M, parameterized in terms of $\Theta(u)$. The operation of projecting $\hat{\Theta}$ onto M is called m-projection (fig. A-1) and corresponds to the m-step of *em* (fig. A-2).

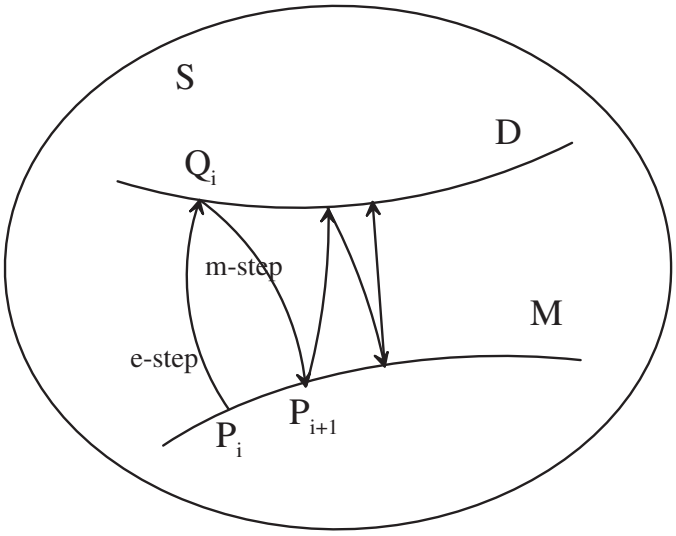


Figure A-2: Data-manifold D, model-manifold M, as well as E-step and M-step in S.

Since part of the random variables are hidden, part of the data-manifold D can be chosen freely. We parameterize the data-manifold in terms of η_v, the observed data dimensions, and η_h, the hidden data dimensions (fig. A-1). Since the η_h can be chosen arbitrarily, while the η_v are fixed we obtain a sub-manifold of dimension h, parameterized by η_h

$$D = \{\hat{\eta} = (\hat{\eta}_v, \hat{\eta}_h) | \hat{\eta}_v = \mathbf{r}_v,\ \hat{\eta}_h : \text{arbitrary}\} \quad . \tag{A.12}$$

The projection onto D is called the e-projection, which corresponds to the e-step in em (fig. A-2).

A.3.2 EM and em algorithm

Distributions of the exponential family can be parameterized in two distinct ways. Since both parameterizations are *flat*, i.e. two distributions are connected by straight lines in either parameterization, exponential distributions form a dually flat manifold.

Given two distributions p_1 and p_2, a parameterization of the curve $t \in [0 \quad 1])$, and a normalization factor $\Psi(t)$, we can connect $\log p_1$ and $\log p_2$ to obtain

$$\log p(x;t) = (1-t)\log p_1(x) + t \log p_2(x) \quad . \tag{A.13}$$

Alternatively this can be written as

$$p(x;t) = e^{t \cdot \frac{p_2(x)}{p_1(x)} + \log p_1(x) - \Psi(t)} \tag{A.14}$$

Yet another representation of this geodesic is in terms of the parameter $\Theta(t)$ of the exponential family,

$$\Theta^*(t) = (1-t)\ \Theta_1 + t\ \Theta_2 = \Theta_1 + t\ (\Theta_2 - \Theta_1) \quad . \tag{A.15}$$

From this expression it is clear that the connection is a linear projection (geodesic) in Θ-coordinates [Ama95].

Alternatively, we can make a connection in the mixture parameterization $p(x)$.

$$p^*(x,t) = (1-t)\ p_1(x) + t\ p_2(x) \tag{A.16}$$

We observe that the mixture parameterization yields a straight line in η-coordinates.

$$\eta^*(t) = \eta_1 + t\ (\eta_2 - \eta_1) \tag{A.17}$$

A straight connection in Θ-coordinates is called an e-geodesic, and the manifold S is e-flat in Θ. A straight line in η-coordinates is called an m-geodesic, and the manifold S is m-flat in the coordinate system η. The e-geodesics are different from the m-geodesics and vice versa.

Before explaining em we also need to introduce the divergence $K(P,Q)$ which for members of the exponential family can be written as

$$K(\Theta_P, \eta_Q) = \Psi(\Theta_P) + \phi(\eta_Q) - \Theta_{PiQi} \quad . \tag{A.18}$$

where P is parameterized in terms of Θ_P and Q is parameterized in terms of η_Q. K equals the KL divergence

$$KL(P,Q) = E_{\Theta_P}\left[\log \frac{p(r;\Theta_P)}{p(r;\Theta_Q)}\right] \quad . \tag{A.19}$$

with

$$\begin{aligned} K(P,Q) &\neq K(Q,P) \\ K(P,Q) &\geq 0 \\ K(P,Q) &= 0\ , \quad \text{for} \quad P = Q \end{aligned} \tag{A.20}$$

The importance of the K-divergence lies in the property illustrated in figure A-3. Given three distributions P, Q and R in a dually flat manifold, such that in Q the m-geodesic connecting P and Q is orthogonal to the e-geodesic connecting Q and R, then

$$K(P,R) = K(P,Q) + K(Q,R) \quad . \tag{A.21}$$

We furthermore define e and m projections of a distribution on a sub-manifold in our dually flat space. Assuming we are given a sub-manifold of distributions M and a distribution S that is not part of M, we want to find the distribution Q in M that is closest to S. Since the divergence is asymmetric there are two solutions for Q. The point $\hat{Q}$ that minimizes the distance $P(P,Q)$, $Q \in M$, is given by the m-projection of P to M. It is unique if M is an e-flat sub-manifold. The point $\hat{Q}$ that minimizes the distance $P(Q,P)$, $Q \in D$, is given by the e-projection of P to D. It is unique if D is an m-flat sub-manifold [Ama95].

Let's recall our initial situation with a model manifold M and a data manifold D (fig. A-1), as well as the parameterization of the m-flat manifold D in terms of $\eta = (\eta_v, \eta_h)$ and the e-flat manifold M in terms of $\Theta = (\Theta_v, \Theta_h)$. It is our intent to find the points

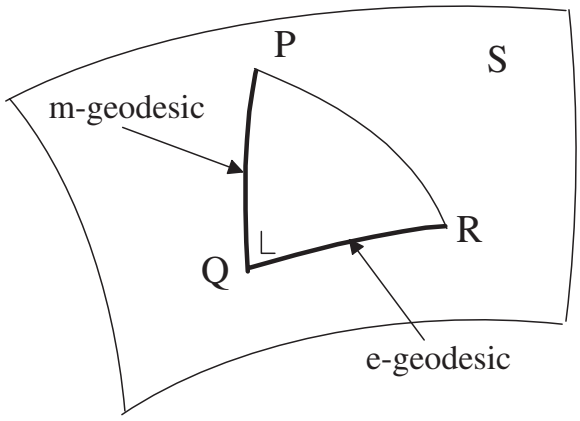


Figure A-3: Generalized Pythagoras theorem [Ama95].

$\tilde{P} \in M$ and $\tilde{Q} \in D$ that minimize the divergence $K(P||Q)$. By now we know how to find the $\hat{Q}$ that minimizes $K(\hat{Q}||P)$. Given a fixed P we e-project P onto D. Since the sub-manifold D is m-flat, the e-projection is orthogonal to D in $\hat{Q}$. The projection is linear in η-coordinates and solves

$$\begin{aligned} \eta_v^* &= \frac{\partial}{\partial \Theta_h} \Psi(\Theta_v^*, \Theta_h^P) \qquad \text{(A.22)} \\ \eta_v^* &= c \end{aligned}$$

and

$$\begin{aligned} \Theta_v^* &= \frac{\partial}{\partial \eta_v} \phi(\eta_v^*, c) \qquad \text{(A.23)} \\ \Theta_h^* &= \Theta_h^P \end{aligned}$$

Likewise we know how to find the $\hat{P}$ that minimizes $K(Q||\hat{P})$. Given a fixed Q we m-project Q onto M. Since the sub-manifold M is e-flat, the m-projection is orthogonal to M in $\hat{P}$. The projection is linear in η-coordinates and solves equations similar to (A.22).

The em-algorithm uses these updates iteratively as illustrated in fig. A-2 . We summarize the algorithm:

1. Choose an initial distribution $P_0 \in M$.
2. **e-step:** e-project the current P onto D (A.22). This results in $\hat{Q} \in D$ which minimizes $K(\hat{Q}||P)$.
3. **m-step:** m-project the current Q onto M. This results in $\hat{P} \in M$ which minimizes $K(Q||\hat{P})$.
4. Go back to (2), unless the algorithm has converged to a minimum of $K(Q||P)$.

EM and em algorithm are equivalent for most relevant cases. [Ama95] proves this equivalence and discusses exceptions as well as efficiency differencies.

Appendix B

Sensor technology and hardware design

B.1 Sensor technology

B.1.1 Sensing the bow

The technology used for providing bow data extends earlier work by Paradiso and Gershenfeld [PG97, Sch96]. More measurements have been added and have been compactly packaged.

Fig. B-6 illustrates the basic functional elements of the bow sensing technology. The bow is instrumented with four oscillators, implemented as square wave generators using low power 555 timer chips. The first oscillator (25kHz) is placed at the frog and is driving a chain of 20 $2K$ resistors running along the bow bar. The other end of the resistor series is connected to the output pin of the second oscillator (35kHz), placed at the tip of the bow. The resistor chain functions as a voltage divider between signal level and ground. The signal strength decreases linearly over the length of the bow, in opposite directions for the two oscillators. Both signals couple capacitively into an antenna mounted underneath the string. Depending on the position of the bow relative to the strings the two received signals vary in amplitude. Bowing close to the frog causes the signal from oscillator one to be predominant while bowing close to the tip causes the other signal to be strong.

