Hidden Markov Models *and* Dynamical Systems

Hidden Markov Models *and* Dynamical Systems

Andrew M. Fraser

Los Alamos National Laboratories
Los Alamos, New Mexico

siam®

Society for Industrial and Applied Mathematics • Philadelphia

10 9 8 7 6 5 4 3 2 1

This work was supported by a sabbatical leave and the use of a computer from Portland State University and by the Los Alamos National Laboratory LDRD Office through project 20030037DR.

Library of Congress Cataloging-in-Publication Data

Fraser, Andrew M.
Hidden Markov models and dynamical systems / Andrew M. Fraser.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-898716-65-8
1. Markov processes. 2. Dynamics. I. Title.
QA274.7.F726 2008
519.2'33–dc22

2008028742

Partial royalties from the sale of this book are placed in a fund to help students attend SIAM meetings and other SIAM-related activities. This fund is administered by SIAM and qualified individuals are encouraged to write directly to SIAM for guidelines.

For Marguerite

Contents

Preface **xi**
Acknowledgments xii

1 Introduction **1**
1.1 Laser Example 2
1.2 State Space Models 4
1.2.1 Tasks 6
1.3 Discrete HMMs 7
1.3.1 Example: Quantized Lorenz Time Series 11
1.3.2 Example: Hidden States as Parts of Speech 13
1.3.3 Remarks 15

2 Basic Algorithms **19**
2.1 The Forward Algorithm 20
2.1.1 Vector Notation 23
2.1.2 General Filtering 24
2.2 The Backward Algorithm 25
2.3 The Viterbi Algorithm 27
2.3.1 General Decoding 29
2.3.2 MAP Sequence of States or Sequence of MAP States? 29
2.4 The Baum–Welch Algorithm 31
2.4.1 Weights and Reestimation 32
2.4.2 Remarks 37
2.5 The EM Algorithm 38
2.5.1 Monotonicity 42
2.5.2 Convergence 43

3 Variants and Generalizations **47**
3.1 Gaussian Observations 48
3.1.1 Independent Scalar Observations 48
3.1.2 Singularities of the Likelihood Function and Regularization 50
3.1.3 The EM Algorithm for MAP Estimation 51
3.1.4 Vector Autoregressive Observations 52
3.2 Related Models 55

4 Continuous States and Observations and Kalman Filtering 59
4.1 Algorithms with Integrals . . . 60
4.1.1 Forward Algorithm . . . 60
4.1.2 Backward Algorithm . . . 61
4.2 Linear Gaussian Systems . . . 62
4.2.1 Kalman Filter: The Forward Algorithm . . . 64
4.2.2 The Backward Algorithm . . . 65
4.2.3 Smoothing . . . 65
4.3 Algorithm Derivations and Details . . . 66
4.3.1 Forward Kalman Filter . . . 66
4.3.2 Backward Recursion . . . 69
4.3.3 Smoothing . . . 71
4.3.4 Inverse Covariance Form . . . 71
4.3.5 Extended Kalman Filter . . . 71

5 Performance Bounds and a Toy Problem 73
5.1 Fidelity Criteria and Entropy . . . 78
5.1.1 Definitions . . . 78
5.2 Stretching and Entropy . . . 82
5.2.1 Maps of the Unit Circle . . . 82
5.3 Lyapunov Exponents and Pesin's Formula . . . 84
5.3.1 A Theoretical Bound on Model Likelihood . . . 85
5.4 Benettin's Procedure for Calculating Lyapunov Exponents Numerically . 87
5.5 A Practical Performance Bound . . . 89
5.6 Approaching the Bound . . . 94

6 Obstructive Sleep Apnea 97
6.1 The Challenge and the Data . . . 97
6.1.1 The Data . . . 99
6.2 First Classification Algorithms and Two Useful Features . . . 101
6.2.1 Using Information from Experts to Train . . . 101
6.2.2 Nonlinear Dynamics . . . 102
6.2.3 The Excellent Eye of Dr. McNames . . . 103
6.3 Decoding Sequences of Classifications . . . 104
6.4 Assembling the Pieces . . . 106
6.4.1 Extracting a Low Pass Filtered Heart Rate . . . 107
6.4.2 Extracting Respiration Information . . . 108
6.4.3 Classifying Records . . . 110
6.4.4 Model Topology and Training Data . . . 112
6.4.5 Tunable Parameters . . . 112
6.4.6 Results . . . 114
6.5 Classification Versus Estimation . . . 116

A Formulas for Matrices and Gaussians 117

B Notes on Software 121

Bibliography **125**
Books and Collections 125
Review Articles 126
Dissertations and Theses 127
Research Articles 127
Web Sites 129

Index **131**

Preface

This book arose from a pair of symposia on hidden Markov models (HMMs) that Kevin Vixie organized at the 2001 SIAM Conference on Applications of Dynamical Systems. At the end of the first symposium someone asked the speakers for a simple reference that explains the basic ideas and algorithms for applying HMMs. We were stumped. A group of the participants suggested writing this book to answer the question. I have aimed the book at readers who have backgrounds and interests typical of those who attended that conference. In my view, HMMs are discrete-state, discrete-time stochastic dynamical systems, and they are often used to approximate dynamical systems with continuous state spaces operating in continuous time. Thus, by using familiar analogies, it is easy to explain HMMs to readers who have studied dynamical systems.

The basic techniques were well developed in the 1970's for work on speech and language processing. Many in speech research learned about the techniques at a symposium held at the Institute for Defense Analysis in Princeton, NJ. J. D. Ferguson edited the proceedings [4], and copies were given to the attendees.[1] The volume was called *the blue book* by workers in the field. I was not part of that community, but I have a copy of the blue book. It explains the basic algorithms and illustrates them with simple concrete examples. I hope *this* book is as simple, useful, and clear.

Although there are other books and papers that are about HMMs exclusively or in part, I hope that readers find the following features of this present volume useful:

It is introductory. An undergraduate background in engineering, mathematics, or science that includes work in probability, linear algebra, and differential equations provides the prerequisites for most of the book. The exceptions are ideas from dynamical systems and information theory. In particular, I use the Gibbs inequality (see (2.53)) in developing the estimate maximize (EM) algorithm in Chapter 2. Although Chapter 5 deals with Lyapunov exponents and entropies, it is not a prerequisite for any other chapter.

Algorithms are explained and justified. I present enough of the theory behind the basic algorithms in Chapter 2 so that a reader can use it as a guide to developing his own variants.

I provide code implementing the algorithms and data for the examples. Although algorithms are given in pseudocode in the text, a working implementation of each of the

[1] The volume is available at a few libraries.

algorithms that I describe is available on the Web [56]. I have chosen to write the programs in the Python language [59] because it is easy to read and the interpreter is free software. I have written the programs to follow the descriptions and notation in the text. I provide data and scripts (makefiles, shell scripts, etc.) that make all of the figures in the text. On a GNU system, issuing "make book.pdf" from a command line compiles the software, runs the numerical experiments, makes the figures, and formats the entire book.

It uses analogies to dynamical systems. For example, I demonstrate the HMM training algorithm by applying it to data derived from the Lorenz system. The result, as Fig. 1.9 illustrates, is that the algorithm estimates a discrete-state generating mechanism that is an approximation to the state space of the original Lorenz system.

I illustrate with a practical example. In Chapter 6, I present an application to experimental measurements of electrocardiograms (ECGs).

Acknowledgments

In writing this book, I have used much that I learned from colleagues in Portland, OR. In particular, Todd Leen kept me informed about developments in the machine learning community. I have relied on the review of the ergodic theory of dynamical systems in Kevin Vixie's dissertation [27] for Chapter 5 of this book. Shari Matzner and Gerardo Lafferriere helped me understand the convergence properties of the EM algorithm. Also the following colleagues have given me constructive comments on drafts of the book: Katherine Backus, Patrick Campbell, Ralf Juengling, Shari Matzner, Kary Myers, Reid Porter, Cosma Shalizi, and Rudolph Van der Meer.

I thank the following people and organizations for providing the data that I used for examples:

Carl Otto Weiss for providing Tang's laser data [51],

PhysioNet for providing Penzel's ECG data [20] and George Moody's *WFDB* software [58],

Project Gutenberg for digitizing and distributing *A Book of Prefaces*, by H. L. Mencken.

I was fortunate to meet Karl Hegbloom in Portland. Karl contributes to several free software projects, and he helped me with the figures and software. Ralf Juengling also helped with the software. In addition to personal help with software, I relied on free software written by too many contributors to mention. Almost all of the tools I used are available in the Debian [55] and Ubuntu [61] distributions.

I acknowledge Portland State University for support of a sabbatical leave during which I started the book and for a computer on which I did much of the development. I thank Principal Investigator James Kamm for arranging to have the Los Alamos National Laboratory LDRD Office support work on Chapter 6 through project 20030037DR.

Chapter 1
Introduction

In a dynamical system, a rule maps states $x \in \mathcal{X}$ forward in time. Familiar examples include discrete-time maps and differential equations in which the states are elements of $\mathbb{R}^n$. At time t, the current state $x(t)$ of a dynamical system provides all the information necessary to calculate future states, and information about past states is redundant. Thus the sequence of states in a dynamical system satisfies the *Markov property*, which I will more formally define in (1.7). In applications, I often think of measured data as being a function of states of a dynamical system with $y(t) = G(x(t))$. The function F that maps states forward in time and the function G that maps states to observations make up a *state space model* for sequences of observations. If the observation function is not invertible, then knowledge of the measurement $y(t)$ at a single time t is not sufficient to specify the state $x(t)$ uniquely, and one could say that $x(t)$ is *hidden*. That is the sense of *hidden* in the term "hidden Markov model."

Given a short sequence of observations, say $(y(1), y(2), y(3))$, one might hope to *reveal* the state $x(3)$ by looking for all initial states that are consistent with the observations. The strategy will work only in special cases. Let Ψ be the set of initial states that are consistent with the all of observations, i.e., for every $x \in \Psi$, $G(x) = y(1)$, $G \circ F(x) = y(2)$, and $G \circ F \circ F(x) = y(3)$. If I find that for all $x \in \Psi$, $F \circ F(x) = \hat{x}$, then the measurements are sufficient to identify $x(3) = \hat{x}$ uniquely. If such a revelation procedure works, then one can use it to map long sequences of observations to long sequences of states and from there to do forecasting both of states and observations.

For most of the state space models that I consider, both the function that governs the state dynamics and the observation function have random elements. Only imagination limits what constitutes the set of states in a state space model. I will consider discrete-state spaces that are sets with a finite number of elements and state spaces that are real vector spaces. The sets of observations are similarly varied. As a prelude, I look at some measurements of a laser system that is "Lorenz like."

1.1 Laser Example

In 1963 Lorenz [42] reported interesting behavior in numerical solutions of the system of equations

$$\dot{x}_1 = sx_2 - sx_1, \tag{1.1a}$$
$$\dot{x}_2 = -x_1x_3 + rx_1 - x_2, \tag{1.1b}$$
$$\dot{x}_3 = x_1x_2 - bx_3, \tag{1.1c}$$

which he had proposed as an approximation for fluid convection. In (1.1), $x = (x_1, x_2, x_3)$ is a vector of mode amplitudes, and the parameters s, r, and b describe properties of the fluid. The paper is widely cited, not because it is a good model for convection but because the interesting behavior of the solutions has characteristics that are typical of what is now called *chaos*. The Lorenz system has been used countless times as an example of a system whose solution trajectories are unstable, aperiodic, and bounded, i.e., *chaotic*. I will use numerical simulations of the system as illustrations throughout this book.

In 1975 Haken [37] observed that under certain conditions a laser should obey the same equations. For a laser, one interprets the components of the state x as the electric field, the polarization, and the population inversion in the laser medium. In 1992 Tang et al. [51] reported measurements of the time dependent intensity of the electric field for such a laser. The measured quantity corresponds to $(x_1(t))^2$. Fig. 1.1 shows a sequence of Tang's measurements. I produced the second trace in the figure by numerically integrating (1.1) with initial conditions and parameters selected to optimize the match. I used the *fmin_powell* function in Travis Oliphant's *SciPy* package [60] to implement the optimization. The similarity of the two traces convincingly supports the claim that the laser system is governed by the Lorenz equations.

In working with Tang's laser data, I used a stochastic state space model with the form

$$x(t+1) = F(x(t)) + \eta(t), \tag{1.2a}$$
$$y(t) = G(x(t)) + \epsilon(t). \tag{1.2b}$$

I implemented the function F by integrating the Lorenz system for an interval τ_s, and I used independently and identically distributed (i.i.d.) Gaussian noise with mean zero and covariance $\mathbf{I}\sigma_\eta^2$ to implement the state noise $\eta(t)$. The measurement model is $G(x(t)) = S_g \cdot (x_1(t))^2 + O_g$, where S_g and O_g are scale and offset parameters, and the measurement noise $\epsilon(t)$ is i.i.d. Gaussian noise with mean zero and covariance σ_ϵ^2. The model has the following 11 free parameters:

Initial state distribution. I model the distribution of the initial state as a Gaussian. I use three free scalar parameters to specify the mean and set the covariance to $\mathbf{I} \cdot 10^{-2}$.

Lorenz system parameters. The values of r, s, and b in (1.1) constitute three free parameters.

Integration time. The single parameter τ_s. This is the evolution time in the Lorenz system used to model the time between consecutive samples of the laser system.

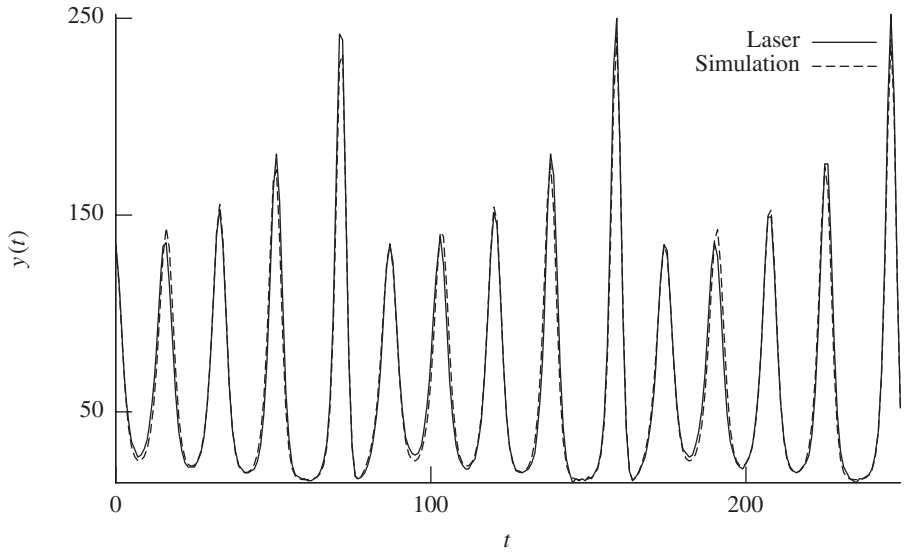


Figure 1.1. *Laser intensity measurements. The trace labeled* Laser *is a plot of laser intensity measurements provided by Tang et al. The trace labeled* Simulation *plots a numerical simulation of the Lorenz system* (1.1) *with parameters* $r = 21.16$, $s = 1.792$, *and* $b = 0.3670$ *and measurement parameters* $\tau_s = 0.1435$, $S_g = 7.071$, *and* $O_g = 15.16$. *I used the optimization procedure described in the text to select these parameters. The simulated intensities were derived from the state by* $y(t) = S_g \cdot (x_1(t))^2 + O_g$. *I specified an absolute error tolerance of* 10^{-7} *per time step for the numerical integrator.*

Offset and scale. The pair of parameters O_g and S_g.