The third oscillator (49kHz), placed on the frog, drives a wire antenna running along the bow. The signal couples capacitively into a receiving antenna mounted at the bridge of the violin/cello. The signal amplitude varies depending on the distance between the bow and antenna. Hence the amplitude indicates the bow-bridge distance.

The fourth oscillator (64-72kHz), placed on the frog, is frequency-modulated and also driving an antenna running along the bow. A force-sensitive strip is mounted on the bow at the position of the forefinger of the bow hand. When the player increases the pressure on the bow, the resistance of the sensor strip decreases. This change is used to modulate the control voltage of the timer chip, resulting in a maximal frequency change of about 8kHz. The signal couples into the bridge antenna and is frequency demodulated with respect to the pressure change. Alternatively the force-sensitive strip can be mounted between bow hair and wood at the tip of the bow [TC99]. This design measures the actual force

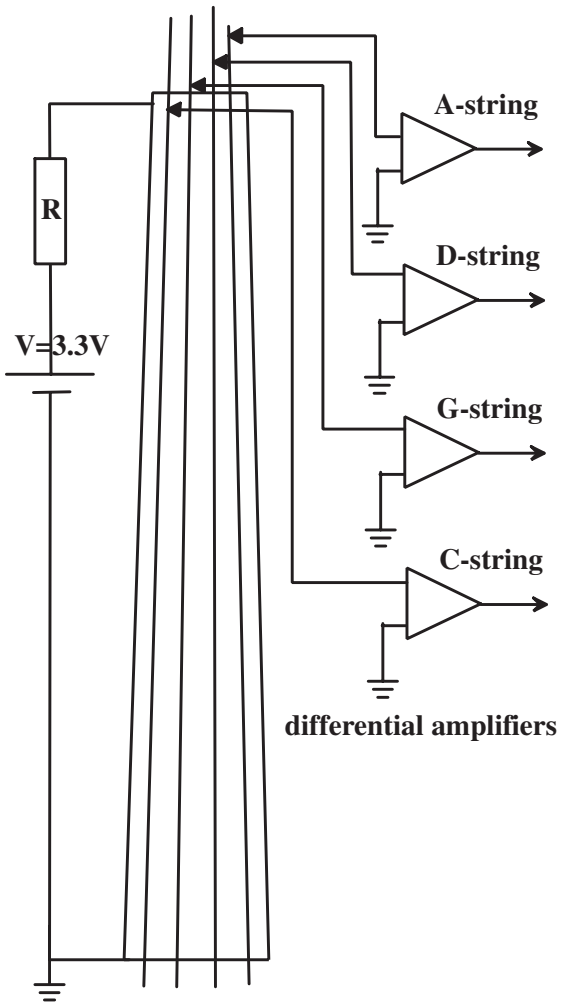


Figure B-1: Functional diagram of the fingerboard.

of the bow on the string, as opposed to the force seen by the player. Unfortunately, the measurement is not as sensitive as the first alternative, and the values depend on the bow position relative to the strings.

The copper antenna pieces are connected to a FET source follower which in turn connects to the signal-conditioning board. Four band-pass filters (two stage, second order state-variable filters) select the different carriers (fig. B-2 and B-3). The isolated signals are amplified. Signal 1-3 (frog signal $u_f(t)$; tip signal $u_t(t)$; bridge signal $u_y(t)$) are rectified and low-pass filtered to detect the amplitude of each signal. The fourth signal is amplified and used in a PLL chip which locks on the carrier signal. The PLL provides a DC voltage indicating the instantaneous frequency of the carrier $u_p(t)$ (pressure).

$u_f(t)$ and $u_t(t)$ are combined to provide the left-right position of the bow

$$u_x(t) = \frac{u_t(t) - u_f(t)}{u_t(t) + u_f(t)} \quad . \tag{B.1}$$

B.1.2 Sensing fingers on the fingerboard

The position of the finger on the fingerboard is measured using a stainless steel strip mounted on the fingerboard (fig. B-1). The one thousands of an inch thick strip is covering the violin or cello fingerboard providing a total resistance of about one Ohm over the full length. A constant voltage of about 100 mV is applied at the end of the fingerboard facing the bridge, creating a voltage gradient over the length of the strip. The strings function as pickup electrodes wherever a finger is pressing the string on the fingerboard. Since the strings are isolated from one another, each string corresponds to an independent sensor channel allowing for double stops to be resolved. The pickup voltages are amplified with a series of two high input impedance differential amplifier stages, converting the voltages to cover the input range of the analog-to-digital converter chip.

The stainless steel was chosen, because it is strong enough to mechanically resist the pressed string and a the same time is electrically resistive. Higher resistivity would be preferable but barely comes along with mechanically strong materials. The DC measurement requires a clear contact between the string and the fingerboard. This is typically the case for cello playing, but not necessarily for violin playing. The alternative of AC-based capacitive coupling between string and fingerboard has been discarded since a capacitive measurement does not return a precise voltage at the point of contact but smears out over the length of the string. The current measurement does not allow a player to press with two fingers on the same string. Although players tend to support the *pitch*-finger with fingers lower on the string, they typically don't push the support finger down all the way. The only hand position that is not supported by this system is the thumb position of the cellist.

B.1.3 Recording violin audio signals

Different commercial audio recording devices are used. An electric RAAD cello[1] is used which has an integrated pickup system mounted to the top plate of the instrument. The acoustic violin is recorded with a small guitar microphone that fits through the f-holes. Our machine-learning approach is very sensitive to amplitude variations that are not caused by the player action, rather by unrelated causes such as room acoustics or displacement between instrument and microphone. By mounting a microphone inside the instrument (without interfering with the acoustical properties) these issues are eliminated. The sound picked up inside an instrument is arguably not the same as the sound projected into the room, yet the advantages of the transducer mounted close to the instrument were by far more important for this work.

Two types of custom string-sensing technology are used. A dynamic pickup system records the signal on the string itself. Small permanent magnets are mounted under each

[1] made by Richard Armin, RAAD Instrument Inc., Toronto, Canada.

string at the end of the fingerboard. The voltages induced by the vibrating strings are picked up between the two ends of each string. The voltages are amplified separately by two stages of differential amplifier circuits.

A piezo pickup system is used to record the audio signals at the bridge. Little pieces of piezo electric material are placed between the strings and the bridge. The induced voltages are conditioned and amplified separately.

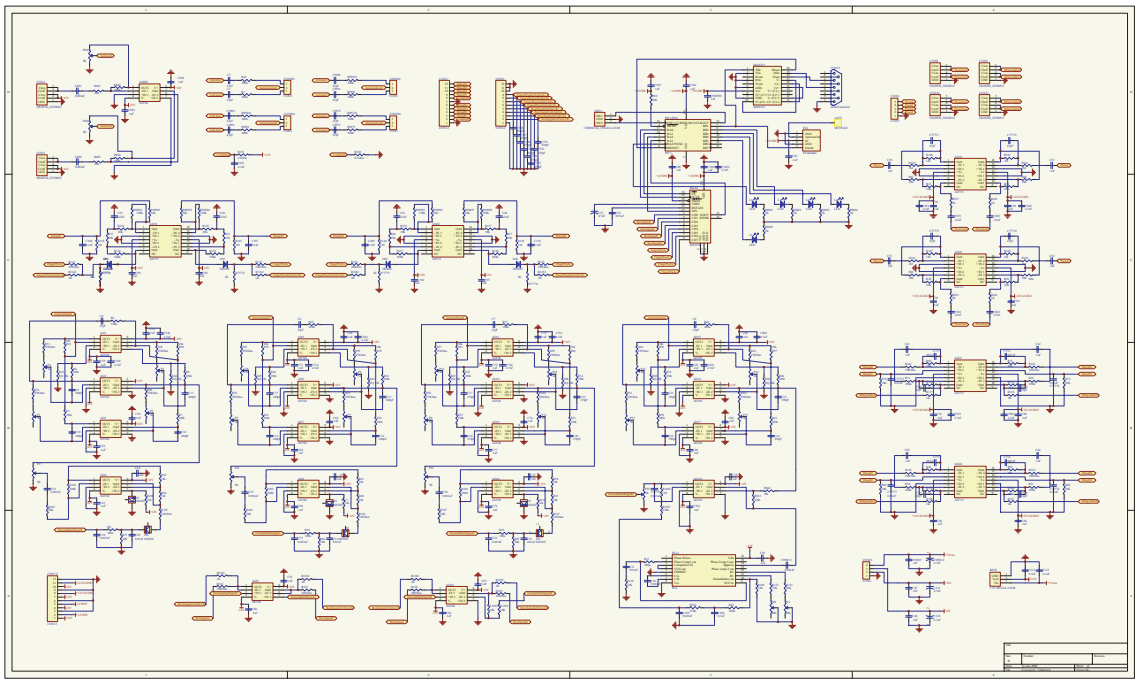


Figure B-2: Violin/cello sensor board - schematic. See [WEB] for a digital copy of the schematic.

B.2 Hardware packaging

B.2.1 Violin sensor board

The violin sensor board has been designed to accommodate any instrument of the violin family (fig. B-3). It has been packaged

1. to be easily mounted on the different instruments. All the analog data processing is done on the sensor board. A microprocessor generates serial data that is wirelessly transmitted (T) to a little receiver unit connecting to the network. Alternatively the analog sensor data can be sent off to a DAQ system (AD).

2. to be stand-alone. With a battery pack providing the operating power, the sensing is wireless. In order for the audio recording to be wireless, off-the-shelf audio transmitters could be connected to any of the recording channels.

3. to easily be adapted to different instrument geometries and tasks. The antenna geometry varies between violin (one antenna) and cello (two antennas). Little shorting jumpers (AJ) connect any of the bow-sensing channels to any of the antennas.

4. in three functional parts: a) the analog bow and fingerboard processing (AP), b) the digital conversion and encoding of the sensor data (D), and c) the audio processing for the string pickups (AU). Parts b) and c) can be powered down independently with shorting jumpers (PJ) in order to reduce noise. In addition the audio part (AU) can be cut off to reduce the size of the board when only the sensor interface is needed.