State noise. The single parameter σ_η.

Measurement noise. The single parameter σ_ϵ.

Using this set of parameters, I wrote a routine based on the *extended Kalman filter* techniques described in Chapter 4 to calculate approximate probabilities which I write as $P_{*|\theta}$, where θ denotes the collection of parameters. By passing that routine and Tang's data to the *SciPy* optimization package, I found a vector of parameters that satisfies[2]

$$\hat{\theta} = \operatorname*{argmax}_{\theta} \; P(y_1^{250}|\theta), \tag{1.3}$$

where $P(y_1^{250}|\theta)$ is the conditional probability that a sequence of 250 observations will have the values $y(1), y(2), \ldots, y(250)$ given the parameters θ. The vector $\hat{\theta}$ is called the

[2]Here $\operatorname{argmax}_\theta$ means the value of θ that maximizes what comes next. For example,

$$\max_{\theta} -(1-\theta)^2 = 0$$

and

$$\operatorname*{argmax}_{\theta} -(1-\theta)^2 = 1.$$

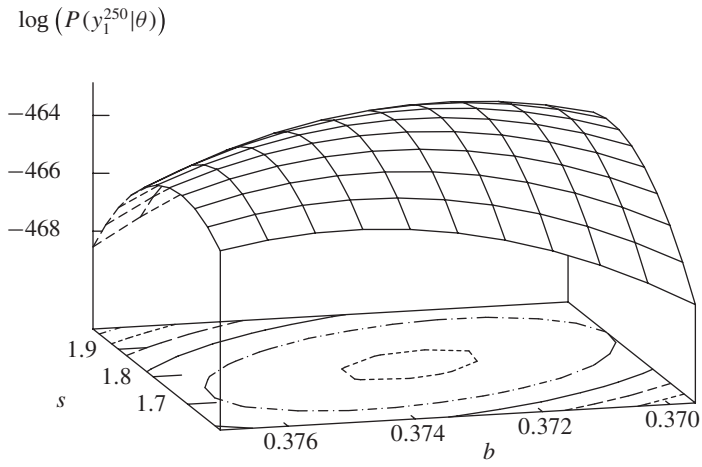


Figure 1.2. *Log likelihood as function of s and b. Other parameters were taken from the vector $\hat{\theta}$ that maximizes the likelihood $P(y_1^{250}|\theta)$ (see* (1.3)*).*

maximum likelihood estimate of the parameter vector. Fig. 1.2 sketches a piece of the log-likelihood function.

Given the maximum likelihood parameters, $\hat{\theta}$, and the observations, I can calculate many interesting quantities. For example, in Fig. 1.3, I have plotted the sequence of states that has the highest probability, i.e.,

$$\hat{x}_1^{250} = \underset{x_1^{250}}{\operatorname{argmax}}\, P(x_1^{250}|y_1^{250}, \hat{\theta}), \tag{1.4}$$

and, in Fig. 1.4, I have plotted a forecast that I made by iterating the function F on the state $\hat{x}$ that has highest probability given the first 250 observations, i.e.,

$$\hat{x} = \underset{x(250)}{\operatorname{argmax}}\, P(x(250)|y_1^{250}, \hat{\theta}). \tag{1.5}$$

1.2 State Space Models

To get state space models that are more general than the form (see (1.2)) that I used to describe the laser data, I suppose only that a conditional probability distribution $P_{X(t+1)|X(t)}$ governs evolution in state space and another conditional distribution $P_{Y(t)|X(t)}$ governs the observations $Y(t)$. Combining these two conditional distribution functions with a distribution $P_{X(1)}$ of initial states defines probabilities for any collection of states and observations. In particular, it defines the joint probabilities of the *stochastic process* or *information source* consisting of sequences of observations. I refer to such a combination as a *model* and denote it as $P_{*|\theta}$. So defined, the class of state space models is so broad that, to do anything useful, I must use smaller subclasses. Typically, I assume that the conditional distributions are time invariant and that a finite set of parameters θ specifies the model. Notice that I have not

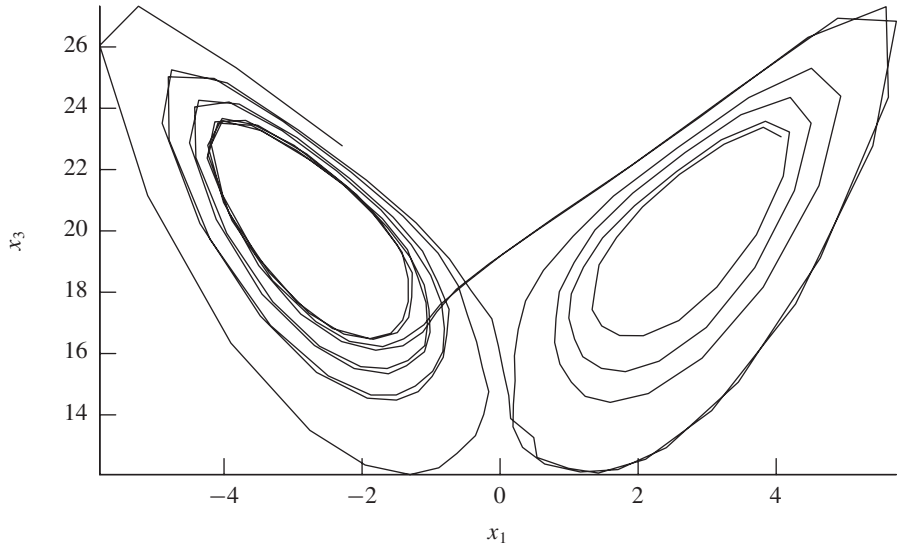


Figure 1.3. *State trajectory* $\hat{x}_1^{250}$ *estimated from observation sequence* y_1^{250} *(see* (1.4)*). Components* x_1 *and* x_3 *of the Lorenz system (see* (1.1)*) are plotted. Recovering the familiar Lorenz figure suggests both that the laser data is* Lorenz like *and that the algorithm for estimating states from observations is reasonable.*

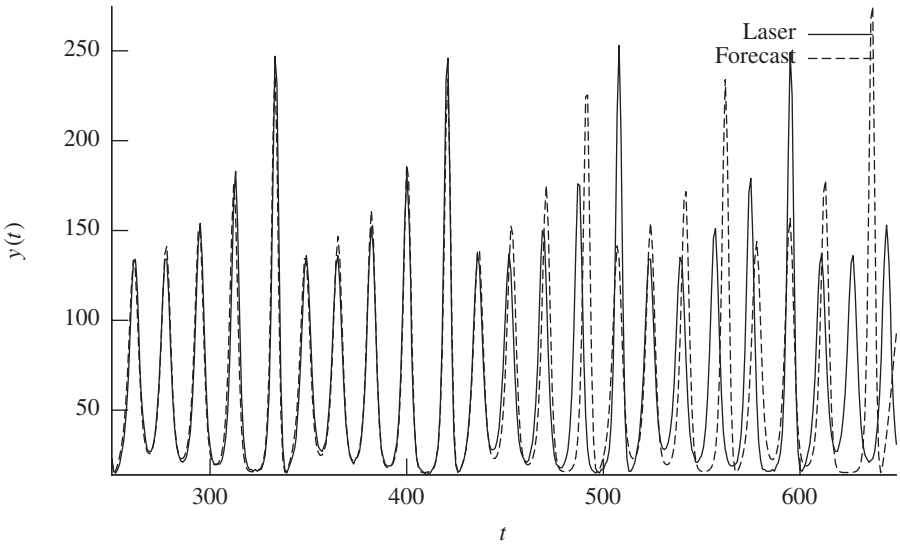


Figure 1.4. *Forecast observation sequence. I set the noise terms* η *and* ϵ *to zero and iterated* (1.2) 400 *times to generate the forecast* $\hat{y}_{251}^{650}$*. I started with the initial condition* $\hat{x}$ *defined by* (1.5)*. The forecast begins to fail noticeably after* $t = 500$*. The failure suggests that the period five cycle in the forecast is unstable. The laser cycle must have been stable to appear in the data. Thus an essential characteristic of the model is wrong.*

specified the sets from which I draw the states or observations; they could be discrete, real scalars, real vectors, or something else.

1.2.1 Tasks

One can use a class of state space models with parameters $\left\{P_{*|\theta}\right\}$ for many tasks including the following:

Model parameter estimation. Given a model class $\left\{P_{*|\theta}\right\}$ and a sequence of observations y_1^T, one often uses the maximum likelihood estimate

$$\hat{\theta}_{MLE} \equiv \underset{\theta}{\operatorname{argmax}}\, P(y_1^T|\theta) \tag{1.6}$$

to characterize the source Y.

Trajectory estimation. Given a particular model $P_{*|\theta}$ and a sequence of observations y_1^T, one can calculate the conditional distribution of states $P(x_1^T|y_1^T,\theta)$. For example, Fig. 1.3 plots the result of a calculation of

$$\hat{x}_1^{250} = \underset{x_1^{250}\in\mathcal{X}^{250}}{\operatorname{argmax}}\, P(x_1^{250}|y_1^{250},\theta).$$

Short term forecasting. Given a model $P_{*|\theta}$ and a distribution of states, $P_{X(t)}$, at time t, one can calculate the conditional distribution of future states or observations. For example, Fig. 1.4 plots the result of a calculation of

$$\hat{y}_{251}^{500} = \underset{y_{251}^{500}}{\operatorname{argmax}}\, P(y_{251}^{500}|y_1^{250}, x(250),\theta)P_{x(250)}.$$

Simulation. Given a model $P_{*|\theta}$, one can characterize its long term behavior, answering questions such as, "What is a hundred year flood?" I often find that models that I fit are not good for such long term extrapolation. For example, the laser data that I described in the previous section seems to come from a stable period five orbit, but the periodic orbit that the trajectory in Fig. 1.3 approximates is linearly unstable. Thus the long term behavior of my estimated model is very different from the actual laser system.

Classification. Given sample signals such as y_1^T and two possible signal sources, α and β, where $P_{*|\alpha}$ characterizes healthy units and $P_{*|\beta}$ characterizes defective units, one can classify a unit on the basis of the *likelihood ratio*

$$R(y_1^T) = \frac{P(y_1^T|\beta)}{P(y_1^T|\alpha)}.$$

If $R(y_1^T)$ is above some threshold, y_1^T is classified as defective.

1.3 Discrete HMMs

In this section, I describe the simplest HMMs: those that are discrete in time, state, and observation. I begin with a couple of definitions. Three random variables $X(1)$, $X(2)$, and $X(3)$ constitute a *Markov chain* if

$$P_{X(3)|X(1),X(2)} = P_{X(3)|X(2)}, \tag{1.7}$$

which is equivalent to $X(1)$ and $X(3)$ being conditionally independent given $X(2)$, i.e.,

$$P_{X(3),X(1)|X(2)} = P_{X(3)|X(2)} P_{X(1)|X(2)}.$$

An indexed sequence of random variables X_1^T is a *Markov process* if for any $t : 1 < t < T$ the variables before and after t are conditionally independent given $X(t)$, i.e.,

$$P_{X_1^{t-1},X_{t+1}^T|X(t)} = P_{X_1^{t-1}|X(t)} P_{X_{t+1}^T|X(t)}. \tag{1.8}$$

I will restrict my attention to time invariant models, i.e., those for which the transition probabilities are constant over time. Begin by considering the ordinary (*unhidden*) Markov model or process sketched in Fig. 1.5. The set of states $\mathcal{S} = \{u, v, w\}$, the probability distribution for the initial state

$$P_{S(1)} = \left[\tfrac{1}{3}, \quad \tfrac{1}{3}, \quad \tfrac{1}{3}\right], \tag{1.9}$$

and the transition matrix

		$S(t+1)$		
$P(s(t+1)\|s(t))$		u	v	w
$S(t)$	u	0	1	0
	v	0	$\frac{1}{2}$	$\frac{1}{2}$
	w	$\frac{1}{2}$	$\frac{1}{2}$	0

(1.10)

define the model, and the model determines the probability of any sequence of states s_1^T, which I write[3] as $P_{S_1^T}\left(s_1^T\right)$. For example, I calculate the probability that a sequence of four states has the values $s_1^4 = (u, v, w, v)$ (i.e., $s(1) = u$, $s(2) = v$, $s(3) = w$, and $s(4) = v$)

[3] I use uppercase letters to denote random variables and P to denote probability distribution functions. A random variable used as a subscript on P specifies that I mean the distribution of that random variable. I can give P an argument to specify the value of the distribution function at that value; e.g., $P_X(3)$ is the probability that the random variable X has the value 3, and $P_X(x)$ is the probability that the random variable X has the value x. I usually drop subscripts on P when the context or argument resolves ambiguity as to which probability function I mean.

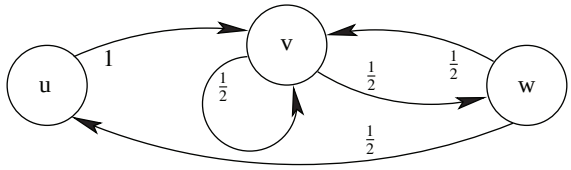


Figure 1.5. *A Markov model.*

as follows:

$$P(s_1^4) = P(s(1)) \prod_{\tau=2}^{4} P(s(\tau)|s_1^{\tau-1}) \tag{1.11}$$

$$= P(s(1)) \prod_{\tau=2}^{4} P(s(\tau)|s(\tau-1)), \tag{1.12}$$

$$P(u, v, w, v) = P(v|u, v, w) \cdot P(w|u, v) \cdot P(v|u) \cdot P(u) \tag{1.13}$$

$$= P(v|w) \cdot P(w|v) \cdot P(v|u) \cdot P(u) \tag{1.14}$$

$$= \frac{1}{2} \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{3} = \frac{1}{12}. \tag{1.15}$$

Applying Bayes' rule $\left(P_{A|B} P_B = P_{A,B}\right)$ recursively yields (1.11) and the special case, (1.13). Equations (1.12) and (1.14) follow from (1.11) and (1.13), respectively, by the *Markov assumption*, (1.8), which says that in determining the probability of the tth state given any sequence of previous states only the $(t-1)$th state is relevant.

A common exercise is to find a *stationary* probability distribution, i.e., given a transition matrix T find the probability vector V (nonnegative entries that sum to one) that satisfies

$$VT = V. \tag{1.16}$$

If (1.16) holds, then

$$P_{S(2)} = P_{S(1)} T = P_{S(1)} = P_{S(t)} \quad \forall t,$$

and in fact all probabilities are independent of shifts in time, i.e.,

$$P_{S_1^t} = P_{S_{1+\tau}^{t+\tau}} \quad \forall (t, \tau),$$

which is the definition of a stationary process. Quick calculations verify that the initial probability and transition matrix in (1.9) and (1.10) do not satisfy (1.16) but that the distribution $V = \left[\frac{1}{7}, \quad \frac{4}{7}, \quad \frac{2}{7}\right]$ does. Although the example is not a stationary stochastic process, it relaxes towards such a process in the sense that

$$\lim_{t\to\infty} P_{S(t)} = \lim_{t\to\infty} P_{S(1)} T^t = \left[\frac{1}{7}, \quad \frac{4}{7}, \quad \frac{2}{7}\right].$$

The important points about a Markov model also apply to HMMs, namely, that

- the model determines the probability of arbitrary sequences of observations,

- and the assumptions about independence and time invariance permit specification of the model by a small number of parameters.

Now suppose, as sketched in Fig. 1.6, that when the system arrives in a state that rather than observing the state directly one observes a random variable that depends on the state. The matrix that specifies the random map from states to observations is the following:

		Y		
$P(y\|s)$		d	e	f
	u	1	0	0
S	v	0	$\frac{1}{3}$	$\frac{2}{3}$
	w	0	$\frac{2}{3}$	$\frac{1}{3}$

Combined with the distribution of initial states (1.9) and transition matrix (1.10) it specifies this HMM. The notion is that the underlying Markov process chugs along unaware of the observations and that, when the process arrives at each successive state $s(t)$, an observation $y(t)$ is produced in a fashion that depends only on the state $s(t)$.

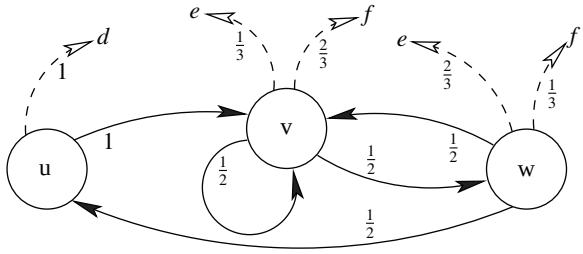


Figure 1.6. *An HMM.*

Now calculate the probability that a sequence of four observations from this process would have the values $y_1^4 = (d, e, f, e)$. As an intermediate step, calculate $P(y_1^4, s_1^4)$ for the given observation sequence and all possible state sequences. Then add to obtain

$$\sum_{s_1^4} P(y_1^4, s_1^4) = \sum_{s_1^4} P(y_1^4|s_1^4) P(s_1^4) = P(y_1^4). \tag{1.17}$$

Conveniently the only state sequences that could have produced the observation sequence are (u, v, v, v), (u, v, v, w), and (u, v, w, v). For any other state sequence $P(y_1^4, s_1^4) = 0$. The arithmetic details appear in Table 1.1.

Now examine this calculation more carefully beginning with a statement of the model assumptions.

The observations are conditionally independent given the states: Given the current state, the probability of the current observation is independent of states and observations at all earlier times, i.e.,

$$P_{Y(t)|S_1^t, Y_1^{t-1}} = P_{Y(t)|S(t)}. \tag{1.18}$$

Table 1.1. *A calculation of the probability that the first four observations produced by the model in Fig.* 1.6 *would be a particular sequence. Adding the fractions in the right-hand column yields the result:* $P(d, e, f, e) = \frac{7}{324}$.

s_1^4	$P(s_1^4)$	$P(y_1^4 \mid s_1^4)$	$P(y_1^4, s_1^4)$
uvvv	$\frac{1}{3} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2}$	$1 \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}$	$\frac{2}{324}$
uvvw	$\frac{1}{3} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2}$	$1 \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}$	$\frac{4}{324}$
uvwv	$\frac{1}{3} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2}$	$1 \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3}$	$\frac{1}{324}$

The state process is Markov: Given the current state, the probability of the next state is independent of earlier states. Combined with the first assumption, this implies that the next state is also conditionally independent of past observations, i.e.,

$$P_{S(t+1)|S_1^t, Y_1^t} = P_{S(t+1)|S(t)}. \tag{1.19}$$

Using Bayes' rule and these assumptions one can justify

$$P(y_1^T, s_1^T) = P(s_1^T)\, P(y_1^T \mid s_1^T),$$

$$P(s_1^T) = P(S(1)) \prod_{t=2}^{T} P(s(t)|s(t-1)), \tag{1.20}$$

$$P(y_1^T \mid s_1^T) = \prod_{t=1}^{T} P(y(t)|s(t)) \tag{1.21}$$

and conclude that

$$P(y_1^T, s_1^T) = P(s(1)) \prod_{t=2}^{T} P(s(t)|s(t-1)) \prod_{t=1}^{T} P(y(t)|s(t)).$$

The fact that the state u produces the observation d exclusively and no other state can produce d means that to produce the observation sequence (d, e, f, e) a state sequence must begin with u and not return to u. That constraint reduces the number of possible state sequences to eight. The impossibility of state w following itself further constrains the possible state sequences to the three listed in the calculations of Table 1.1. One can verify the values for $P(s_1^T)$ and $P(y_1^T \mid s_1^T)$ in those calculations by applying (1.20) and (1.21).

The calculation of $P(y_1^4)$ in Table 1.1 is easy because, of the $3^4 = 81$ conceivable state sequences, only three are consistent with the observations and model structure. In general, however, if there are N_S states and an observation sequence with length T, then implementing

$$P(y_1^T) = \sum_{s_1^T} P\left(s_1^T, y_1^T\right)$$

naively requires order $(N_S)^T$ calculations. If N_S and T are as large as 100, $(N_S)^T$ is too many calculations for any conceivable computer.

There is a family of algorithms whose complexities are linear in the length T that make it possible to use HMMs with interestingly long time series. The details of these algorithms constitute Chapter 2; here I list only their names and objectives. In these descriptions, I denote by θ the vector of parameters that define an HMM, namely the state transition probabilities, the initial state probabilities, and the conditional observation probabilities:

$$\theta \equiv \left\{ \begin{array}{l} \left\{ \; P_{S(t+1)|S(t)} \left(s'|s\right) \; \forall s, s' \; \right\}, \\ \left\{ \; P_{S(1)} \left(s\right) \; \forall s \; \right\}, \\ \left\{ \; P_{Y(t)|S(t)} \left(y_i|s'\right) \; \forall y_i, s' \; \right\}. \end{array} \right\}.$$

The forward algorithm: For each time step t and each state s, the forward algorithm calculates the conditional probability of being in state s at time t given all of the observations up to that time, i.e., $P_{S(t)|Y_1^t,\theta}\left(s|y_1^t,\theta\right)$. It also calculates $P\left(y(t)|y_1^{t-1},\theta\right)$, the conditional probability of each observation given previous observations. Using these terms it calculates the probability of the entire data sequence given the model:

$$P\left(y_1^T|\theta\right) = P\left(y(1)|\theta\right) \cdot \prod_{t=2}^{T} P\left(y(t)|y_1^{t-1},\theta\right).$$

The forward algorithm is the first phase of the Baum–Welch algorithm.

The Viterbi algorithm: Given a model θ and a sequence of observations y_1^T, the Viterbi algorithm finds the most probable state sequence $\hat{s}_1^T$, i.e.,

$$\hat{s}_1^T = \underset{s_1^T}{\operatorname{argmax}} \; P\left(s_1^T|y_1^T,\theta\right). \tag{1.22}$$

In speech recognition algorithms, y_1^T is a representation of the sound pressure waveform, s_1^T determines a sequence of words, and a variant of the Viterbi algorithm estimates the latter from the former.

The Baum–Welch algorithm: (It is often called the *forward backward algorithm*.) Given a sequence of observations y_1^T and an initial set of model parameters θ_0, a single pass of the Baum–Welch algorithm calculates a new set of parameters θ_1 that has higher likelihood;

$$P\left(y_1^T|\theta_1\right) \geq P\left(y_1^T|\theta_0\right). \tag{1.23}$$

Equality can occur only at critical points of the likelihood function (where $\partial_\theta P\left(y_1^T|\theta\right) = 0$). In generic cases, running many iterations of the Baum–Welch algorithm yields a sequence θ_0^n that approaches a local maximum of the likelihood.

1.3.1 Example: Quantized Lorenz Time Series

To illustrate these algorithms, I have applied them to data that I synthesized by numerically integrating the Lorenz system ((1.1) with parameter values $r = 28$, $s = 10$, and $b = \frac{8}{3}$)

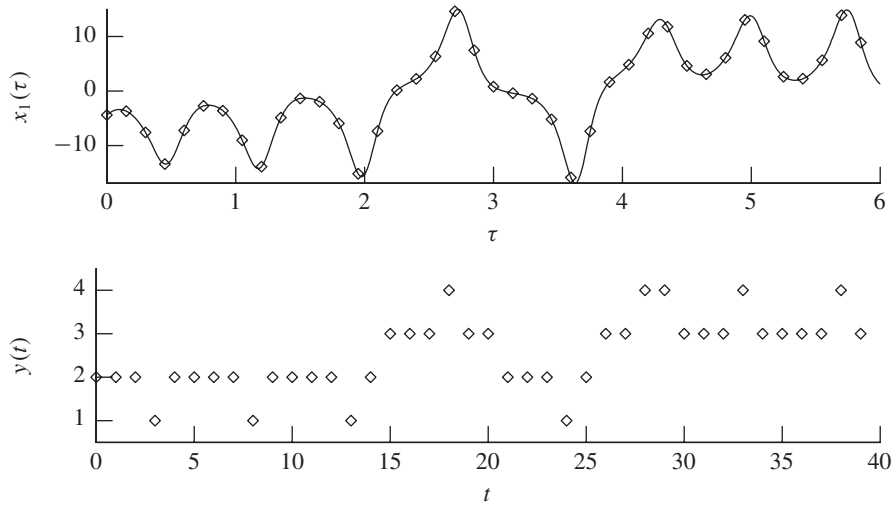


Figure 1.7. *Generating the observations y_1^{40}. The curve in the upper plot depicts the first component $x_1(\tau)$ of an orbit of the Lorenz system (see (* 1.1*)), and the points marked $\diamond$ indicate the values sampled with an interval $\tau_s = 0.15$. The points in the lower plot are the quantized values $y(t) \equiv \lceil \frac{x_1(t \cdot \tau_s)}{10} + 2 \rceil$, where $\lceil u \rceil$ is the least integer greater than or equal to u.*

and recording 40,000 vectors $x(\tau)$ with a sampling interval $\tau_s = 0.15$. Then I produced a sequence of integer valued observations $y_1^{40,000}$ by dividing the x_1 values by 10, adding 2, and keeping the integer part. The result is that, for each integer $1 \leq t \leq 40,000$, $x_1(t \cdot 0.15)$ yields $y(t) \in \{0, 1, 2, 3\}$. Fig. 1.7 depicts the first few observations.

I randomly generated an HMM with 12 hidden states[4] and four possible observations and then used 1,000 iterations of the Baum–Welch algorithm to select a set of parameters $\hat{\theta}$ with high likelihood for the data. Finally, I used the Viterbi algorithm to find the most likely state sequence:

$$\hat{s}_1^T = \underset{s_1^T}{\text{argmax}}\, P\left(s_1^T | y_1^T, \hat{\theta}\right).$$

Although the plot of *decoded* state values in Fig. 1.8 is not very enlightening, I can illustrate that there is a relationship between the learned decoded states and the original Lorenz system states by going back to the original data. For each state s, I identify the set of integer times t such that the decoded state is s, i.e., $\left\{t : \hat{s}(t) = s\right\}$, and then I find what the Lorenz system state was at each of these times and plot that set of points. In the upper right box of Fig. 1.9, I have plotted points in the original state space that correspond to hidden state number one, i.e., the set of pairs $\left\{(x_1(t \cdot \tau_s), x_3(t \cdot \tau_s)) : \hat{s}(t) = 1\right\}$. In searching for model parameters that give the observed data high likelihood, the Baum–Welch algorithm "discovers" a discrete hidden state structure, and Fig. 1.9 shows that the discrete hidden state structure is an approximation of the continuous state space that generated the data.

[4] I chose the number of hidden states to be 12 capriciously so that I could organize Fig. 1.9 on a 4×4 grid.

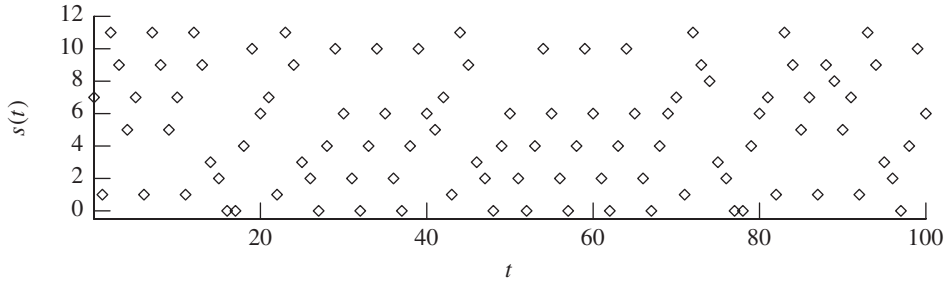


Figure 1.8. *A plot of the state sequence found by Viterbi decoding a quantized time series from the Lorenz system. Here the number of the decoded state $s(t)$ is plotted against time t. Although it is hard to see any structure in the plot because the numbers assigned to the states are not significant, Fig.* 1.9 *illustrates that the decoded states are closely related to positions in the generating state space.*

1.3.2 Example: Hidden States as Parts of Speech

HMMs were developed for speech and text processing. For unscientific audiences, I find the application to language modeling the easiest way to motivate HMMs. Consider, for example, the sentence, "The dog ate a biscuit." and its reduction to a sequence of parts of speech: *article noun verb article noun*. By choosing different particular articles, nouns, and verbs and placing them in the order specified by the sequence of parts of speech, I can produce many other sentences such as, "An equation describes the dynamics." The parts of speech are like hidden states, and the particular words are like observations.

Rather than using a dictionary or my own knowledge to build a model of language, here I describe the experiment of applying the Baum–Welch algorithm to some sample text to create an HMM. I hope that the experiment will *discover* parts of speech. I fetched the fourth edition of *A Book of Prefaces* by H. L. Mencken from *Project Gutenberg* [57], fit an HMM with the Baum–Welch algorithm, and decoded a state sequence with the Viterbi algorithm. I chose a book of essays rather than a novel because I expected that it would not have as much dialogue. I feared that different speakers in a novel would require different models.

The experiment consisted of the following steps:

Parse the text: I reduced the text to a sequence of tokens. Each token was a word, a number, or a special character such as punctuation. I retained distinctions between lower- and uppercase. The length of the resulting sequence was 68,857 tokens, i.e., w_1^T with $T = 68{,}857$.