Fig. B-2 illustrates the schematic of the sensing board and the bow instrumentation, Fig. B-3 shows the layout of the board. Supporting documents are available from [WEB].

B.2.2 Instrumenting violin and cello

Fig. 10-6 shows the sensing board mounted in the back of the cello and sensors mounted on the violin. In the case of the violin, a single antenna mounted next to the bridge picks up the four bow signals. The instrument is grounded through the copper-taped chin rest.

The cello instrumentation uses two antennas. One is mounted next to the bridge and picks up the signal representing the bow-bridge distance and the pressure-modulated signal. A second antenna running underneath the strings between fingerboard and bridge picks up the left-right bow signals. Unlike the violin, the cello does not require additional ground return to the player.

B.2.3 Instrumenting the Marching Cello

The Marching Cello interface (fig. 10-16) uses the same sensor geometry as the instrumented RAAD cello. Two antennas are mounted on a little piece of fiberglass. The antenna piece is attached to the fingerboard but can easily be removed and mounted on any part of the player's body. The player decides if he bows his knee, leg, or belly ... .

The fingerboard consists of four strips of stainless steel mounted like cello strings. The strips are connected in parallel. Every finger position on any of the strings maps into a unique voltage. Since there are no strings that can function as pickup electrodes for the finger position voltage, a copper-taped foil is mounted on top of the stainless steel. The foil is held apart from the steel by thin spacing material on either side of the strip. When the player pushes on the fingerboard the foil makes contact with the stainless steel, but it returns to its rest position immediately after release. Every string-electrode connects to a separate signal-conditioning channel.

B.3 Sensor calibration

The sensor technology provides repeatable measurements at essentially millimeter and millisecond resolution. However, the raw digital data needs a calibration step for a number of reasons:

1. The numerical scaling of the raw sensor data is meaningless. For example, the bow position should be given in units relative to a rest position, yet, the 12 bit ADC spits out an arbitrary number between 0 and 4096. Meaningful scaling of the raw values helps in the modeling process since often high-level information is needed to understand and guide the model or to process special cases.

2. Different ADC instrumentation is used for different purposes. For data collection, an off-the-shelf data acquisition board is used, that allows for the simultaneous collection of 13 channels (8 sensor channels + 5 audio channels) at 22050 kHz.[2] Since separate audio and data collection boards are difficult to synchronize, this solution is preferred to operating with a multitude of recording devices. For real-time synthesis, an onboard ADC chip,[3] a PIC-microprocessor, [4]a one-way digital radio chip,[5] and a lightweight ethernet board[6] are used to communicate the sensor data to a PC running the model (updaterate $\approx$ 50Hz).

3. Different sensor instruments were used for recording and for interfacing to the model. Data was collected on a violin and a cello. The synthesis interface was yet another cello-like device (section B.2.3). Although the type of measurements and the sensor technology were the same, the raw values differed due to different instrument geometries. Cross-synthesis between these input devices was only possible through careful calibration to meaningful parameters.

4. Some of the measurements are distorted to start with, meaning that linear scaling is not sufficient to turn them into meaningful values. Fig. B-4 shows some raw measurements and their transformation into the final calibrated information. It can be seen how the bow-bridge distance is dependent on the bow position relative to the strings. This is due to the finite-length antenna running along the bow, which causes signals to drop at the ends of the bow. Given the second measurement of left-right bow position and a nonlinear mapping, this dependency can be eliminated. Hence calibration also helps to eliminate unwanted artifacts of the measurement technology.

Instead of eliminating these effects one by one, we compensate for all of them in one functional mapping of the raw sensor data.

Figure B-5 illustrates the calibration points of the bow position. A grid of 21 x 4 points is recorded (bow-string distance x bow-bridge distance). These points serve as data points for a polynomial model. The targets correspond to a grid covering a two-dimensional space ranging from 0 to 1 in each axis. Data and targets are used to train a polynomial model, that is, a CWM model that is collapsed into a single cluster. Second to fifth order polynomials were used. The raw data is projected onto meaningful position data.

Since the resistive measurement of the finger position is assumed linear, the fingerboard is mapped using two calibration points only: one semi-tone above the open-string and

[2] National Instruments DAQ Board , 16 bit, 500kBit/s, 16 channels
[3] MAXIM technologies, MAX186.
[4] Micorchip PIC16F84.
[5] Radiometrix TX/RX units.
[6] Filament board MIT Media Lab [OPH+00].

two octaves above the open string. A linear (CWM) model is used to map this interval onto $\lambda \in [\frac{1}{\sqrt[12]{2}} \quad \frac{1}{4}]$, which corresponds to the appropriate wavelength modification. The frequency then equals

$$f = f_0 * \frac{1}{\lambda} \tag{B.2}$$

where f_0 is the frequency of the open string.

The CWM calibration approach has also been used to calibrate other computer-human interface devices. Examples include the laser-range finder [SP98] and a polyphonic floor-board for the Flying Karamazov Brothers [RSR+00].

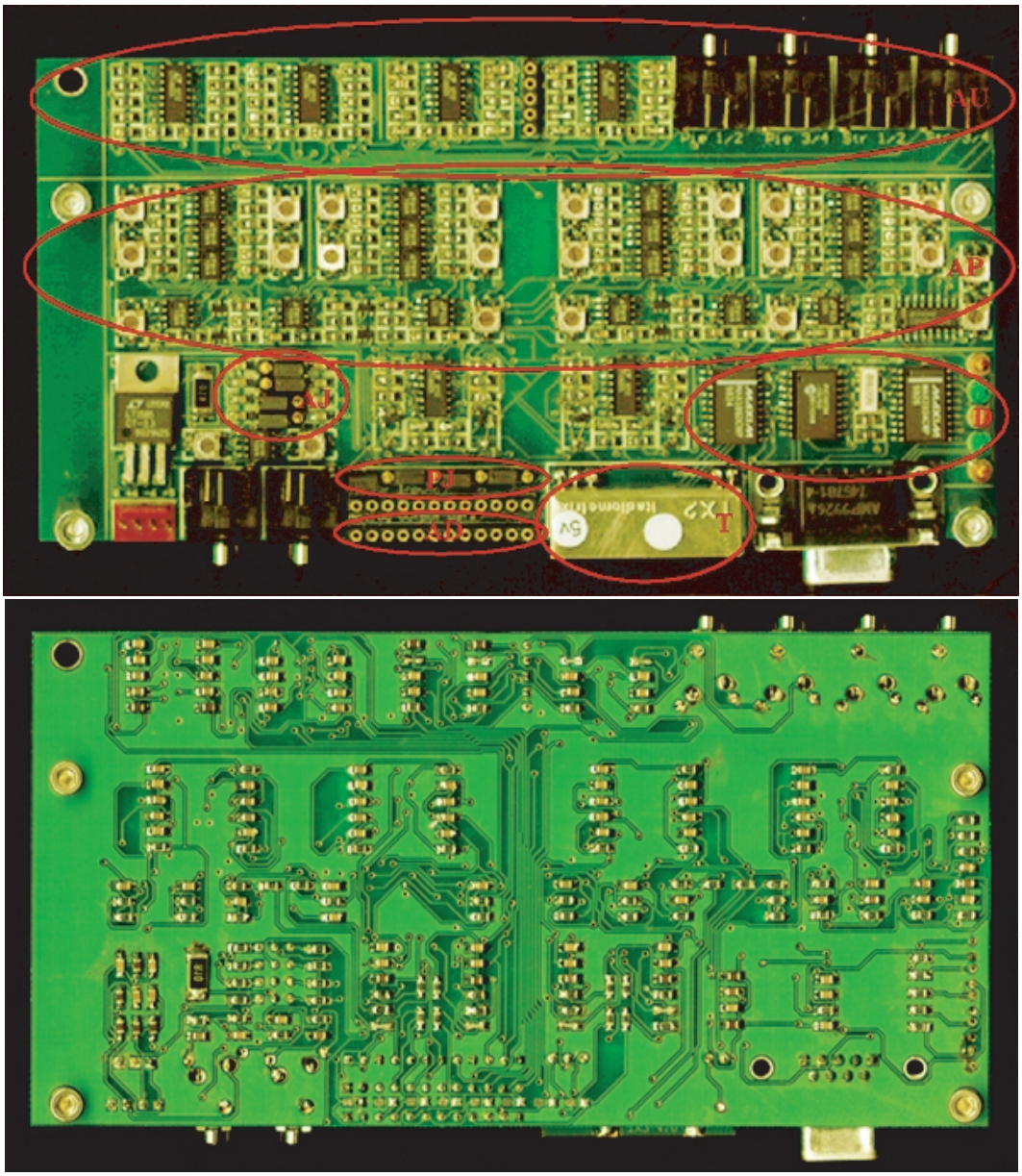


Figure B-3: Violin/cello sensor board. Front and (vertically flipped) back. **AP**: analog processing; **AU**: audio processing; **D**: digital processing; **T**: transmitter unit; **AD**: connector for analog data; **PJ**: power off jumpers; **AJ**: antenna jumpers.