Identify unique tokens: There were 9,772 unique tokens of which 2,759 appear in the text more than twice.

Create a map from tokens to rank: I sorted the tokens by the frequency of their occurrence so that for the most frequent token, w', $R(w') = 1$ and for the most infrequent token, $\bar{w}$, $R(\bar{w}) = 9,772$.

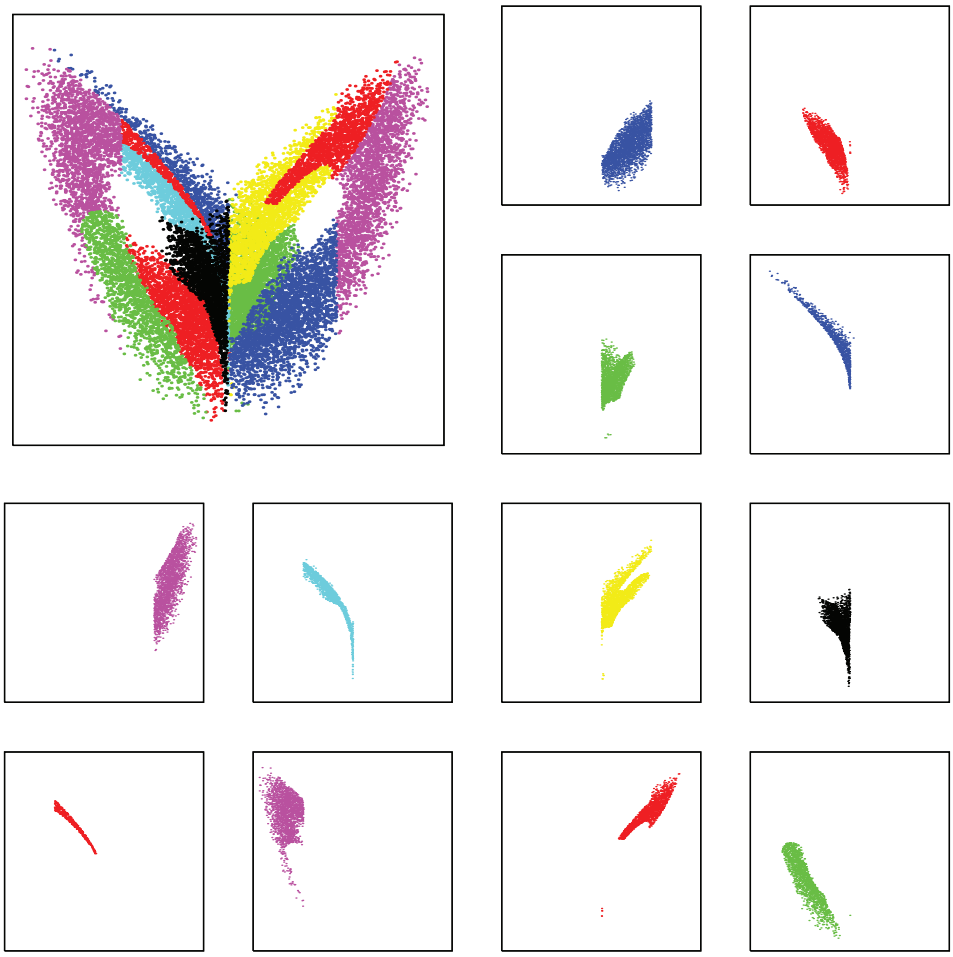


Figure 1.9. *The relationship between the hidden states of an HMM and the original coordinates of the Lorenz system.*

Map tokens to integers: I created a sequence of integers y_1^T, where $y(t) \in \{0, \ldots, 2760\}$ for all t. If the token $w(t)$ appeared in the text less than three times, I set $y(t) = 2,760$. Otherwise, I set $y(t)$ to $R(w(t))$.

Train an HMM: Starting from a 15 state model with random parameters, I used 100 iterations of the Baum–Welch algorithm to obtain a trained model.

Decode a sequence of states: By applying the Viterbi algorithm to y_1^T, I obtained s_1^T, where $s(t) \in \{1, \ldots, 15\}$.

Print the most frequent words for each state: For each state s, count the number of times each integer y occurs, i.e., $c(y, s) = \sum_{t:s(t)=s} \delta(y, y(t))$. Then print the words that

correspond to the 10 most frequently occurring integers (excluding the special value $y = 2{,}760$).

The results appear in Table 1.2. As I note in the caption, most of the states *do* correspond to parts of speech.

Table 1.2. *Words most frequently associated with each state. While I have no interpretation for three of the states, some of the following interpretations of the other states are strikingly successful.*

1 – Adjectives
2 – Punctuation and other tokens that appear at the end of phrases
6 – Capitalized articles and other tokens that appear at the beginning of phrases
7 – Objective pronouns and other words with similar functions
8 – Nouns
9 – Nouns
10 – Helping verbs
11 – Nominative pronouns
12 – Articles
13 – Conjunctions
14 – Prepositions
15 – Relative pronouns

1	other	first	American	same	new	own	whole	great	Puritan	very
2	.	,	;	that	]	?	"	and	-	!
3	more	be	out	not	X	only	on	much	up	make
4	"	indeed	Carrie	York	Sec	Titan	Gerhardt	Here	)	W
5	to	-	not	no	as	so	a	be	and	been
6	"	The	[	and	A	In	New	His	in	,
7	it	all	them	him	life	us	which	course	what	Dreiser
8	man	hand	book	artist	men	literature	books	end	story	work
9	book	sort	work	sense	man	way	thing	story	books	out
10	is	was	are	has	have	had	were	would	may	could
11	he	it	"	He	there	they	It	But	I	we
12	the	a	his	its	an	their	that	this	any	such
13	and	but	as	that	or	"	even	not	so	if
14	of	in	and	to	for	with	by	as	at	from
15	,	–	that	;	"	which	who	and	as	(

1.3.3 Remarks

Equations (1.18) and (1.19) state the assumptions of HMMs, i.e., that the state process is Markov and that the observations are conditionally independent given the states. While the assumptions may not seem symmetric in time, in fact they are. If you would like to verify the symmetry, you may wish to begin by deriving the following useful facts about

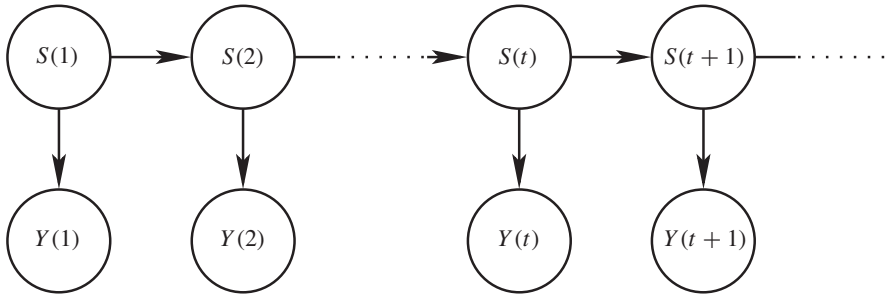


Figure 1.10. *Bayes' net schematic for an HMM. The drawn edges indicate the dependence and independence relations: Given $S(t)$, $Y(t)$ is conditionally independent of everything else, and given $S(t-1)$, $S(t+1)$, and $Y(t)$, $S(t)$ is conditionally independent of everything else.*

independence relations:

$$\begin{aligned} P(A|B,C) = P(A|B) &\iff P(A,C|B) = P(A|B)\cdot P(C|B) \\ &\iff P(C|A,B) = P(C|B), \qquad (1.24) \\ P(A|B,C,D) = P(A|B) &\iff P(A,C,D|B) = P(A|B)\cdot P(C,D|B) \\ &\implies P(A,C|B) = P(A|B)\cdot P(C|B) \\ &\iff P(A|B,C) = P(A|B). \qquad (1.25) \end{aligned}$$

The first chain of implications, (1.24), says that although a Markov assumption can be stated asymmetrically, it implies a symmetric relationship. The second chain, (1.25), says that if A is conditionally independent of C and D given B, then A is conditionally independent of C alone given B. By symmetry, A is also conditionally independent of D given B.

From the assumptions ((1.18) and (1.19)), one can derive the following:

The joint process is Markov:

$$P_{Y_{t+1}^T, S_{t+1}^T | Y_1^t, S_1^t} = P_{Y_{t+1}^T, S_{t+1}^T | Y(t), S(t)}.$$

Given $S(t)$, $Y(t)$ is conditionally independent of everything else:

$$P_{Y(t)|Y_1^{t-1}, Y_{t+1}^T, S_1^T} = P_{Y(t)|S(t)}.$$

The assumptions describe *conditional independence* relations. Fig. 1.10 represents these relations as a *Bayes' net* [13, 1].

One might imagine that HMMs are simply higher order Markov processes. For example, consider the suggestion that the states depicted in Fig. 1.9 correspond to sequential pairs of observations and that the model is a second order Markov model that is characterized by $P_{Y(t+1)|Y(t),Y(t-1)}$, the set of observation probabilities conditioned on the *two* previous observations. Although the number of unique sequential pairs y_t^{t+1} that occur in the data is in fact 12, the fact that some of the states in Fig. 1.9 straddle the quantization boundaries

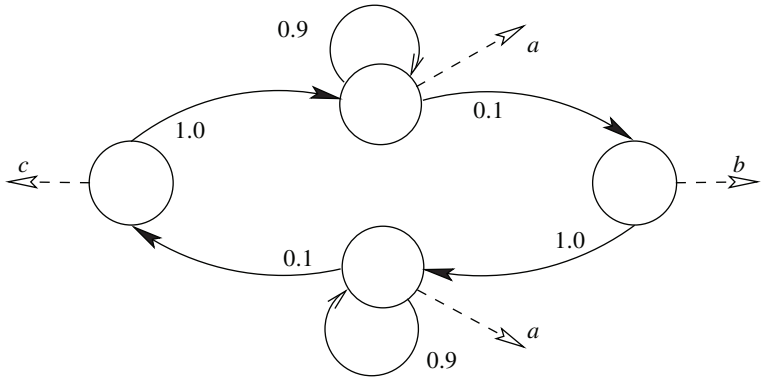


Figure 1.11. *An HMM that cannot be represented by a Markov model of any order. Consider the string of observations "$b, a, a, \ldots, a, a, a$." For each "a" in the string, the previous non-"a" observation was "b." Since the model will not produce another "b" before it produces a "c," the next observation can be either a "c" or another "a" but not a "b." Because there is no limit on the number of consecutive "a's" that can appear, there is no limit on how far back in the observation sequence you might have to look to know the probabilities of the next observation.*

at $x = -10$ and $x = 10$ belies the suggestion. In general, the class of HMMs is more powerful than the class of simple Markov models in the sense that the former includes the latter but not vice versa.

To conclude this chapter, note the following points about discrete HMMs:

1. Although the hidden process is first order Markov, the observation process may not be a Markov process (of any order).

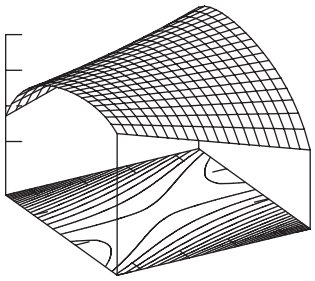
2. Any Markov model of any order can be represented by an HMM.

3. Even if the functions governing the dynamics and observations of a continuous state space system are nonlinear, a discrete HMM can approximate the system arbitrarily well by using large numbers of states N_S and possible observation values N_Y.

4. For estimating model parameters, larger numbers of training data are required as N_S and N_Y are increased.

As an illustration of point 1, consider the process depicted in Fig. 1.11, which produces observation strings consisting of runs of a's interspersed with occasional b's and c's. In the observation stream, the b's and c's alternate no matter how long the runs of a's are that fall in between. Such behavior cannot be captured by a simple Markov process of any order.

The possibility of long term memory makes state space models, e.g., HMMs, more powerful than Markov models. That observation suggests that if there is noise, then the *delay vector reconstructions* described in the chaos literature [44, 50, 32, 23] are suboptimal because they discard information from earlier observations that could be used to more accurately specify the state.



Chapter 2
Basic Algorithms

In *Hidden Markov Analysis: An Introduction* [18], Ferguson wrote the following:

> The three basic problems of hidden Markov analysis are
>
> 1. computing $P(y_1^T|\theta)$;
> 2. adjusting model parameters to maximize $P(y_1^T|\theta)$;
> 3. choosing a state sequence s_1^T which is optimal in some sense given y_1^T.

In the same order as Ferguson stated the problems, these are the names of the solutions:

The forward algorithm: Given a vector of model parameters θ and a sequence of observations y_1^T, the forward algorithm calculates $P(y_1^T|\theta)$ recursively using $O(T)$ calculations. It is the discrete analogue of the celebrated Kalman filter for linear Gaussian systems (see Chapter 4).

The Baum–Welch algorithm: Also called *the forward backward algorithm*, the Baum–Welch algorithm takes a sequence of observations y_1^T and implements a map from parameter space to itself. Each iteration of the map uses $O(T)$ calculations and increases the likelihood, i.e., $P(y_1^T|\theta(n+1)) \geq P(y_1^T|\theta(n))$. For all initial choices of parameters, except perhaps those lying in a pathological low-dimensional set with zero volume, the iterations converge linearly to a local maximum.

The Viterbi algorithm: Given a vector of model parameters θ and a sequence of observations y_1^T the Viterbi algorithm calculates $\hat{s}_1^T \equiv \text{argmax}_{s_1^T} P(s_1^T|y_1^T)$ recursively using $O(T)$ calculations. It is the discrete analogue of *smoothing* for linear Gaussian systems (again, see Chapter 4).

In this chapter, Section 2.1 describes the forward algorithm, Section 2.3 describes the Viterbi algorithm, and Section 2.4 describes the Baum–Welch algorithm. Also, Section 2.2 describes the *backward algorithm* which, like the forward algorithm, is a key component of the Baum–Welch algorithm.

Much of the literature on the algorithms and much of the available computer code use Ferguson's notation [18]. In particular, Rabiner's widely cited article [22] follows Ferguson's notation. Now I will describe this book's notation for the model parameters.

$\boldsymbol{P_{S(t+1)|S(t)}(s|\tilde{s})}$ The probability that at time $t+1$ the system will be in state s given that at time t it was in state $\tilde{s}$. Notice that the parameter is independent of time t. Ferguson called these parameters *the A matrix* with $a_{i,j} = P_{S(t+1)|S(t)}(j|i)$.

$\boldsymbol{P_{S(1)}(s)}$ The probability that the system will start in state s at time 1. Ferguson used a to represent this vector of probabilities with $a_i = P_{S(1)}(i)$.

$\boldsymbol{P_{Y(t)|S(t)}(y_k|\tilde{s})}$ The probability that the observation value is y_k given that the system is in state $\tilde{s}$. Ferguson used b to represent this with $b_j(k) = P_{Y(t)|S(t)}(k|\tilde{j})$.