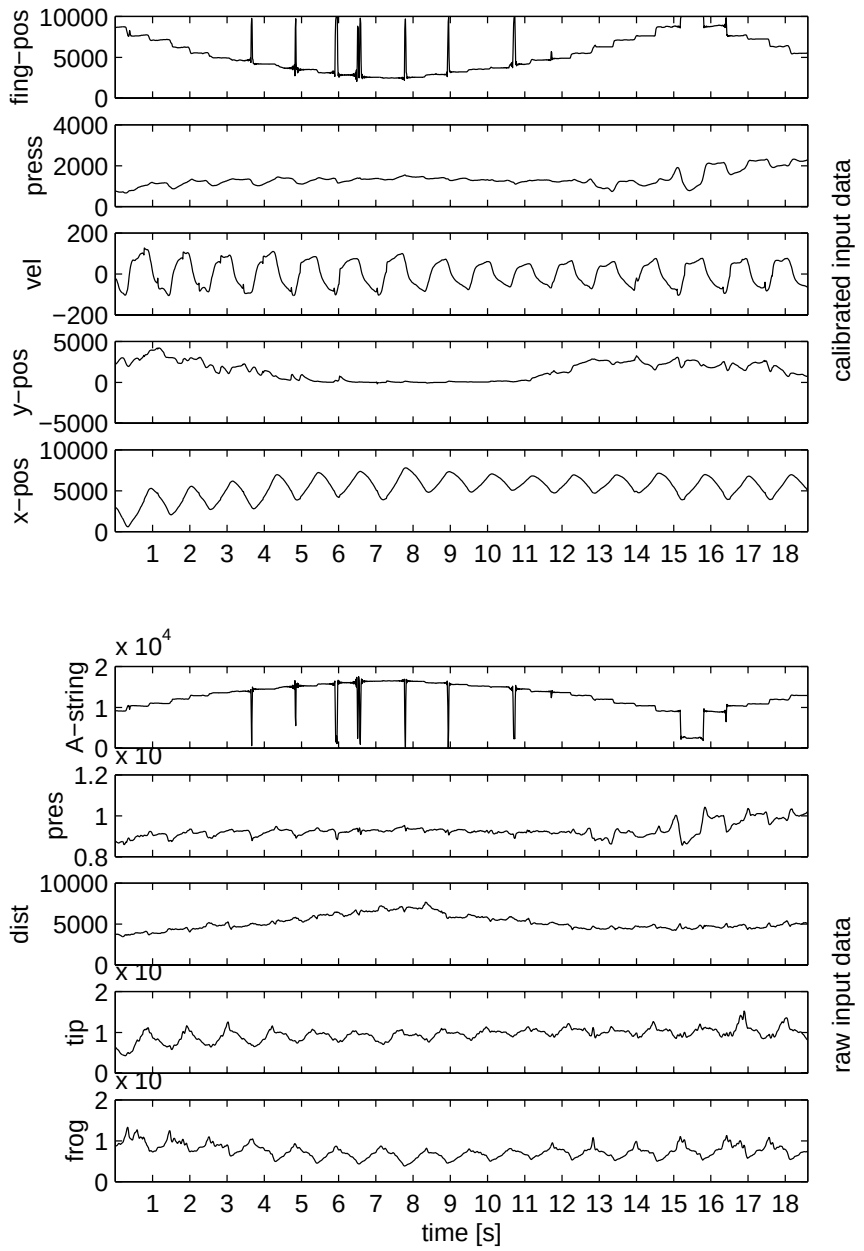


Figure B-4: Raw violin input data, along with calibrated data, and high-level information.

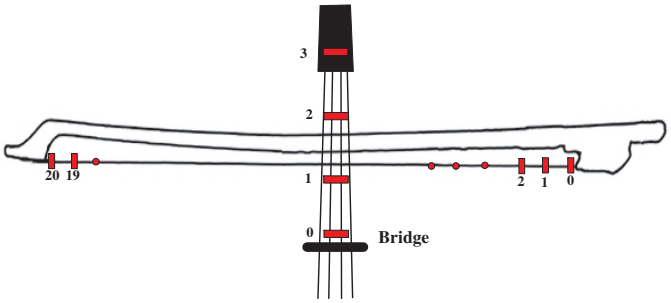


Figure B-5: Grid of measurement points in x and y direction for the bow calibration.

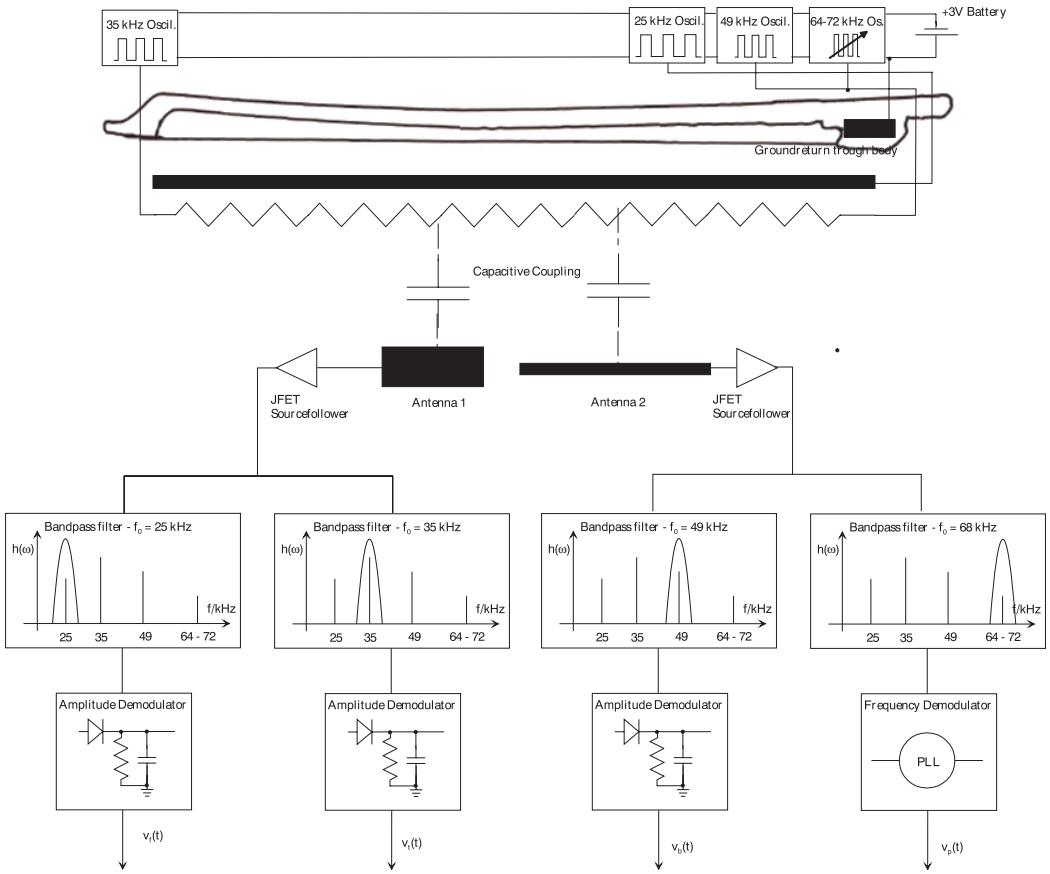


Figure B-6: Functional diagram of the bow sensing technology.

Appendix C

Code

C.1 Cluster-weighted modeling implementations

C.1.1 Matlab interface

The following function declarations constitute the Matlab interface to the cwm-code. The routines are commented in terms of input and output data structs that are visible to the user. Two example programs are given, illustrating the core functionality of the code (`detection_demo.m` and `prediction_demo.m`).

`cwm.m` trains a cwm model based on input-output training data. The function handles discrete input data (`w_d`) and real valued input data (`w`), as well as discrete output data (`y_d`) and real valued output data (`y`). It distinguishes between fast state vectors (`x`), and slow state vectors (`w`). It also handles autoregressive time series predicition (`dim_x_ar`, `dim_w_ar`).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cwm.m
% Bernd Schoner, (c) 1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datapoints, datastat, clusters, hmmodel]= ...
          cwm(w, w_d, x, y, y_d, nclusters, niterations, polyorder, covariance_w, covariance_y, ...
          dim_w_ar, lag_w_ar, dim_x_ar, lag_x_ar, nmodels, graphics)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% input data structs:
%      w -- the input data matrix for the weighting (slow dynamics). w is of dimension number of
%         points N times dim_w (weighting dimension).
%              _                   _
%             | x11, x12, ..., x1D |
%         w = | x21, x22, ..., x2D |
%             | ..., ..., ..., ... |
%             |_xN1, xN2, ..., xND_|
%
%      w_d -- discrete input data. Each input in w_d effectively causes a different model to be
%         built. w_d is of dimension N times dim_w_d.
%      x -- the input matrix which the data regression is based on. x is of dimension N times dim_x
%         (regression dimension).
%      y -- the output matrix. y is of dimension N times dim_y, where the output is real
%         valued.
%      y_d -- discrete ouput data. Each different input in y_d defines a label which is predicted
```

```
%          by the model. y_d is of dimension N times dim_y_d.
%       nclusters -- scalar, >0. Number of clusters allocated.
%       niterations -- scalar, >0 Number of Expectation-Maximization iterations. niterations should
%          be chosen in a such a way that the data likelihood fully converges. If nclusters==1 =>
%          niterations=1 .
%       polyorder -- scalar, >=0. polynomial degree of the local models (0=constant, 1=linear,
%          2=quadratic ...).
%       covariance_w -- =0: variance input clusters. =1: covariance input clusters.
%       covariance_y -- =0: variance output clusters. =1: covariance output clusters.
%       dim_w_ar -- number of lagged dimensions in the weighting domain. =0 for A typical
%          non-time-series applications.
%       lag_w_ar -- vector of lagged input values (length>=dim_w_ar);
%       dim_x_ar -- number of lagged dimensions in the regression domain. =0 for A typical
%          non-time-series applications.
%       lag_x_ar -- vector of lagged input values (length>=dim_x_ar);
%       nmodels -- number of HMM states if an HMM structure is used. if =0 no HMM is allocated.
%       graphics -- =1: graphics are updated during iterations. =-1: no graphics,
%          =0: graphics are shown after convergence.
%
% output data structs:
%       datapoints -- struct containing the packaged data information.
%       datastat -- struct containing mean and standard deviation of the input data.
%       clusters -- struct containing cluster info of the converged model.
%       hmmodel -- struct containing hmm info of the converged hmm model. Empty if nmodels==0.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

`predict.m` is the natural partner of `cwm.m`. It takes the model parameters generated by `cwm.m` and predicts new output given new input. New input data should be allocated using `init_datapoints.m`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% predict.m
% B.Schoner, (c) 1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function datapoints_pred = predict(datapoints, datastat, clusters, hmmodel, pointpred0freerunning1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% input data structs:
%       datapoints -- struct containing the packaged data information.
%       datastat -- struct containing mean and standard deviation of the input data.
%       clusters -- struct containing cluster info of the converged model.
%       hmmodel -- struct containing hmm info of the converged hmm model model. Empty if nmodels=0.
%       pointpred0freerunning1 -- =0: pointprediction (autoregressive models), =1: freerunning
%          iterative model.
%
% output data structs:
%       datapoints_pred -- struct that contains the predicted data. The struct equals datapoints,
%          except for the members y_pred and y_d_pred which contain the predicted continuous and
%          discrete data.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

`init_datapoints.m` should be used to allocated new (out-of-sample) input data, to be used with `predict.m`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% init_datapoints.m
% Bernd Schoner, (c) 1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function datapoints = init_datapoints(w, w_d, x, y, y_d, dim_w_ar, lag_w_ar, dim_x_ar, lag_x_ar)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% input data structs:
%      w -- the input data matrix for the weighting (slow dynamics). w is of dimension number of
%         points N times dim_w (weighting dimension).
%               _                  _
%              | x11, x12, ..., x1D |
%          w = | x21, x22, ..., x2D |
%              | ..., ..., ..., ... |
%              |_xN1, xN2, ..., xND_|
%
%      w_d -- discrete input data. w_d is of dimension N times dim_w_d.
%      x -- the input matrix which the data regression is based on. x is of dimension N times dim_x.
%      y -- the output matrix. y is of dimension N times dim_y.
%      y_d -- discrete output data. Each different input in y_d defines a label which is predicted
%         by the model. y_d is of dimension N times dim_y_d.
%      dim_w_ar -- number of lagged dimensions in the weighting domain. =0 for a typical
%         non-time-series applications.
%      lag_w_ar -- vector of lagged input values (length>=dim_w_ar);
%      dim_x_ar -- number of lagged dimensions in the regression domain. =0 for typical
%         non-time-series applications.
%      lag_x_ar -- vector of lagged input values (length>=dim_x_ar).
%
% output data structs:
%      datapoints -- struct containing the packaged data information.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

plot_model.m creates graphical output based on a trained model and a dataset.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot_model.m
% Bernd Schoner, (c)1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plot_model(datapoints, clusters, hmmodel, ...
                    figure_number, inputdimensions)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% input data structs:
%      datapoints -- struct containing the packaged data information.
%      datastat -- struct containing mean and standard deviation of the input data.
%      clusters -- struct containing cluster info of the converged model.
%      hmmodel -- struct containing hmm info of the converged hmm model model. Empty if nmodels=0.
%      figure_number -- figure for graphics.
%      inputdimensions -- input dimensions to plot. Two-dimensional vector [v1 v2],
%         with 1<=vi<=dim_w.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

show_detection_results.m creates a classification matrix for a detection problem.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% show_detection_results.m
% Bernd Schoner, (c)1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function show_detection_results(datapoints.y_d, datapoints.y_d_pred)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% input data members
%       datapoints.y_d -- discrete target vector.
%       datapoints.y_d_pred -- predicted targets.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

`detection_demo.m` reads in two-dimensional labeled data, builds a classifier CWM model and then labels the same data based on that model.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% detection_demo.m
% Bernd Schoner, (c) 1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% READ SYNTHETIC DATA FROM FILE

data_file_name='testdata.dat';
data_file=fopen(data_file_name,'r');
i=0;
while (1)
   i=i+1;
   itemp = fscanf(data_file,'%d',1);
   if feof(data_file)
      break
   end
   record(i,1) = itemp;
   state(i,1) = fscanf(data_file,'%d',1);
   data(i,1) = fscanf(data_file,'%f',1);
   data(i,2) = fscanf(data_file,'%f',1);
end
N=i-1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOOSE PARAMETERS

w=data;
w_d=[];
x=data;
y=[];
y_d=state;
dim_x_ar=0;
lag_x_ar=0;
dim_w_ar=0;
lag_w_ar=0;
nclusters=6;
niterations=15;
polyorder=0;
covariance_w=0;
covariance_y=0;
pointpred0freerunning1=0;
graphics=1;
nmodels=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRAIN THE MODEL

[datapoints, datastat, clusters, hmmodel]= cwm(w, w_d, x, y, y_d, nclusters, niterations, ...
         polyorder, covariance_w, covariance_y, dim_w_ar, lag_w_ar, dim_x_ar, lag_x_ar, ...
         nmodels, graphics);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USE THE MODEL TO CLASSIFY DATA

datapoints_pred = predict(datapoints, datastat, clusters, hmmodel, pointpred0freerunning1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT DETECTION MATRIX

show_detection_results(datapoints_pred.y_d, datapoints_pred.y_d_pred);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

`prediction_demo.m` reads in a time series (Laser data, from the Santa Fe time series competition) and builds an autoregressive model. It then predicts the time series in a one-step-ahead scheme (point predictions) and in a freerunning scheme.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prediction_demo.m
% Bernd Schoner, (c) 1/99
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD LASER DATA

N=2000;
a=zeros(1,2*N);
data_file = fopen('a.dat','r');
for (i=1:2*N)
    a(i)=fscanf(data_file,'%f',1);
end
fclose(data_file);
a=a';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOOSE PARAMETERS

w=[];
w_d=[];
x=[];
y_d=[];

y=a(151:N+150);
dim_x_ar=4;
lag_x_ar=[10 10 10 10];
dim_w_ar=4;
lag_w_ar=[10 10 10 10];
nclusters=10;
niterations=15;
polyorder=1;
covariance_w=0;
covariance_y=0;
nmodels=0;
graphics=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRAIN THE MODEL

[datapoints, datastat, clusters, hmmodel]= cwm(w, w_d, x, y, y_d, nclusters, niterations, ...
         polyorder, covariance_w, covariance_y, dim_w_ar, lag_w_ar, dim_x_ar, lag_x_ar, ...
         nmodels, graphics);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% POINT-PREDICT LASER DATA

pointpred0freerunning1=0;
datapoints = predict(datapoints, datastat, clusters, hmmodel, pointpred0freerunning1);
y_pointpredicted=datapoints.y_pred;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREDICT FREERUNNING LASER DATA

pointpred0freerunning1=;
datapoints = predict(datapoints, datastat, clusters, hmmodel, pointpred0freerunning1);
y_freepredicted=datapoints.y_pred;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT THE DATA

figure(1)
clf
subplot(3,1,1)
plot(y)
ylabel('original laser signal')
zoom xon
subplot(3,1,2)
plot(y_pointpredicted(1:N))
ylabel('point predicted signal')
zoom xon
subplot(3,1,3)
plot(y_pointpredicted(1:N))
ylabel('predicted freerunning signal')
zoom xon

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.1.2 C interface

The following function declarations constitute the C interface to the cwm-code. The routines are commented in terms of the input and output data structs, visible to the user.

`cwm()` trains a cwm model based on input-output training data. The function handles discrete input data (`*x_d`) and real valued input data (`*x`) as well as discrete output data (`*y_d`) and real valued output data (`*y`). It distinguishes between fast state vectors (`*x`) and slow state vectors (`*w`). It also handles autoregressive time series predicition (`dim_x_ar, dim_w_ar`).

```
/****************************************************************************************************
* cwm20.c
* B. Schoner, (c) 3/99
****************************************************************************************************/

int cwm(struct s_clust *clust, struct s_hmm *hmm, struct s_datastatistics *datastatistics,
        int ndatapoints, int dim_w, double *w, int dim_w_d, int *w_d,
        int dim_x, double *x, int dim_y, double *y, int dim_y_d, int *y_d,
        int dim_w_ar, int dim_x_ar, int *lag_w_ar, int *lag_x_ar,
        int nclusters, int niterations, int polyorder,
        int covariance_w, int covariance_y, int nmodels);

/****************************************************************************************************
* input data structs:
*      dim_w -- number of weighting dimensions (slow dynamics).
*      w -- the input data matrix for the weighting. w  is of length number of points N times dim_w
*
```

```
*          w = | x11, x12, ..., x1D, x21, x22, ..., x2D,   ...  ,xN1, xN2, ..., xND|
*
*          This order is the same for the vectors to follow.
*      dim_w_d -- number of discrete input dimensions.
*      w_d -- discrete input data. Each input in w_d effectively causes a different model
*         to be built. w_d is of length N times dim_w_d.
*      dim_x -- number of regression dimensions.
*      x -- the input matrix which the data regression is based on. x is of length N times dim_x
*         (regression dimension).
*      dim_y -- number of real valued output dimensions.
*      y -- the output matrix. y is of length N times output dimension, where the output is real
*         valued.
*      dim_y_d -- number of discrete valued output dimensions.
*      y_d -- discrete ouput data. Each different input in y_d defines a label which is predicted
*         by the model. y_d is of length N times dim_y_d.
*      nclusters --  scalar, >0, number of clusters allocated.
*      niterations -- scalar, >0 number of Expectation-Maximization iterations. niterations should
*         be chosen in a such a way that the model fully converges. If nclusters==1 =>
*         niterations=1 .
*      polyorder -- scalar, >=0. polynomial degree of the local models (0=constant, 1=linear, ...).
*      covariance_w -- =0: variance input clusters. =1: covariance input clusters .
*      covariance_y -- =0: variance output clusters. =1: covariance output clusters .
*      dim_w_ar -- number of lagged dimensions in the weighting domain. =0 for typical
*         non-time-series applications.
*      lag_w_ar -- vector of lagged input values (length>=dim_w_ar);
*      dim_x_ar -- number of lagged dimensions in the regression domain. =0 for typical
*         non-time-series applications.
*      lag_x_ar -- vector of lagged input values (length>=dim_x_ar);
*      nmodels -- number of HMM states if an HMM structure is used. if =0, no HMM is allocated.
*
* output data structs:
*      clust -- struct containing cluster info of the converged model.
*      hmm -- struct containing hmm info of the converged hmm model model. Empty if nmodels=0.
*      datastatistics -- struct containing mean and standard deviation of the input and output data.
*
*********************************************************************************************/
```