$\boldsymbol{\theta}$ The entire collection of parameters. For example, iteration of the Baum–Welch algorithm produces a sequence of parameter vectors θ_1^N with $P\left(y_1^T|\theta(n+1)\right) \geq P\left(y_1^T|\theta(n)\right) : 1 < n < N$. Instead of θ, Ferguson used λ to denote the entire collection of parameters.

2.1 The Forward Algorithm

Given θ, the forward algorithm calculates $P(y_1^T|\theta)$ and several useful intermediate terms. Fig. 2.1 sketches the basic recursion's structure. Since I am considering only one set of parameters, i.e., I am assuming that θ is known and fixed, I will drop the dependence of probabilities on θ from the notation for the remainder of this section. In the example of Table 1.1, I found that I could calculate the right-hand terms of

$$P\left(y_1^T\right) = \sum_{s_1^T} P\left(y_1^T, s_1^T\right)$$

easily from the model assumptions ((1.18) and (1.19)). Unfortunately, the number of possible sequences s_1^T is exponential in T, and it is impossible to do the sum even for modest lengths T.

The forward algorithm regroups the terms and produces the desired result using $O(T)$ calculations. For each time $t : 1 < t \leq T$ one calculates $P(y(t)|y_1^{t-1})$. Although one can write

$$P(y_1^T) = P(y(1)) \prod_{t=2}^{T} P(y(t)|y_1^{t-1}),$$

the values of $P(y(t)|y_1^{t-1})$ are typically small compared to 1, and the product of many such terms is too small to be represented even in double precision. Working with logarithms avoids underflow:

$$\log\left(P(y_1^T)\right) = \log\left(P(y(1))\right) + \sum_{t=2}^{T} \log\left(P(y(t)|y_1^{t-1})\right). \tag{2.1}$$

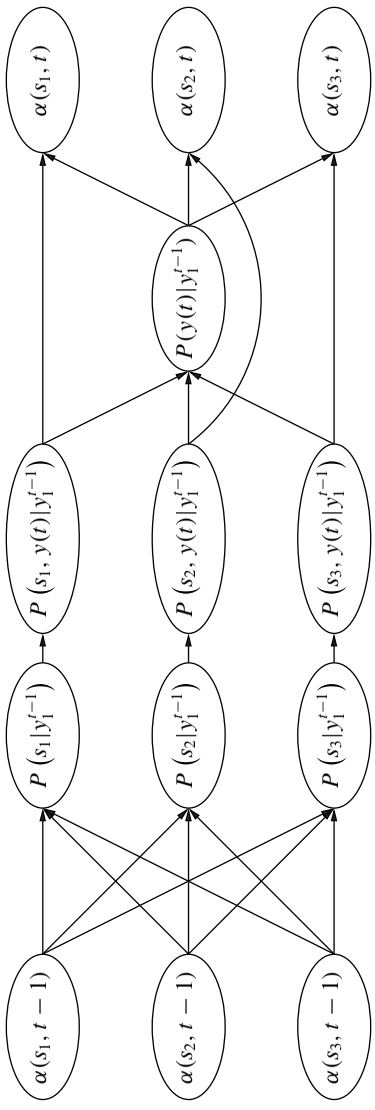


Figure 2.1. *Dependency relations in the forward algorithm (see (2.4)–(2.7) in the text). The figure indicates the calculations the algorithm executes to incorporate the observation at time t for a three state model.*

For each time step one does the four calculations specified in (2.4)–(2.7) below. In these equations, I use boxes to emphasize repeated instances of the same quantity. The algorithm saves the intermediate results:

$$\alpha(s,t) \equiv \boxed{\boxed{P_{S(t)|Y_1^t}\left(s|y_1^t\right)}}, \tag{2.2}$$

$$\gamma(t) \equiv \boxed{\boxed{P(y(t)|y_1^{t-1})}}. \tag{2.3}$$

In words,[5] $\alpha(s,t)$ is the conditional probability of being in state s at time t given all of the observations up to time t, and $\gamma(t)$ is the conditional probability of the observation at time t given all of the previous observations. To initialize the algorithm, one assigns

$$\alpha(s,1) = P_{S(1)|Y(1)}(s|y(1)) = \frac{P_{S(1)}(s)P_{Y(t)|S(t)}(y(1)|s)}{\sum_{\tilde{s}} P_{S(1)}(\tilde{s})P_{Y(t)|S(t)}(y(1)|\tilde{s})} \quad \forall s \in \mathcal{S},$$

where the distributions $P_{S(1)}$ and $P_{Y(t)|S(t)}$ are model parameters.

Forecast the distribution of states. Consider each $s \in \mathcal{S}$ in turn, and calculate the conditional probability, given the previous observations, of the system being in state s at time t:

$$\boxed{P_{S(t)|Y_1^{t-1}}\left(s|y_1^{t-1}\right)} = \sum_{\tilde{s}\in\mathcal{S}} P_{S(t),S(t-1)|Y_1^{t-1}}\left(s,\tilde{s}|y_1^{t-1}\right) \tag{2.4a}$$

$$= \sum_{\tilde{s}\in\mathcal{S}} \left(P_{S(t)|S(t-1),Y_1^{t-1}}\left(s|\tilde{s},\, y_1^{t-1}\right) \right.$$

$$\left. \times \boxed{\boxed{P_{S(t-1)|Y_1^{t-1}}\left(\tilde{s}|y_1^{t-1}\right)}} \right) \tag{2.4b}$$

$$= \sum_{\tilde{s}\in\mathcal{S}} P_{S(t)|S(t-1)}\left(s|\tilde{s}\right) \cdot \alpha(\tilde{s}, t-1). \tag{2.4c}$$

I justify the operations as follows:

(2.4a) $P(a) = \sum_b P(a,b)$.

(2.4b) $P(a|b) \cdot P(b) = P(a,b)$, i.e., Bayes' rule.

(2.4c) Assumption of (1.19): The state process is Markov.

Update the joint probability of states and the current observation. Find for time t and each possible state s the conditional probability, given the previous observations, of

[5] In Ferguson's notation $\alpha(s,t) \equiv P_{S(t),Y_1^t}\left(s, y_1^t\right)$. My notation differs from that by a factor of $P(y_1^t)$. Ferguson did not have notation for the term $P(y(t)|y_1^{t-1})$, which I call γ, but he did use γ for quantities that I will denote by w in (2.20) and (2.21).

the state being s and the observation being $y(t)$ (the observation actually observed):

$$\boxed{P_{S(t),Y(t)|Y_1^{t-1}}\left(s, y(t)|y_1^{t-1}\right)} = P_{Y(t)|S(t),Y_1^{t-1}}\left(y(t)|s, y_1^{t-1}\right) \times \boxed{P_{S(t)|Y_1^{t-1}}\left(s|y_1^{t-1}\right)} \tag{2.5a}$$

$$= P_{Y(t)|S(t)}\left(y(t)|s\right) \cdot \boxed{P_{S(t)|Y_1^{t-1}}\left(s|y_1^{t-1}\right)}. \tag{2.5b}$$

I justify the equations as follows:

(2.5a) Bayes' rule.

(2.5b) Assumption of (1.18): The observations are conditionally independent given the states.

Calculate the conditional probability of the current observation. Find for time t the conditional probability, given the previous observations, of the observation being $y(t)$ (the observation actually observed):

$$\gamma = \boxed{\boxed{P(y(t)|y_1^{t-1})}} = \sum_{s\in\mathcal{S}} \boxed{P_{S(t),Y(t)|Y_1^{t-1}}\left(s, y(t)|y_1^{t-1}\right)}. \tag{2.6}$$

Equation (2.6) is an application of $P(a) = \sum_b P(a, b)$.

Normalize the updated distribution of states. For each possible state s, find the conditional probability of being in that state at time t given all of the observations up to time t. Note that this differs from the first calculation in that the conditioning event includes $y(t)$:

$$\alpha(s,t) \equiv \boxed{\boxed{P_{S(t)|Y_1^t}\left(s|y_1^t\right)}} \tag{2.7a}$$

$$= \boxed{P_{S(t),Y(t)|Y_1^{t-1}}\left(s, y(t)|y_1^{t-1}\right)} \div \boxed{\boxed{P(y(t)|y_1^{t-1})}}. \tag{2.7b}$$

Equation (2.7b) is an application of Bayes' rule, i.e., $P(a|b) \cdot P(b) = P(a, b)$.

2.1.1 Vector Notation

By using notation or a computer language that includes vector and matrix operations, one can write the four major steps in an iteration of the forward algorithm in four lines, e.g.,

$$\mathbf{f} = \alpha[t] \cdot \mathbf{A}, \tag{2.8a}$$

$$\mathbf{u} = \mathbf{f} * \mathbf{b}[y[t+1]], \tag{2.8b}$$

$$\gamma[t+1] = \sum \mathbf{u}, \tag{2.8c}$$

$$\alpha[t+1] = \mathbf{u}/\gamma[t+1]. \tag{2.8d}$$

In these equations I have used notation as follows:

- In each of the equations, $[\tau]$ selects the τth element of a list or matrix.
- $\mathbf{f}$, $\alpha[t]$, $\alpha[t+1]$, and $\mathbf{u}$ are all N_S-dimensional row vectors, where N_S is the number of states.
- In (2.8a), $\alpha[t] \cdot \mathbf{A}$ is a standard vector matrix product, where $\mathbf{A}$ is the $N_S \times N_S$ state transition matrix with elements $a_{i,j} = P_{S(t+1)|S(t)}(j|i)$ and $\mathbf{f}$ is the forecast distribution of states with elements $f_j = P_{S(t+1)|Y_1^t}(j|y_1^t)$.
- In (2.8b), $\mathbf{b}$ is the $N_Y \times N_S$ observation probability matrix with elements $b_{i,j} = P_{Y(t)|S(t)}(i|j)$, $\mathbf{b}[y]$ indicates the particular row of $\mathbf{b}$ corresponding to the observation y, $*$ indicates an elementwise product, and $\mathbf{u}$ is the unnormalized updated distribution of states with components $u_i = P_{S(t+1),Y(t+1)|Y_1^t}(i, y(t+1)|y_1^t)$.
- In (2.8c), $\sum \mathbf{u}$ indicates the sum of the elements of $\mathbf{u}$, i.e., a sum over states.
- And finally, in (2.8d), $\mathbf{u}/\gamma[t+1]$ is a standard vector-scalar division.

In Appendix B, I present four lines of Python code that implement the calculations of (2.8).

2.1.2 General Filtering

The forward algorithm handles observations sequentially. After handling the observation at time t, it recursively calculates the conditional distribution of states, i.e., $P(s(t)|y_1^t)$. While I have given the details of the calculations for the discrete states and observations of an HMM, the procedure also works for other state spaces and observations. Perhaps the best known special case is the *Kalman filter*, which I describe in Chapter 4.

In general, the procedure is called *filtering*. With the definitions

$\alpha_t \equiv P(s(t)\|y_1^t)$	the updated distribution,
$f_t \equiv P(s(t)\|y_1^{t-1})$	the forecast distribution,
$\gamma_t \equiv P(y(t)\|y_1^{t-1})$	the incremental likelihood

and a model characterized by the distributions

α_0	the state prior,
$P(s(t+t)\|s(t))$	the state transition probability,
$P(y(t)\|s(t))$	the conditional observation probability

one can write recursive filtering in general as follows:

$$
\begin{aligned}
f_t &= \mathbb{E}_{S(t-1):\alpha_{t-1}}(P(s(t)|S(t-1))),\\
\gamma_t &= \mathbb{E}_{S(t):f(t)}(P(y(t)|S(t))),\\
\alpha_t &= \frac{f_t \cdot P(y(t)|s(t))}{\gamma_t},
\end{aligned}
$$

where the notation $\mathbb{E}_{S(t):f(t)}F(S(t))$ means the *expected value* of the function F over all values of $S(t)$ using the distribution of $S(t)$ specified by $f(t)$. Note that the distributions that characterize the model need not be discrete. If they are characterized by densities, integrals implement the expectation operations.

2.2 The Backward Algorithm

The backward algorithm is similar to the forward algorithm in structure and complexity, but the terms are not as easy to interpret. After running the forward algorithm, the backward algorithm enables one to calculate $P_{S(t)|Y_1^T}\left(s|y_1^T\right)$, the conditional probability of being in any state $s \in \mathcal{S}$ at any time $t : 1 \leq t \leq T$ given the entire sequence of observations. The forward algorithm provides the terms $\alpha(s,t) \equiv P_{S(t)|Y_1^t}\left(s|y_1^t\right)$ for all (s,t). I will derive the backward algorithm by solving for the terms called $\beta(s,t)$ that yield $P_{S(t)|Y_1^T}\left(s|y_1^T\right)$ when multiplied by $\alpha(s,t)$. In addition to being useful for calculating the distribution of $S(t)$ given *all* of the data, the reestimation formulas of the Baum–Welch algorithm in Section 2.4 use the β terms.

I begin by deriving three expressions for $\beta(s,t)$; one from the property described in the preceding paragraph, a second form for comparison to $\alpha(s,t)$, and a third form that helps explain the recursive algorithm. Equation (2.9) states the desired property:

$$\beta(s,t) = \frac{P_{S(t)|Y_1^T}\left(s|y_1^T\right)}{\alpha(s,t)} \tag{2.9a}$$

$$= \frac{P_{S(t)|Y_1^T}\left(s|y_1^T\right)}{P_{S(t)|Y_1^t}\left(s|y_1^t\right)} \tag{2.9b}$$

$$= \frac{P\left(s|y_1^T\right)}{P\left(s|y_1^t\right)}. \tag{2.9c}$$

In (2.9c), I have begun dropping the subscripts on P.

Starting with (2.9c), I next derive the following form that presents $\beta(s,t)$ as a backward forecast, $P(s(t)|y_{t+1}^T)$, times a factor that has the a priori probability of $s(t)$ in the denominator:

$$\frac{P\left(s|y_1^T\right)}{P\left(s|y_1^t\right)} = \frac{P(s, y_1^T)\, P(y_1^t)}{P(s, y_1^t)\, P(y_1^T)} \tag{2.10a}$$

$$= \frac{P(y_1^T|s)\, P(s)\, P(y_1^t)}{P(y_1^t|s)\, P(s)\, P(y_1^T)} \tag{2.10b}$$

$$= \frac{P(y_1^t|s)\, P(y_{t+1}^T|s)\, P(y_1^t)}{P(y_1^t|s)\, P(y_1^T)} \tag{2.10c}$$

$$= \frac{P(y_{t+1}^T, s)\, P(y_1^t)}{P(s)\, P(y_1^T)} \tag{2.10d}$$

$$= P(s|y_{t+1}^T)\frac{P(y_{t+1}^T)\, P(y_1^t)}{P(s)\, P(y_1^T)}. \tag{2.10e}$$

I justify the above as follows:

(2.10a) Bayes' rule applied to numerator and denominator.

(2.10b) Bayes' rule applied to numerator and denominator.

(2.10c) Assumption of conditional independence of past and future given state.