`predict()` is the natural partner of `cwm()`. It takes the model parameters generated by `cwm()` and predicts new output given new input.

```
/*********************************************************************************************/

int predict(struct s_clust *clust, struct s_hmm *hmm, struct s_datastatistics *datastatistics,
            int ndatapoints, double *w, int *w_d, double *x, double *y, int *y_d,
            double *y_pred, int  *y_d_pred, int pointpred0freerunning);

/*********************************************************************************************
*
* input data structs:
*      clust -- struct containing cluster info of the converged model.
*      hmm -- struct containing hmm info of the converged hmm model model. Empty if nmodels=0.
*      datastatistics -- struct containing mean and standard deviation of the input and output data.
*
*      w -- the input data matrix for the weighting. w is of length number of points N times dim_w.
*      w_d -- discrete input data. w_d is of length N times dim_w_d.
*      x -- the input matrix which the regression is based on. x is of length N times dim_x.
*      y -- the output matrix. y is of length N times dim_y.
*      y_d -- discrete output data. y_d is of length N times dim_y_d.
*      pointpred0freerunning1 -- =0: pointprediction, =1: freerunning iterative model.
*
* output data structs:
*      y_pred -- the output vector y is of length number of points N times dim_y.
*      y_d_pred -- discrete output vector. y_d is of length number of points times dim_y_d.
*
```

```
*****************************************************************************************/
```

The following data structs are allocated by the user.

```
/*****************************************************************************************
* visible data structs
*****************************************************************************************/

struct s_clust {
   int dim_w;
   int dim_w_d;
   int dim_w_ar;
   int lag_w_ar[DIM_W_TOTAL_MAX];
   int dim_x;
   int dim_x_ar;
   int lag_x_ar[DIM_X_TOTAL_MAX];
   int dim_y;
   int dim_y_d;
   int polyorder;
   int covariance_w;
   int covariance_y;
   int nclasses_w_d_total;
   int nclasses_w_d[DIM_W_D_MAX];
   int nclasses_y_d[DIM_Y_D_MAX];
   int nc_classes_total;
   int npolynomials;
   int nclusters;

   double **inputmean;
   double ***inputcovariance;
   double ***outputcovariance;
   double ***outputmodel;
   double **inputdistribution;
   double ***outputdistribution;
   int    **exponents;
   double ***A;
};

struct s_hmm {
   int nmodels;
   int nclusters_per_model;
   int *modelindex;
   double *PI;
   double **a;
};

struct s_datastatistics {
   double *mean_w;
   double *mean_x;
   double *mean_y;
   double *var_w;
   double *var_x;
   double *var_y;
};

/*****************************************************************************************/
```

C.2 Sinusoidal synthesis

The class `c_ss` is used for real-time sinusoidal synthesis.

```
/*****************************************************************************************/
#define SAMPLE_RATE 22050
#define FRAME_SIZE 256
#define NHARMONICS 25
#define PI 3.141592654

struct s_state
{
float old_amplitudes[NHARMONICS];
float old_frequencies[NHARMONICS];
float phase[NHARMONICS];
};

class c_ss
{
public:
      /*****************************************************************************************
      * use init() to initialize the class
      *****************************************************************************************/

      int init();

      /*****************************************************************************************
      * use update() when new input data is available (86 time per second)
      *
      * amplitudes -- vector of new amplitudes (HAMRONICS long).
      * frequencies -- vector of new frequencies (NHARMONICS long).
      * audio_scaling -- static scaling of the output.
      * samples -- vector contains the updated samples (FRAME_SIZE long).
      *****************************************************************************************/

      int update(float *amplitudes, float *frequencies, float audio_scaling, short *samples);

      /*****************************************************************************************/

private:
      struct s_state state;

      float delta_amp[NHARMONICS];
      float delta_freq[NHARMONICS];
      float phase[NHARMONICS];

      float float_samples[FRAME_SIZE];
      float twopi_over_sample_rate;
};

/*****************************************************************************************/
```

C.3 Wavetable synthesis

The class `c_cws` is used for real-time wavetable synthesis (CWS).

```
/*****************************************************************************************/
#define SAMPLE_RATE 22050
#define FRAME_SIZE 256
#define NTHREADS_MAX 2
#define NFRAMES_PER_CHANGE_MAX  35
#define FADER_ENVELOPE_LENGTH 512
#define NZERO_CROSSINGS 10
#define NFRAMES_MIN_PER_CHANGE 15
#define NSAMPLES_PER_ZERO_CROSSING 256
#define PI 3.141592654
```

```
struct cws_state
{
      int ncurrent_audio_threads;
      int current_file_index[NTHREADS_MAX];
      int current_sample_index[NTHREADS_MAX];
      int nframes_since_last_change;

      float new_frame_target_pitch;
      float old_frame_target_pitch;
      float new_frame_target_envelope;
      float old_frame_target_envelope;

      int old_file_index;
      int old_sample_index;
      int file_index[NTHREADS_MAX];
      int sample_index[NTHREADS_MAX];

      float sample_pointer[NTHREADS_MAX];
      int fader_index;
};

class c_cws
{
public:
      /*********************************************************************************************
      * use init() to initialize the class
      *
      * data_filename -- files holding the CWS model and sample sequences.
      *
      *********************************************************************************************/

      int init(char data_filename[]);

      /*********************************************************************************************
      * use update() when new input data is available (86 time per second)
      *
      * envelope -- target envelope of the new frame.
      * pitch -- target pitch of the new envelope.
      * cluster_index -- index of cluster selected to present the model.
      * audio_scaling -- static scaling of the output.
      * samples -- vector contains the updated samples (FRAME_SIZE long).
      *
      *********************************************************************************************/

      int update(float envelope, float pitch, int cluster_index,
                 float audio_scaling, short *samples);

      /*********************************************************************************************/

      int find_sample_pointer(int new_file_index, int new_sample_index, float *new_sample_pointer,
      int old_file_index, float old_sample_pointer);
      int resample(int file_index, float *target_pitch, float *target_envelope,
            float old_sample_pointer, float *new_sample_pointer, int nsamples, float *new_samples,
            int *old_indeces);
      int clear_buffers();

private:
      struct cws_state state;
      struct cws_param param;
      float **audio_data;
      float **pitch_data;
      float **envelope_data;
```

```
      float target_pitch[FRAME_SIZE];
      float target_envelope[FRAME_SIZE];
      float delta_target_pitch;
      float delta_target_envelope;

      float thread_target_envelope[NTHREADS_MAX][FRAME_SIZE];
      float envelope_scaling[NTHREADS_MAX][FRAME_SIZE];
      float pitch_scaling[NTHREADS_MAX][FRAME_SIZE];

      float fadein_envelope[FADER_ENVELOPE_LENGTH];
      float fadeout_envelope[FADER_ENVELOPE_LENGTH];

      float new_samples[FRAME_SIZE];
      float float_samples[FRAME_SIZE];
      int   old_indeces[FRAME_SIZE];

      int resample_buffer_coeffs[2*NZERO_CROSSINGS+1];
      float resample_buffer[4*NZERO_CROSSINGS*NSAMPLES_PER_ZERO_CROSSING+1];
};

/*****************************************************************************************************/
```

Bibliography

[AKN92] Shunichi Amari, Koji Kurata, and Hiroshi Nagaoka. Information geometry of boltzmann machines. *IEEE Transactions on Neural Networks*, 3(2):260–271, 1992.

[Ama95] Shunichi Amari. Information Geometry of the EM and em Algorithms for Neural Networks. *Neural Networks*, 8(9):1379–1408, 1995.

[Arf79] D. Arfib. Digital synthesis of complex spectra by means of multiplication of non-linear distorted sine waves. *Journal of the Audio Engineering Society*, pages 757–779, 1979.

[Ass60] Amrican Standarts Association. American standart acoustical terminology. definition 12.9. timbre, 1960.

[Bar93] Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39:930–945, 1993.

[Bas80] M. Bastiaans. Gabor's expansion of a signal into gaussian elementary signals. *Proceedings of the IEEE*, 68:538–539, 1980.

[Bas85] M. Bastiaans. On the sliding-window representation of signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(4):868–873, 1985.

[Bea82] J.W. Beauchamp. Synthesis by spectral amplitude and 'brightness' matching of analyzed musical instrument tones. *J.Audio.Eng.Soc.*, 30(6):396–406, 1982.

[Bea93] J.W. Beauchamp. Unix workstation software for analysis, graphics, modification, and synthesis of musical sounds. *Audio Engineering Society Preprint*, (3479, L1-7), 1993.

[BH92] Robert Grover Brown and Patrick Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering.* New York, New York, 2nd edition, 1992.

[BJ76] G.E.P. Box and G.M. Jenkins. *Time Series Analysis: Forecasting and Control.* Holden-Day, 1976.

[BJ95] Robert Bristow-Johnson. A detailed analysis of a time-domain formant-corrected pitch-shifting algorithm. *J. Audio Eng. Soc.*, 43(5):340–352, 1995.

[Bla73] John Blacking. *How musical is man?* University of Washington Press, Seattle and London, 1973.

[BNP] M. Brand, N.Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Puerto Rico.

[Bos99] Mark Vanden Bossche, 1999. Personal communication.