(2.10d) Bayes' rule applied to $P(y_{t+1}^T|s)$.

(2.10e) Bayes' rule applied to $P(y_{t+1}^T, s)$.

Applying Bayes' rule to $\frac{P(y_1^t)}{P(y_1^T)}$ in (2.10c) yields

$$\beta(s,t) \equiv \frac{P_{Y_{t+1}^T|S(t)}\left(y_{t+1}^T|s\right)}{P\left(y_{t+1}^T|y_1^t\right)}, \tag{2.11}$$

which I will use in explaining the backward algorithm. Note that if $\alpha(s,t) = 0$, (2.9) is undefined, but one can nonetheless use the definition, (2.11).

The backward algorithm starts at the final time T with β set to one[6] for each state, $\beta(s,T) = 1$ for all $s \in \mathcal{S}$, and solves for β at earlier times with the following recursion that goes *backwards* through time:

$$\beta(\tilde{s}, t-1) = \sum_{s\in\mathcal{S}} \beta(s,t) \frac{P_{Y(t)|S(t)}\left(y(t)|s\right) \cdot P_{S(t)|S(t-1)}\left(s|\tilde{s}\right)}{\gamma(t)}. \tag{2.12}$$

Note that $\gamma(t) \equiv P\left(y(t)|y_1^{t-1}\right)$ is calculated by the forward algorithm and that the terms in the numerator are model parameters. To justify (2.12), start with (2.11) and write

$$\beta(\tilde{s}, t-1) \equiv \frac{P_{Y_t^T|S(t-1)}\left(y_t^T|\tilde{s}\right)}{P\left(y_t^T|y_1^{t-1}\right)} \tag{2.13}$$

$$= \frac{\sum_{s\in\mathcal{S}} P_{Y_t^T,S(t)|S(t-1)}\left(y_t^T, s|\tilde{s}\right)}{P\left(y_t^T|y_1^{t-1}\right)}. \tag{2.14}$$

Now drop the subscripts, factor each term in the numerator of (2.14) using Bayes' rule twice

$$P\left(y_t^T, s|\tilde{s}\right) = P\left(y_{t+1}^T|y(t), s, \tilde{s}\right) P\left(y(t)|s, \tilde{s}\right) \cdot P\left(s|\tilde{s}\right),$$

apply the model assumptions to simplify the conditioning events

$$P\left(y_t^T, s|\tilde{s}\right) = P\left(y_{t+1}^T|s\right) \cdot P\left(y(t)|s\right) \cdot P\left(s|\tilde{s}\right),$$

and substitute in the expression for $\beta(s,t)$ from (2.11):

$$P\left(y_t^T, s|\tilde{s}\right) = \beta(s,t) \cdot P\left(y_{t+1}^T|y_1^t\right) \cdot P\left(y(t)|s\right) \cdot P\left(s|\tilde{s}\right).$$

Next, expand the denominator of (2.14) using Bayes' rule:

$$P\left(y_t^T|y_1^{t-1}\right) = P\left(y_{t+1}^T|y_1^t\right) \cdot P\left(y(t)|y_1^{t-1}\right).$$

Finally, by substituting these values into the fraction of (2.14), obtain the recursion (2.12):

$$\beta(\tilde{s}, t-1) = \frac{\sum_{s\in\mathcal{S}} \beta(s,t) \cdot P\left(y_{t+1}^T|y_1^t\right) \cdot P_{Y(t)|S(t)}\left(y(t)|s\right) \cdot P_{S(t)|S(t-1)}\left(s|\tilde{s}\right)}{P\left(y_{t+1}^T|y_1^t\right) \cdot P\left(y(t)|y_1^{t-1}\right)}$$

$$= \sum_{s\in\mathcal{S}} \beta(s,t) \frac{P_{Y(t)|S(t)}\left(y(t)|s\right) \cdot P_{S(t)|S(t-1)}\left(s|\tilde{s}\right)}{\gamma(t)}.$$

[6]The premises that $\alpha(s,t)\beta(s,t) = P_{S(t)|Y_1^T}\left(s|y_1^T\right)$ and $\alpha(s,T) \equiv P_{S(T)|Y_1^T}\left(s|y_1^T\right)$ imply that $\beta(s,T) = 1$ for all s.

2.3 The Viterbi Algorithm

For some applications, one needs to estimate a sequence of states from a sequence of observations. The Viterbi algorithm finds the *best* sequence $\hat{s}_1^T$ in the sense of maximizing the probability $P\left(s_1^T | y_1^T\right)$. That is equivalent to maximizing $\log\left(P\left(y_1^T, s_1^T\right)\right)$ because $P\left(y_1^T\right)$ is simply a constant, and the log is monotonic, i.e.,

$$\begin{aligned}\hat{s}_1^T &\equiv \underset{s_1^T}{\text{argmax}}\, P(s_1^T | y_1^T) \\ &= \underset{s_1^T}{\text{argmax}} \left(P(s_1^T | y_1^T) \cdot P(y_1^T)\right) \\ &= \underset{s_1^T}{\text{argmax}} \log\left(P(y_1^T, s_1^T)\right).\end{aligned}$$

As in the implementation of the forward algorithm, using logs avoids numerical underflow. If I define $\log\left(P\left(y_1^t, s_1^t\right)\right)$ as the *utility* (negative cost) of the state sequence s_1^t given the observation sequence y_1^t, then the Viterbi algorithm finds the maximum utility state sequence.

Initially one calculates $\log\left(P_{Y(1),S(1)}\left(y(1), s\right)\right)$ for each state $s \in \mathcal{S}$. Then for each successive time step $t : 1 < t \leq T$ one considers each state and determines the best predecessor for that state and the utility of the best state sequence ending in that state. The Viterbi algorithm and the forward algorithm have similar structures (see Fig. 2.1 and Fig. 2.2). Roughly, the forward algorithm does a sum over predecessor states, while the Viterbi algorithm finds maxima.

Using the definitions

$$\begin{aligned}u(s_1^t) \quad & \text{utility of state sequence } s_1^t \\ & \equiv \log\left(P(y_1^t, s_1^t)\right), \\ v(s,t) \quad & \text{utility of best sequence ending with } s(t) = s \\ & \equiv \max_{s_1^t : s(t)=s} u(s_1^t), \\ \omega(s,s',t) \quad & \text{utility of best sequence ending with } s(t), s(t+1) = s, s' \\ & \equiv \max_{s_1^{t+1} : s(t)=s \,\&\, s(t+1)=s'} u(s_1^{t+1}), \\ B(s',t) \quad & \text{best predecessor state given } s(t+1) = s' \\ & \equiv \underset{s}{\text{argmax}}\, \omega(s,s',t),\end{aligned}$$

I can describe the two phases of the algorithm. In a forward pass calculate and save the two-dimensional array of $B(s,t)$ values for all $t \in [2, \ldots, T]$ and $s \in \mathcal{S}$. For each t also calculate the one-dimensional array of $v(s,t)$ values for all $s \in \mathcal{S}$. Do the calculations recursively as follows:

$$\omega(s,s',t) = v(s,t) + \log\left(P_{s(t+1)|s(t)}(s'|s)\right) + \log\left(P_{y(t+1)|s(t+1)}(y(t+1)|s')\right), \quad (2.15)$$

$$B(s',t) = \underset{s}{\text{argmax}}\, \omega(s,s',t), \quad (2.16)$$

$$v(s',t+1) = \omega(B(s',t), s', t). \quad (2.17)$$

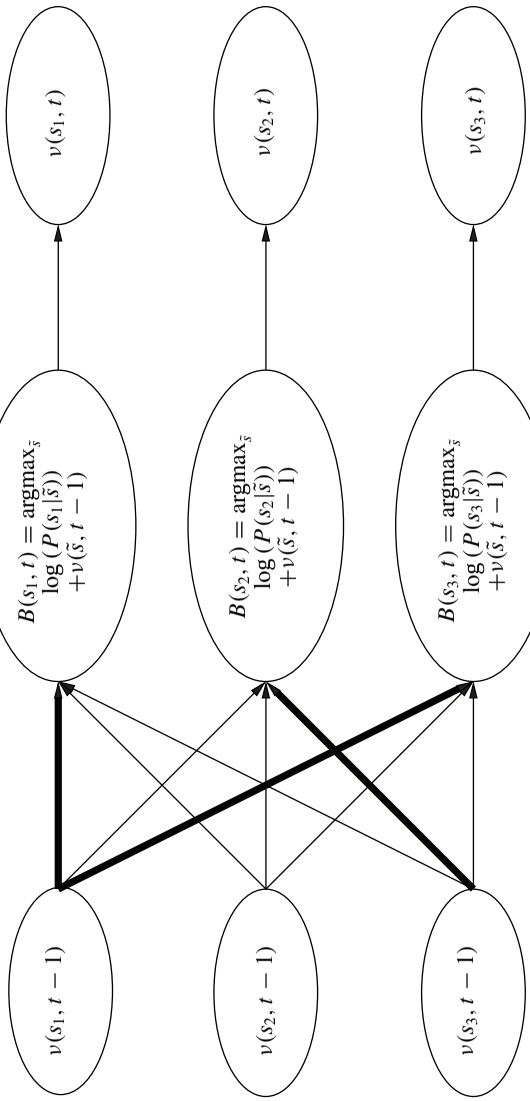


Figure 2.2. *Dependency relations in the Viterbi algorithm.*

After the final iteration of the forward pass, identify the best final state as

$$\hat{s}(T) = \underset{s}{\operatorname{argmax}}\, \nu(s, T).$$

Finally, backtrack through the B array to find the other states in the sequence $\hat{s}_1^T$, i.e.,

$$\begin{aligned} &\text{for } t \text{ from } T-1 \text{ to } 1: \\ &\quad \hat{s}(t) = B(\hat{s}(t+1), t+1). \end{aligned}$$

Equation (2.15) says that the total utility of the best path ending in state s at time t and state s' at $t+1$ is the utility of the best path ending in state s at time t plus two terms, one that accounts for the transition from s to s' and one that accounts for the probability of state s' producing the observation $y(t+1)$. The equation follows from taking logs of

$$\begin{aligned} &P_{Y(t+1),Y_1^t,S(t+1),S_1^t}\left(y(t+1), y_1^t, s', s_1^t\right) \\ &\qquad = P\left(y_1^t, s_1^t\right) \cdot P_{S(t+1)|S(t)}\left(s'|s\right) \cdot P_{Y(t+1)|S(t+1)}\left(y(t+1)|s'\right), \end{aligned}$$

which in turn follows from the model assumptions ((1.18) and (1.19)).

Note that for each time step in the forward pass of the algorithm one must calculate each of the $N_S \times N_S$ terms in $\omega(s, s', t)$, and then for each value of s' one must consider N_S possibilities to find $B(s', t)$ and $\nu(s', t+1)$. Thus the algorithm has $O\left((N_S)^2 T\right)$ complexity. Also note that in (2.15) the term $P(y(t+1)|s')$ is the same and occurs in each $\omega(s, s', t)$ regardless of the value of s. Thus the term makes no difference in the calculation of (2.16), and moving it to (2.17) reduces the number of calculations without affecting the result. Fig. 2.3 depicts such calculations for one time step of the forward pass of the algorithm for a three state model.

2.3.1 General Decoding

Given an HMM and a sequence of observations, the Viterbi algorithm returns the state sequence that maximizes $P(s_1^T | y_1^T)$, which is called *maximum* a posteriori (MAP) estimate of the state sequence. As in Section 2.1.2, where I described the generalization of the *forward algorithm* to general *filtering*, here I note that the Viterbi algorithm can be applied to more general models. For a continuous state space, the key is to describe the functions $B(s, t)$ and $\nu(s, t)$ with a few parameters.

2.3.2 MAP Sequence of States or Sequence of MAP States?

Here I consider the difference between the sequence of MAP states and the MAP sequence of states. The MAP state at a particular time t is the best guess for where the system was at that time. While it seems reasonable that a sequence of such guesses would constitute a good guess for the entire trajectory, it is not an optimal trajectory estimate. In fact, as the following example demonstrates, such a trajectory may even be impossible.

Consider the HMM that is drawn in Fig. 2.4 and the sequence of observations $y_1^6 = (a, b, b, b, b, c)$. Obviously, any sequence of states that is consistent with y_1^6 must begin in e, end in g, and pass through state f exactly once. The only unknown remaining is the time at which the system was in state f. Here is a tabulation of the four possible state sequences:

Initialize:
  for each s
    $\nu_{\text{next}}(s) = \log\left(P_{Y(1),S(1)}\,(y(1), s)\right)$

Iterate:
  for t from 2 to $T-1$
    Swap $\nu_{\text{next}} \leftrightarrow \nu_{\text{old}}$
    for each s_{next}

      for each s_{old}
        $\omega(s_{\text{old}}, s_{\text{next}}) = \nu(s_{\text{old}}, t) + \log\left(P(s_{\text{next}}|s_{\text{old}})\right)$
          $+ \log\left(P(y(t+1)|s_{\text{next}})\right)$

      # Find best predecessor
      $B(s_{\text{next}}, t+1) = \text{argmax}_{s_{\text{old}}}\, \omega(s_{\text{old}}, s_{\text{next}})$

      # Update ν
      $\nu_{\text{next}}(s_{\text{next}}) = \omega(B(s_{\text{next}}, t+1), s_{\text{next}})$
Backtrack:
  $\bar{s} = \text{argmax}_s\, \nu_{\text{next}}(s)$
  $\hat{s}(T) = \bar{s}$
  for t from $T-1$ to 1
    $\bar{s} = B(\bar{s}, t+1)$
    $\hat{s}(t) = \bar{s}$

Figure 2.3. *Pseudocode for the Viterbi algorithm.*

$s(1)$	$s(2)$	$s(3)$	$s(4)$	$s(5)$	$s(6)$	$P(y_1^6, s_1^6)/z$	$P(s_1^6\|y_1^6)$
e	e	e	e	f	g	0.9^3	0.30
e	e	e	f	g	g	$0.9^2 \cdot 0.8$	0.26
e	e	f	e	g	g	$0.9 \cdot 0.8^2$	0.23
e	f	g	g	g	g	0.8^3	0.21

In the table, the term z represents the factors that are common in $P(y_1^6, s_1^6)$ for all of the possible state sequences. Only the factors that are different appear in the seventh column. The largest entry in the last column is 0.30, which corresponds to the MAP estimate: $\hat{s}_1^6 = (e, e, e, e, f, g,)$.

The next table displays the values of $P(s(t)|y_1^6)$, the a posteriori probability for the three possible states:

		t					
$P(s(t)\|y_1^6)$		1	2	3	4	5	6
	e	**1.0**	**0.79**	**0.56**	0.30	0	0
s	f	0	0.21	0.23	0.26	0.30	0
	g	0	0	0.21	**0.44**	**0.70**	**1.0**

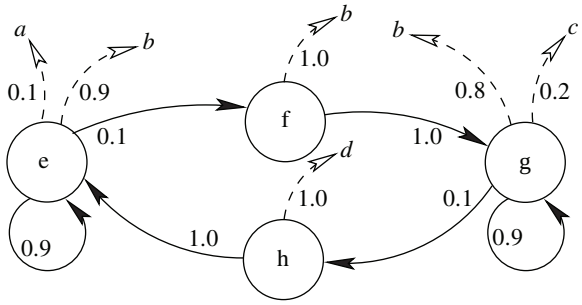


Figure 2.4. *An HMM used to illustrate that the MAP sequence of states is not the same as the sequence of MAP states.*

The table quantifies the intuition that the a posteriori probability starts in state e at time $t = 1$ and sort of diffuses completely over to state g by time $t = 6$. Notice that although all of the probability passes through state f, at no time is it the most probable state. Thus the sequence of MAP states is e, e, e, g, g, g, which is an *impossible* sequence. On the other hand, the MAP sequence of states, e, e, e, e, f, g, is entirely plausible.

2.4 The Baum–Welch Algorithm

Given an initial vector of model parameters θ for an HMM and a sequence of observations y_1^T, iteration of the Baum–Welch algorithm produces a sequence of parameter vectors θ_1^N that almost always converges to a local maximum of the likelihood function $P_\theta\left(y_1^T\right)$. The algorithm was developed by Baum and collaborators [29, 28] in the 1960s at the Institute for Defense Analysis in Princeton, NJ. In each iteration, it estimates the distribution of the unobserved states and then maximizes the expected log likelihood with respect to that estimate. Although Baum et al. limited their attention to HMMs, the same kind of iteration works on other models that have unobserved variables. In 1977, Dempster, Laird, and Rubin [31] called the general procedure the *EM algorithm*.

The EM algorithm operates on models $P_{\mathbf{Y},\mathbf{S},\theta}$ with parameters θ for a mix of data that is observed ($\mathbf{Y}$) and data that is unobserved ($\mathbf{S}$). (For this application, $\mathbf{Y}$ is a sequence of observations Y_1^T and $\mathbf{S}$ is a sequence of discrete hidden states S_1^T.) The steps in the algorithm are the following:

1. Guess[7] a starting value of $\theta(1)$, and set $n = 1$.
2. Characterize the distribution $P(\mathbf{S}|\mathbf{y}, \theta)$. This is the E step.
3. Choose $\theta(n + 1)$ to maximize the *auxiliary function* Q,

$$\theta(n+1) = \underset{\theta}{\operatorname{argmax}}\, Q(\theta, \theta(n)), \tag{2.18}$$

[7]Although a symmetric model in which the transition probability from each state to every other state is the same and the observation probabilities are all uniform is easy to describe, such a model is a bad choice for $\theta(1)$ because the optimization procedure will not break the symmetry of the states.

where

$$Q(\theta', \theta) \equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)} \left(\log P(\mathbf{y}, \mathbf{S}, \theta') \right). \tag{2.19}$$

(Here the notation $\mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}(F(\mathbf{S}))$ means the expected value of $F(\mathbf{S})$ over all values of $\mathbf{S}$ using the distribution $P(\mathbf{S}|\mathbf{y}, \theta)$.) This is the M step.

4. If not converged, go to 2 with $n \leftarrow n + 1$.

I defer further discussion of the general EM algorithm to Section 2.5 and now proceed to the details of its application to HMMs, i.e., the Baum–Welch algorithm.

2.4.1 Weights and Reestimation

For the Baum–Welch algorithm, steps 2 and 3 of the EM algorithm are the following:

E step: Run the forward algorithm described in Section 2.1 to calculate the values of $\alpha(s, t)$ and $\gamma(t)$ for each time $t \in [1, \ldots, T]$ and each state $s \in \mathcal{S}$, and run the backward algorithm described in Section 2.2 to calculate the values of $\beta(s, t)$.

M step: Reestimate the model parameters using the formulas in Table 2.1.

I write the reestimation formulas in terms of *weights* which express the conditional probability of being in specific states at specific times given the observed data y_1^T. I denote the conditional probability of being in state s at time t given all of the data by

$$w(s, t) \equiv P_{S(t)|Y_1^T}\left(s|y_1^T\right) = \alpha(s, t)\beta(s, t), \tag{2.20}$$

and I denote the conditional probability, given all of the data, of being in state s at time t and being in state $\tilde{s}$ at time $t + 1$ by

$$\tilde{w}(\tilde{s}, s, t) \equiv P_{S(t+1),S(t)|Y_1^T}\left(\tilde{s}, s|y_1^T\right) \tag{2.21}$$

$$= \frac{\alpha(s, t) \cdot P_{S(t+1)|S(t)}\left(\tilde{s}|s\right) \cdot P_{Y(t+1)|S(t+1)}\left(y(t + 1)|\tilde{s}\right) \cdot \beta(\tilde{s}, t + 1)}{\gamma(t + 1)}. \tag{2.22}$$

To derive (2.22), start with

$$P_{S(t+1),S(t)|Y_1^T}\left(\tilde{s}, s|y_1^T\right) = \frac{P(\tilde{s}, s, y_{t+2}^T|y_1^{t+1})}{P(y_{t+2}^T|y_1^{t+1})},$$

in which I have applied Bayes' rule and dropped the subscripts on P. Then the next four lines provide the result

$$P(\tilde{s}, s|y_1^T) = \frac{P(y_{t+2}^T|\tilde{s}, s, y_1^{t+1})P(\tilde{s}, s, |y_1^{t+1})}{P(y_{t+2}^T|y_1^{t+1})} \tag{2.23a}$$

$$= \frac{\beta(\tilde{s}, t + 1)P(y(t + 1), \tilde{s}, s|y_1^{t+1})}{P(y(t + 1)|y_1^t)} \tag{2.23b}$$

$$= \frac{\beta(\tilde{s}, t + 1)}{\gamma(t + 1)} P(y(t + 1)|\tilde{s}, s, y_1^t)P(\tilde{s}|s, y_1^t)P(s|y_1^t) \tag{2.23c}$$

$$= \frac{\beta(\tilde{s}, t + 1)P(y(t + 1)|\tilde{s})P(\tilde{s}|s)\alpha(s, t)}{\gamma(t + 1)} \tag{2.23d}$$

with the following justifications:

Table 2.1. *Summary of reestimation formulas.*

Note that formulas for $w(s,t)$ and $\tilde{w}(\tilde{s},s,t)$ appear in (2.20) and (2.22), respectively.

Description	Expression	New value
Initial state probability	$P_{S(1)\|\theta(n+1)}\,(s\|\theta(n+1))$	$w(s,1)$
State transition probability	$P_{S(t+1)\|S(t),\theta(n+1)}\,(\tilde{s}\|s,\theta(n+1))$	$\dfrac{\sum_{t=1}^{T-1}\tilde{w}(\tilde{s},s,t)}{\sum_{s'\in\mathcal{S}}\sum_{t=1}^{T-1}\tilde{w}(s',s,t)}$
Conditional observation probability	$P_{Y(t)\|S(t),\theta(n+1)}\,(y\|s,\theta(n+1))$	$\dfrac{\sum_{t:y(t)=y} w(s,t)}{\sum_t w(s,t)}$

(2.23a) Bayes' rule.

(2.23b) Apply the definition of β to the combination of the denominator and the first term in the numerator of (2.23a) and then expand the second term in the numerator by Bayes' rule.

(2.23c) Apply the definition of γ to the denominator of (2.23b) and Bayes' rule to the second term in the numerator.

(2.23d) Apply the model assumptions.

Reestimation

With (2.20) and (2.22) for $w(s,t)$ and $\tilde{w}(\tilde{s},s,t)$ in terms of known quantities (α, β, γ, y_1^T, and the old model parameters $\theta(n)$), one can use the formulas in Table 2.1 to calculate new estimates of the model parameters, $\theta(n+1)$, with higher likelihood. Pseudocode for the entire iterative procedure appears in Fig. 2.5.

To derive the formulas in Table 2.1, start with step 3 of the EM algorithm (see (2.19)), which is to maximize the auxiliary function

$$Q(\theta',\theta) \equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{y},\mathbf{S}|\theta')\right)$$

with respect to θ'. For an HMM, substitute the sequence of *hidden* states s_1^T for $\mathbf{S}$ and the sequence of observations y_1^T for $\mathbf{y}$. Note that the joint probability of a state sequence s_1^T and the observation sequence y_1^T is

$$P\left(s_1^T, y_1^T\right) = P_{S(1)}\,(s(1)) \cdot \prod_{t=2}^{T} P_{S(2)|S(1)}\,(s(t)|s(t-1)) \cdot \prod_{t=1}^{T} P_{Y(1)|S(1)}\,(y(t)|s(t))\,,$$

Notation:

$\theta(n)$ is the model, or equivalently the set of parameters, after n iterations of the Baum–Welch algorithm.

$\boldsymbol{\alpha}_n$ is the set of conditional state probabilities calculated on the basis of the nth model and the data y_1^T. See (2.2) and (2.7).

$$\boldsymbol{\alpha}_n \equiv \left\{ P_{S(t)|Y_1^t,\theta(n)} \left(s | y_1^t, \theta(n) \right) : \forall s \in \mathcal{S} \,\&\, 1 \leq t \leq T \right\}$$

$\boldsymbol{\beta}_n$ is a set of values calculated on the basis of the nth model $\theta(n)$ and the data y_1^T. See (2.11) and (2.12).

$$\boldsymbol{\beta}_n \equiv \left\{ \frac{P_{Y_{t+1}^T|S(t)} \left(y_{t+1}^T | s \right)}{P \left(y_{t+1}^T | y_1^t \right)} : \forall s \in \mathcal{S} \,\&\, 1 \leq t < T \right\}$$

$\boldsymbol{\gamma}_n$ is the set of conditional observation probabilities calculated on the basis of the nth model $\theta(n)$ and the data y_1^T. See (2.3) and (2.6).

$$\boldsymbol{\gamma}_n \equiv \left\{ P \left(y(t) | y_1^{t-1}, \theta(n) \right) : \; 2 \leq t \leq T \right\}$$

Initialize:

Set $n = 1$ and choose $\theta(1)$

Iterate:

$(\boldsymbol{\alpha}_n, \boldsymbol{\gamma}_n) \leftarrow \text{forward}(y_1^T, \theta(n))$	See Section 2.1, page 20
$\boldsymbol{\beta}_n \leftarrow \text{backward}(\boldsymbol{\gamma}_n, y_1^T, \theta(n))$	See Section 2.2, page 25
$\theta(n+1) \leftarrow \text{reestimate} \left(y_1^T, \boldsymbol{\alpha}_n, \boldsymbol{\beta}_n, \boldsymbol{\gamma}_n, \theta(n) \right)$	See Table 2.1, page 33
$n \leftarrow n + 1$	
Test for completion	

Figure 2.5. *Summary and pseudocode for optimizing model parameters by iterating the Baum–Welch algorithm.*

or, equivalently,

$$\log P \left(y_1^T, s_1^T \right) = \log P_{S(1)} \left(s(1) \right) + \sum_{t=2}^{T} \log P_{S(2)|S(1)} \left(s(t) | s(t-1) \right) + \sum_{t=1}^{T} \log P_{Y(1)|S(1)} \left(y(t) | s(t) \right).$$

I can optimize Q by breaking it into a sum in which each of the model parameters appears in only one of the terms and then optimizing each of the terms independently:

$$Q(\theta', \theta) = \sum_{s_1^T \in \mathcal{S}^T} \left(P\left(s_1^T | y_1^T, \theta\right) \log P\left(s_1^T, y_1^T | \theta'\right) \right) \tag{2.24}$$

$$\equiv Q_{\text{initial}}(\theta', \theta) + Q_{\text{transition}}(\theta', \theta) + Q_{\text{observation}}(\theta', \theta), \tag{2.25}$$

where

$$Q_{\text{initial}}(\theta', \theta) \equiv \sum_{s_1^T \in \mathcal{S}^T} \left(P\left(s_1^T | y_1^T, \theta\right) \log P_{S(1)|\theta'}\left(s(1)|\theta'\right) \right), \tag{2.26}$$

$$Q_{\text{transition}}(\theta', \theta) \equiv \sum_{s_1^T \in \mathcal{S}^T} \left(P\left(s_1^T | y_1^T, \theta\right) \sum_{t=2}^{T} \log P_{S(2)|S(1),\theta'}\left(s(t)|s(t-1), \theta'\right) \right), \tag{2.27}$$

$$Q_{\text{observation}}(\theta', \theta) \equiv \sum_{s_1^T \in \mathcal{S}^T} \left(P\left(s_1^T | y_1^T, \theta\right) \sum_{t=1}^{T} \log P_{Y(1)|S(1),\theta'}\left(y(t)|s(t), \theta'\right) \right). \tag{2.28}$$

To simplify the appearance of expressions as I optimize Q, I introduce notation for logs of parameters:

$$L_{\text{initial}}(s) \equiv \log P_{S(1)|\theta'}\left(s|\theta'\right), \tag{2.29}$$

$$L_{\text{transition}}(s, \tilde{s}) \equiv \log P_{S(2)|S(1),\theta'}(\tilde{s}|s, \theta'), \tag{2.30}$$

$$L_{\text{observation}}(y, s) \equiv \log P_{Y(1)|S(1),\theta'}\left(y|s, \theta'\right). \tag{2.31}$$

Now to optimize Q_{initial} write (2.26) as

$$Q_{\text{initial}}(\theta', \theta) = \sum_{s_1^T \in \mathcal{S}^T} P\left(s_1^T | y_1^T, \theta\right) L_{\text{initial}}(s(1)) \tag{2.32}$$

$$= \sum_{s \in \mathcal{S}} L_{\text{initial}}(s) \sum_{s_1^T : s(1)=s} P\left(s_1^T | y_1^T, \theta\right) \tag{2.33}$$

$$= \sum_{s} L_{\text{initial}}(s) P_{S(1)|Y_1^T,\theta}\left(s|y_1^T, \theta\right) \quad \text{(see (2.20))} \tag{2.34}$$

$$= \sum_{s} L_{\text{initial}}(s) w(s, 1). \tag{2.35}$$

I wish to find the set $\{L_{\text{initial}}(s)\}$ that maximizes $Q_{\text{initial}}(\theta', \theta)$ subject to the constraint

$$\sum_{s} e^{L_{\text{initial}}(s)} \equiv \sum_{s} P_{S(1)|\hat{\theta}}\left(s|\hat{\theta}\right) = 1.$$