[BP93] J.C. Brown and M.S. Puckette. A high resolution fundamental frequency determination based on phase changes of the fourier transform. *J. Acoust. Soc. Am.*, 94(2):662–667, 1993.

[Bro99] J.C. Brown. Musical instrument identification using pattern recognition with cepstral coefficients as features. *J. Acoust. Soc. Am.*, 105(3):1933–1941, 1999.

[BS91] I.N. Bronstein and K.A. Semendjajew. *Taschenbuch der Mathematik.* Nauka/Teubner/Deutsch, 25th edition, 1991.

[BS95] A.J. Bell and T.J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

[Bun94] W.L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[Bun96] W.L. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[Cas92] Martin Casdagli. A dynamical systems approach to modeling input-output systems. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting*, Santa Fe Institute Studies in the Sciences of Complexity, pages 265–281, Redwood City, 1992. Addison-Wesley.

[CD88] W.S. Cleveland and S.J. Devlin. Regression analysis by local fitting. *J. A. Statist. Assoc.*, 83:596–610, 1988.

[Cho73] J. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.

[CM89] F. Charpentier and E. Moulines. Pitch-Synchronous Waveform Prosessing Techniques for Text-to-Speech Synthesis Using Diphones. In *Proceedings of Eurospeech 89(2)*, pages 13–19, 1989.

[Cre84] Lothar Cremer. *The Physics of the Violin.* MIT Press, Cambridge, Massachusetts, 1984.

[CS86] F. Charpentier and M. Stella. Diphone Synthesis Using an Overlap-Add Technique for Speech Waveforms Concatenation. In *Proc. ICASSP*, pages 2015–2018, 1986.

[CS94] R.R. Coifman and N. Saito. Constructions of local orthonormal bases for classification and regression. *Comptes Rendus Acad. Sci. Paris, Serie I*, 2:191–196, 1994.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, 1991.

[CW92] R.R. Coifman and M.W. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38:713–718, 1992.

[dA47] J.l.R. d' Alembert. Investigation of the Curve formed by a Vibrating String. In *Acoustics: Historical and Philosophical Development*. Dowden, Hutchinson and Ross, Stroudsburg, 1747.

[Dau88] I. Daubechies. Orthonormal basis of compactly supported wavelets. *Comm. Pure Applied Math.*, 41:909–996, 1988.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood From Incomplete Data via the EM Algorithm. *J. R. Statist. Soc. B*, 39:1–38, 1977.

[DPR91] G. DePoli, A. Piccialli, and C. Roads. *Representations of Musical Signals*. MIT Press, 1991.

[FGN92] F. Filicori, G. Ghione, and C.U. Naldi. Physics based electron device modeling and computer-aided mmic design. *IEEE Transactions on Microwave Theory and Techniques*, 40/7:1333–1352, 1992.

[FR97] Fletcher and Rosig. *The Physics of Musical Instruments*. Springer, New York, 1997.

[Gab46] D. Gabor. Theory of communication. *Journal of the Institute of Electrical Engineers*, Part III(93):429–457, 1946.

[Gab47] D. Gabor. Acoustical quanta and the theory of hearing. *Nature*, 159(1044):591–594, 1947.

[Ger89] Neil A. Gershenfeld. An experimentalist's introduction to the observation of dynamical systems. In Bai-Lin Hao, editor, *Directions in Chaos*, volume 2, pages 310–384. World Scientific, 1989.

[Ger96] Neil Gershenfeld. Signal entropy and the thermodynamics of computation. *IBM Systems Journal*, 35:577–587, 1996.

[Ger99a] Neil Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, New York, 1999.

[Ger99b] Neil Gershenfeld. *When things start to think.* Henry Holt, New York, 1999.

[Ger00a] Neil Gershenfeld, 2000. personal communication.

[Ger00b] Neil Gershenfeld. *The Physics of Information Technology.* Cambridge University Press, New York, 2000.

[GJ96] Zoubin Ghahramani and M.I. Jordan. Factorial hidden markov models. *Advances in Neural Information Processing Systems*, 8, 1996.

[GJP95] Frederico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.

[GK92] G.N. Georgiou and C. Koutsougeras. Complex domain backpropagation. *IEEE Trans. Circuits Syst. Part II: Analog and Digital Signal Processing*, 39:330–334, 1992.

[Gre78] J. Grey. Timbre discrimination in musical patterns. *Journal of the Acoustical Society of America*, 64:467–472, 1978.

[GSM99] Neil A. Gershenfeld, Bernd Schoner, and Eric Metois. Cluster-weighted modeling for time series analysis. *Nature*, 379:329–332, 1999.

[GW93] Neil A. Gershenfeld and Andreas S. Weigend. The future of time series: Learning and understanding. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, Santa Fe Institute Studies in the Sciences of Complexity, pages 1–63, Reading, MA, 1993. Addison–Wesley.

[Hay96] Simon Haykin. *Adaptive Filter Theory.* Prentice-Hall, Upper Saddle River NJ, 3rd edition, 1996.

[Hay99] Simon Haykin. *Neural Networks.* Prentice-Hall, Upper Saddle River NJ, 1999.

[HD96] D. Hoernel and P. Degenhardt. A neural organist improvising baroque-style melodic variations. In *Proceedings International Computer Music Conference*, pages 59–62, Thessaloniki, Greece, 1996.

[Hib99] Michele Hibon, 1999. Personal communication.

[HKCH97] J.M. Hajda, R.A. Kendall, E.C. Carterette, and M.L. Harshberger. Methodological Issues in Timbre Research. In I. Deliege and J. Sloboda, editor, *Perception and Cognition of Music.* Psychology Press, East Essex, UK, 1997.

[HM98] D. Hoernel and W. Menzel. Learning musical structure and style with neural networks. *Computer Music Journal*, 22(4):44–62, 1998.

[HM00] M. Hibon and S. Makridakis. M3 - competition. *International Journal of Forecasting*, 2000. to appear.

[Hou97] A.J.M. Houtsma. Pitch and timbre: Definition, meaning and use. *Journal of New Music Research*, 26:104–115, 1997.

[HR96] D. Hoernel and T. Ragg. Learning musical structure and style by recognition, prediction and evolution. In *Proceedings International Computer Music Conference*, pages 59–62, Hong Kong, 1996.

[Hun92] Norman F. Hunter. Application of nonlinear time-series models to driven systems. In M. Casdagli and S. Eubank, editors, *Nonlinear modeling and forecasting*. Addison-Wesley, 1992.

[HW95] D. Heckerman and M. Wellman. *Bayesian Networks*. Communications of the Association Machinery. 1995.

[HWAT93] Udo Hüebner, Carl-Otto Weiss, Neal Abraham, and Dingyuan Tang. Lorenz-like chaos in nh_3-fir lasers. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, Santa Fe Institute Studies in the Sciences of Complexity, pages 73–105, Reading, MA, 1993. Addison–Wesley.

[JGJS99] M.I. Jordan, Z. Ghahramani, T.S. Jaakola, and L.K. Saul. An introduction to Variational Methods for Graphical Models. *Machine Learning*, 1999.

[JJ94] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[JJS93] N. Jayant, J.Johnson, and R. Safranek. Signal compression based on models of human perception. *Proc. IEEE*, 81(10):1385–1422, 1993.

[JLO90] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quaterly*, 4:269–282, 1990.

[Jor98a] Michael Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, Massachusetts, 1998.

[Jor98b] Michael I. Jordan. Graphical models and variational approximation, 1998. Tutorial slides.

[KMG91] R. Kronland-Martinet and A. Grossman. Application of time-frequency and time-scale methods (wavelet-transforms) to the analysis, synthesis, and transformation of natural sounds. In G. DePoli, A. Piccialli, and C. Roads, editors, *Representations of Musical Signals*, pages 45–85. MIT Press, 1991.

[Lar98] Jean Laroche. Time and pitch scale modification of audio signals. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*. Kluwer Academic Publishers, Boston, 1998.

[LeB79] M. LeBrun. Digital waveshaping synthesis. *Journal of the Audio Engineering Society*, pages 250–266, 1979.

[Len89] K. Lent. An efficient method for pitch shifting digitally sampled sounds. *Computer Music Journal*, 13:65–71, 1989.

[LS88] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[LSM00] Tuomas Lukka, Bernd Schoner, and Alec Marantz. Phoneme discrimination from meg data. *Elsevier Journal for Neuroscience, Special Issue on Neural Computation*, 2000.

[Mac92] Tod Machover. Hyperinstruments. A Progress Report 1987-1991. Technical report, MIT Media Laboratory, 1992.

[Mag00] Yael Maguire, 2000. Personal communication.

[Mar99] Keith Dana Martin. *Sound-Source Recognition. A Theory and Computational Model.* PhD thesis, MIT Media Lab, 1999.

[Mas98] Dana C. Massie. Wavetable sampling synthesis. In Mark Kahrs and Karlheinz Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 311–341. Kluwer Academic Publishers, 1998.

[Met96] Eric Metois. *Musical Sound Information. Musical Gestures and Embedding Synthesis.* PhD thesis, MIT Media Lab, 1996.

[Mey92] H. Meyr. *Regelungstechnik und Systemtheory I und II.* RWTH Aachen, Aachen, 1992.

[MG76] J.D. Markel and A.H. Gray. *Linear prediction of speech.* Springer-Verlag, New York, 1976.

[MGOP+97] B. Mallet-Guy, Z. Ouarch, M. Prigent, R. Quere, and J. Obregon. A distributed, measurement based, nonlinear model of fets for high frequencies applications. In *Microwave Symposium Digest of IEEE 1997 MTT-S International*, pages 869–872, New York, 1997. IEEE.

[Min85] Marvin Minsky. *The Society of Mind.* Touchstone, New York, 1985.

[Moo78] J.A. Moorer. The use of the linear prediction of speech in computer music application. *Rapport IRCAM*, 6, 1978.

[MQ85] R.J. McAulay and T.F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. Technical Report 693, Massachusetts Institute of Technology / Lincoln Laboratory, Cambridge, MA, 1985.

[MQ86] R.J. McAulay and T.F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-34 No.4:744–754, 1986.

[MSD97] A.W. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. 1997.

[Nea96] Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer, New York, 1996.

[NH93] Radford M. Neal and Geoffrey E. Hinton. A new view of the em algorithm that justifies incremental and other variants, 1993.

[OHR91] R. Orton, A. Hunt, and R.Kirk. Graphical control of granular synthesis using cellular automata and the freehand program. In *Proceedings International Computer Music Conference*, pages 416–418, San Franzisco, 1991.

[Oja83] E. Oja. *Subspace Methods of Pattern Recognition.* Research Studies Press, New York, 1983.

[Oli00] Nuria M. Oliver. *Towards Perceptual Intelligence: Statistical Modeling of Human Individual and Interactive Behaviors.* PhD thesis, MIT Media Lab, 2000.

[OPH+00] O. Omojola, R. Post, M. Hancher, Y. Maguire, R. Pappu, B. Schoner, P. Russo, R. Fletcher, and N. Gershenfeld. An installation of interactive furniture. *IBM Systems Journal*, 2000. To appear.

[OS89] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[OW83] A.V. Oppenheim and A.S. Willsky. *Signals and Systems.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[Pea87] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufman, 1987.

[Pet76] T.L. Petersen. Analysis-Synthesis as a Tool for Creating New Families of Sound. In *Proc. of the 54th. Conv. Audio Engineering Society*, Los Angeles, 1976.

[PG97] Joseph A. Paradiso and Neil Gershenfeld. Musical applications of electric field sensing. *Computer Music Journal*, 21(2):69–89, 1997.

[Pop97] K. Popat. *Conjoint Probabilistic Subband Modeling.* PhD thesis, MIT Media Lab, 1997.

[PP93] K. Popat and R.W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. *Proceedings of the SPIE*, 2094, pt.2:756–68, 1993. Visual Communications and Image Processing '93.

[PR99] Geoffrey Peeters and Xavier Rodet. SINOLA: A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum. In *Proc. ICMCs*, Beijing, 1999.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 2nd edition, 1992.

[PYP+96] D. Poeppel, E. Yellin, C. Phillips, T.P.L. Roberts, H.A. Rowley, K. Wexler, and A. Marantz. Task-induced asymmetry of the auditory evoked m100 neuromagnetic field elicited by speech sounds. *Cognitive Brain Research*, 4:231–242, 1996.

[QM98] T.F. Quatieri and R.J. McAulay. Audio signal processing based on sinusoidal analysis/synthesis. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*. Kluwer Academic Publishers, Boston, 1998.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[RD82] J.-C. Risset and Wessel D.L. Exploration of timbre analysis and synthesis. In D. Deutsch, editor, *The Psychology of Music*, pages 26–58. New York: Academic, 1982.

[Ris69] J.-C. Risset. *Catalog of computer-synthesized sound*. Bell Telephone Laboratories, Murray Hill, 1969.

[RJ86] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–16, 1 1986.

[Roa91] C. Roads. Asynchronous granular synthesis. In G. DePoli, A. Piccialli, and C. Roads, editors, *Representations of Musical Signals*, pages 143–185. MIT Press, 1991.

[Roa95] Curtis Roads. *The computer music tutorial*. MIT Press, 1995.

[RSR+00] M. Reynolds, B. Schoner, J. Richards, K. Dobson, and N. Gershenfeld. A polyphonic floor, 2000. preprint.

[Sai94] Naoki Saito. *Local Feature Extraction and Its Applications Using a Library of Basis*. PhD thesis, Yale University, 1994.

[SBDH97] J. Stark, D.S. Broomhead, M.E. Davies, and J. Huke. Taken's embedding theorem for forces and stochastic systems. *Nonlinear Anal.*, 30:5303–5314, 1997.

[SCDG99] B. Schoner, C. Cooper, C. Douglas, and N. Gershenfeld. Data-driven modeling of acoustical instruments. *Journal for New Music Research*, 28(2):417–466, 1999.

[Sch73] J.C. Schelleng. The bowed string and the player. *J. Acoust. Soc. America*, 53:26–41, 1973.

[Sch77] Pierre Schaeffer. *Traité des objects musicaux.* Edition du Seuil, Paris, 1977.

[Sch89] M. Schetzen. *The Volterra and Wiener Theories of Nonlinear Systems.* John Wiley and Sons, Inc., New York, 1989.

[Sch96] Bernd Schoner. State reconstructiion for determining predictability in driven nonlinear acoustical systems. Master's thesis, MIT/RWTH Aachen, 1996.

[Sch00] Eric Scheirer. *Music Listening Systems.* PhD thesis, MIT Media Lab, 2000.

[Ser89] Xavier Serra. *A system for Sound Analysis/Transformation/Synthesis Based on a Deterministic Plus Stochastic Decomposition.* PhD thesis, CCRMA, Department of Music, Stanford University, 1989.

[SG00] Bernd Schoner and Neil Gershenfeld. Cluster-weighted modeling: Probabilistic time series prediction, characterization and synthesis. In Alistair Mees, editor, *Nonlinear Dynamics and Statistics.* Birkhaeuser, Boston, 2000.

[SJ96] L. Saul and M.I. Jordan. Exploiting tractable substructures in intractable networks. *Advances in Neural Information Processing Systems*, 8:486–492, 1996.

[SJJ96] L. Saul, T. Jaakkola, and M.I. Jordan. Meanfield theory for sigmoid belief networks. *Journal of Artificial Intelligence*, 4:61–76, 1996.

[Sma97] Paris Smaragdis. Information theoretic approaches to source separation. Master's thesis, MIT Media Lab, 1997.

[Smi92] Julius O. Smith. Physical modeling using digital waveguides. *Computer Music Journal*, 6(4), 1992.

[Smi98] Julius O. Smith. Principles of digital waveguide models of musical instruments. In Marc Kahrs and Karlheinz Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 417–466. Kluwer Academic Publishers, 1998.

[SP84] J.O. Smith and P.Gosset. A flexible sampling-rate conversion method. *Acoustics, Speech, and Signal Processing*, 2:19.4.1–19.4.2, 1984.

[SP98] Joshua Strickon and Joseph Paradiso. Tracking hands above large interactive surfaces with a low-cost scanning laser rangefinder. In *CHI98, Extended Abstracts*, pages 231–232, New York, 1998. ACM Press.

[SS87] J.O. Smith and X. Serra. An analysis/synthesis program for non-harmonic sounds based on sinusoidal representation. In *Proceedings International Computer Music Conference*, pages 290–297, San Francisco, 1987.

[SS90] Xavier Serra and Julius O. Smith. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

[STH99] R. Sullivan, A. Timmermann, and H.White. Data snooping, technical trading rule performance, and the bootstrap. *Journal of Finance*, 54:1647–1692, 1999.

[Tak81] Floris Takens. Detecting strange attractors in turbulence. In D.A. Rand and L.S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381, New York, 1981. Springer-Verlag.

[TC99] Dan Trueman and Perry R. Cook. BoSSA: The Deconstructed Violin Reconstructed. In *Proceedings International Computer Music Conference*, Beijing, 1999.

[TL91] Peter M. Todd and D. Gareth Loy, editors. *Music and Connectionism.* MIT Press, 1991.

[Tur36] A.M. Turing. *Proc. London Math. Soc.*, 42:1134–1142, 1936.

[VB94a] F. Verbeyst and M. Vanden Bossche. Viomap, the s-parameter equivalent for weakly nonlinear rf and microwave devices. In *Microwave Symposium Digest of IEEE 1994 MTT-S International*, pages 1369–1372, New York, 1994. IEEE.

[VB94b] F. Verbeyst and M. Vanden Bossche. The volterra input-output map of a high frequency amplifier as a practical alternative to load-pull measurements. In *Conference Proceedings TMTC'94*, pages 81–85, New York, 1994. IEEE.

[VMT91] H. Valbret, E. Moulines, and J. Tubach. Voice Transformation Using PSOLA Techique. In *Proceedings of Eurospeech 91(1)*, pages 345–348, 1991.

[Wat95] John Watkinson. *Compression in video and audio.* Focal Press, Oxford, 1995.

[WB98] A. Wilson and A. Bobick. Nonlinear phmms for the interpretation of parameterized gesture. *Computer Vision and Pattern Recognition*, 1998.

[WB99] A. Wilson and A. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9), 1999.

[WDW98] D. Wessel, C. Drame, and M. Wright. Removing the time axis from spectral model analysis-based additive synthesis: Neural networks versus memory-based machine learning. In *Proceedings International Computer Music Conference*, pages 62–65, Ann Arbor, Michigan, 1998.

[WEB] `http://www.media.mit.edu/~schoner/phd/` .

.

[Wes79] David L. Wessel. Timbre space as a musical control structure. *Computer Music Journal*, 3(2):45–52, 1979. republished in *Foundations of Computer Music*, Curtis Roads (Ed., MIT Press).

[WG93] Andreas S. Weigend and Neil A. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Santa Fe Institute Studies in the Sciences of Complexity. Addison–Wesley, Reading, MA, 1993.

[Whi00] H. White. A reality check for data snooping. *Econometrica*, 2000. forthcoming.

[WMS95] A.S. Weigend, M. Mangeas, and A.N. Srivastava. Nonlinear gated experts for time series: discovering regimes and avoiding overfitting. *International Journal of Neural Systems*, 6:373–99, 1995.

[Woo92] James Woodhouse. Physical modelling of bowed strings. *Computer Music Journal*, 16(4):43–56, 1992.

[WWS96] A. Willsky, G. Wornell, and J. Shapiro. *Stochastic Processes, Detection and Estimation*. EECS/MIT, Cambridge, 1996. class-notes.