The method of Lagrange multipliers yields[8]

$$L_{\text{initial}}(s) = \log w(s, 1) \quad \forall s; \tag{2.36}$$

i.e., the new estimates of the initial probabilities are

$$P_{S(1)|\theta(n+1)}\,(s|\theta(n+1)) = w(s, 1). \tag{2.37}$$

To derive new estimates of the state transition probabilities, write

$$Q_{\text{transition}}(\theta', \theta) = \sum_{s_1^T \in \mathcal{S}^T} P\left(s_1^T | y_1^T, \theta\right) \sum_{t=2}^{T} L_{\text{transition}}(s(t-1), s(t)) \tag{2.38}$$

$$= \sum_{s,\tilde{s}} L_{\text{transition}}(s, \tilde{s}) \sum_{t=2}^{T} \sum_{\substack{s_1^T : s(t)=\tilde{s}, \\ s(t-1)=s}} P\left(s_1^T | y_1^T, \theta\right) \tag{2.39}$$

$$= \sum_{s,\tilde{s}} L_{\text{transition}}(s, \tilde{s}) \sum_{t=1}^{T-1} \tilde{w}(\tilde{s}, s, t). \tag{2.40}$$

Optimization yields

$$P_{S(t+1)|S(t),\theta(n+1)}\,(\tilde{s}|s, \theta(n+1)) = \frac{\sum_{t=1}^{T-1} \tilde{w}(\tilde{s}, s, t)}{\sum_{s'} \sum_{t=1}^{T-1} \tilde{w}(s', s, t)}. \tag{2.41}$$

Similarly, I derive the new estimates of the conditional observation probabilities from

$$Q_{\text{observation}}(\theta', \theta) = \sum_{s_1^T \in \mathcal{S}^T} P\left(s_1^T | y_1^T, \theta\right) \sum_{t=1}^{T} L_{\text{observation}}(y(t), s(t)) \tag{2.42}$$

$$= \sum_{y \in \mathcal{Y}, s \in \mathcal{S}} L_{\text{observation}}(y, s) \sum_{t : y(t)=y} \sum_{s_1^T : s(t)=s} P\left(s_1^T | y_1^T, \theta\right) \tag{2.43}$$

$$= \sum_{y,s} L_{\text{observation}}(y, s) \sum_{t : y(t)=y} w(s, t). \tag{2.44}$$

Optimization yields

$$P_{Y(t)|S(t),\theta(n+1)}\,(y|s, \theta(n+1)) = \frac{\sum_{t : y(t)=y} w(s, t)}{\sum_{t=1}^{T} w(s, t)}. \tag{2.45}$$

[8] To **maximize** $Q(\vec{L}) \equiv \sum_s L(s)w(s)$ **subject to** $\sum_s e^{L(s)} = 1$, I write a Lagrangian

$$\mathcal{L}(\vec{L}, \lambda) = \sum_s L(s)w(s) - \lambda \sum_s e^{L(s)}$$

and differentiate

$$\frac{\partial \mathcal{L}}{\partial L(s)} = w(s) - \lambda e^{L(s)}.$$

At the maximum the derivative is zero and

$$w(s) = \lambda e^{L(s)} = \lambda P(s).$$

So in the new model for each s, the parameter $P(s)$ is proportional to the term $w(s, 1)$ calculated for the old model, and the proportionality λ is set to ensure the probabilities sum to one, i.e., $P(s) = w(s, 1)$.

2.4.2 Remarks

Training on Multiple Segments

Simple modifications to the code for the Baum–Welch algorithm enable it to train on data that consists of a collection of independent segments $\vec{\mathbf{y}} \equiv \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$, where $\mathbf{y}_k = (y_k)_1^{T_k}$. In particular for each iteration, one should do the following:

- Run the forward and backward algorithms on each segment $\mathbf{y}_k$ to calculate $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$.
- Create $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ by concatenating $\boldsymbol{\alpha}_k$ for all k and $\boldsymbol{\beta}_k$ for all k, respectively.
- Reestimate all model parameters by applying the formulas in Table 2.1 to the concatenated $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\vec{\mathbf{y}}$.
- Modify the reestimated initial state probabilities using

$$P_{S(1)|\theta(m+1)}\,(s|\theta(m+1)) = \frac{1}{n}\sum_{k=1}^{n}\alpha_k(s,1)\beta_k(s,1).$$

Probabilities of the Initial State

Using the procedure of the previous section for a few independent observation sequences with several iterations of the Baum–Welch algorithm produces a model with estimates of the probabilities of the initial states, $P_{S(1)}\,(s)$ for all $s \in \mathcal{S}$, that reflect the characteristics at the beginning of the given sequences. Those estimates are appropriate if all observation sequences come from state sequences that start in a similar fashion. Such models are not stationary. For applications with ergodic state dynamics, I calculate the initial state probability estimates using

$$P_{S(1)(\text{ergodic})}\,(s) = \frac{\sum_t w(s,t)}{\sum_{\tilde{s},t} w(\tilde{s},t)}. \tag{2.46}$$

Disappearing Transitions

For some observation sequences, y_1^T, and initial models, repeated iteration of the Baum–Welch algorithm leads to state transition probabilities that are too small to be represented in double precision. As part of the implementation of the Baum–Welch algorithm, I prune transitions with conditional probabilities of less than 1×10^{-20}. Such pruning

- prevents numerical underflow exceptions;
- simplifies the models;
- makes the algorithms that operate on the models run faster.

Maximizing Likelihood over Unrealistic Classes

I often fit simple HMMs to data that come from systems that have complicated continuous state spaces. For example, I fit models with roughly 10 states to ECGs even though I believe that it would require at least scores of continuous valued parameters to specify the states of the physiological parameters that affect the signal. By fitting unrealistically simple models, I reduce the variance of the parameter estimates at the expense of having parameters that are harder to interpret.

Multiple Maxima

The Baum–Welch algorithm generically converges to a *local* maximum of the likelihood function. For example, I obtained the model used to generate Fig. 1.9 by iterating the Baum–Welch algorithm on an initial model with random parameters. By rerunning the experiment with five different seeds for the random number generator, I obtained the five different results that appear in Fig. 2.6.

Bayesian Estimates Instead of Point Estimates

I do not really believe that the maximum likelihood estimate (MLE) produced by the Baum–Welch algorithm is precisely the *one true answer*. It is what Bayesians call a *point estimate*. Parameter values near the MLE are entirely consistent with the data. A complete Bayesian parameter estimation scheme begins with an a priori distribution P_θ that characterizes knowledge about what values are possible and then uses Bayes' rule to combine that knowledge with observed data y to calculate an a posteriori *distribution* of parameters $P_{\theta|y}$. In the next chapter, I present a variant on the Baum–Welch algorithm that uses a prior distribution on parameter values and produces an estimate that maximizes the a posteriori probability, i.e., a MAP estimate. However, like the MLE, the MAP is a point estimate. It does a poor job of characterizing the set of plausible parameters.

In addition to yielding only a point estimate, the Baum–Welch algorithm converges to *local* maxima. A Bayesian *Markov chain Monte Carlo* approach would address all of these objections at the expense of being even slower. Although others have obtained Bayesian a posteriori parameter distributions for HMMs using *Markov chain Monte Carlo* and *variational Bayes'* procedures (see, for example, [48] and [26]), I will restrict my attention to point estimates from the Baum–Welch algorithm and simple variations.

2.5 The EM Algorithm

In Section 2.4, I described the Baum–Welch algorithm for finding parameters of an HMM that maximize the likelihood, and I noted that the Baum–Welch algorithm is a special case of the EM algorithm. Here I examine the general EM algorithm in more detail. Readers willing to accept the Baum–Welch algorithm without further discussion of or interest in (2.18) should skip this section. Dempster, Laird, and Rubin [31] coined the term *EM algorithm* in 1977. More recent treatments include Redner and Walker [47], McLachlan and Krishnan [12], and Watanabe and Yamaguchi [17]. Although *EM* originally stood for

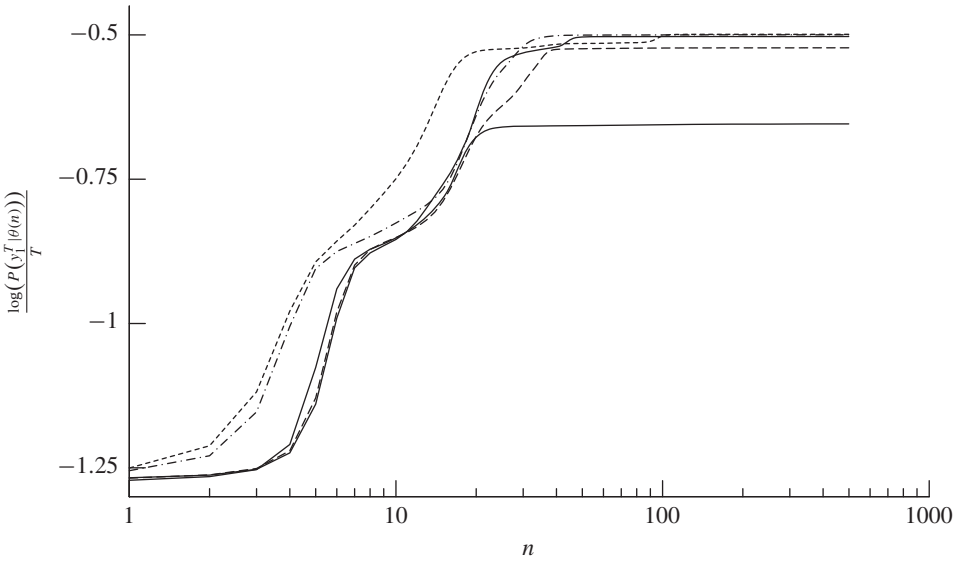


Figure 2.6. *Convergence of the Baum–Welch algorithm. Here I have plotted* $\frac{\log(P(y_1^T|\theta(n)))}{T}$ *(the log likelihood per step) as a function of the number of iterations n of the Baum–Welch algorithm for five different initial models* $\theta(1)$*. I used the same sequence of observations* y_1^T *that I used for Fig.* 1.9*, and I used different seeds for a random number generator to make the five initial models. Note the following characteristics: The five different initial models all converge to different models with different likelihoods; the curves intersect each other, as some models improve more with training than others; convergence is difficult to determine because some curves seem to have converged for many iterations and later rise significantly. Although it appears that three of the initial models all converge to* -0.5*, close examination of the data suggests that they are converging to different models with log likelihoods per step of* -0.49898*,* -0.49975*, and* -0.50275*.*

estimate and *maximize*, it is now thought to stand for *expectation maximization*. Recall that the algorithm operates on models $P_{\mathbf{Y},\mathbf{S},\theta}$ with parameters θ for a mix of data that are observed ($\mathbf{Y}$) and data that are unobserved ($\mathbf{S}$) and that the steps in the algorithm are the following:

1. Guess a starting value of $\theta(1)$, and set $n = 1$.
2. Characterize the distribution $P(\mathbf{S}|\mathbf{y}, \theta)$.
3. Choose $\theta(n+1)$ to maximize the *auxiliary function* Q,

$$\theta(n+1) = \underset{\theta}{\operatorname{argmax}}\, Q(\theta, \theta(n)), \tag{2.47}$$

where

$$Q(\theta', \theta) \equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)} \left(\log P(\mathbf{y}, \mathbf{S}|\theta') \right). \tag{2.48}$$

4. Increment n.

5. If not converged, go to 2.

Steps 2 and 3 do all of the work. Note that if the unobserved data ($\mathbf{S}$) is discrete, then the auxiliary function (Q) is $\mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{y},\mathbf{S}|\theta')\right) = \sum_{\mathbf{s}} P(\mathbf{s}|\mathbf{y},\theta)\left(\log P(\mathbf{y},\mathbf{s}|'_\theta)\right)$. Dempster, Laird, and Rubin [31] called the characterization of $P(\mathbf{S}|\mathbf{y},\theta)$ the *estimation step* and the optimization of $Q(\theta,\theta(n))$ over θ the *maximization step*.

To illustrate the EM algorithm, I generated 10 observations from an i.i.d. Gaussian mixture model

$$P\left(y_1^T|\theta\right) = \prod_{t=1}^{T} \frac{1}{\sqrt{2\pi}}\left(\lambda e^{-\frac{1}{2}(y(t)-\mu_1)^2} + (1-\lambda)e^{-\frac{1}{2}(y(t)-\mu_2)^2}\right) \tag{2.49}$$

with a vector of free parameters $\theta = (\lambda, \mu_1, \mu_2)$ set to $(\frac{1}{2}, -2, 2)$. (See the plot in Fig. 2.7.) Then I tried to recover the parameters from the observations using the EM algorithm. The model generates an observation by

1. selecting one of two states or classes s with probability λ for state $s = 0$ and probability $(1-\lambda)$ for state $s = 1$;

2. drawing from a Gaussian with mean μ_s and variance $\sigma^2 = 1$.

Note that for each observation the class s is unobserved.

As an initial model, I set $\theta(1) = \left(\frac{1}{2}, -1, 1\right)$. Now for each $\theta(n)$ the EM algorithm consists of the following:

E-step. For each observation $y(t)$, *estimate* $w(t)$, the probability that the state was $s = 0$ given the current model parameters and the observation, i.e.,

$$w(t) = P_{s(t)|y(t),\theta(n)}\left(0|y(t),\theta(n)\right). \tag{2.50}$$

M-step. Adjust the parameters to *maximize* the auxiliary function. Letting $W = \sum_{t=0}^{10} w(t)$, some effort verifies that the following assignments maximize $Q(\theta,\theta(n))$:

$$\lambda(n+1) = \frac{1}{10}W, \tag{2.51a}$$

$$\mu_1(n+1) = \frac{1}{W}\sum_{t=0}^{10} y(t)w(t), \tag{2.51b}$$

$$\mu_2(n+1) = \frac{1}{10-W}\sum_{t=0}^{10} y(t)(1-w(t)). \tag{2.51c}$$

The first two iterations of the EM algorithm produce the estimates $\lambda = 0.603$, $\mu_1 = -2.028$, and $\mu_2 = 1.885$. Intermediate calculations and density plots appear in Fig. 2.7. The estimates repeat without change to a precision of 12 decimal digits after a total of 53 iterations, and the final estimate is $\lambda = 0.674601277409$, $\mu_1 = -1.81907548168$, and $\mu_2 = 2.31697397996$.

	t	1	2	3	4	5	6	7	8	9	10
	$y(t)$	−1.02	−3.39	−3.38	2.45	−0.13	−1.66	3.09	2.01	0.03	−2.73
	$w(t)$	0.89	1.00	1.00	0.01	0.56	0.97	0.00	0.02	0.49	1.00
$\theta(1)$	$w(t)y(t)$	−0.91	−3.39	−3.38	0.02	−0.07	−1.61	0.01	0.04	0.01	−2.71
	$(1-w(t))y(t)$	−0.12	−0.00	−0.00	2.44	−0.06	−0.06	3.08	1.98	0.01	−0.01
	$w(t)$	0.98	1.00	1.00	0.00	0.60	1.00	0.00	0.00	0.45	1.00
$\theta(2)$	$w(t)y(t)$	−1.00	−3.39	−3.38	0.00	−0.08	−1.66	0.00	0.00	0.01	−2.73
	$(1-w(t))y(t)$	−0.02	−0.00	−0.00	2.45	−0.05	−0.00	3.09	2.01	0.01	−0.00

$\theta(1) = (0.5, -1, 1)$ $\theta(2) = (0.592, -2.024, 1.779)$ $\theta(3) = (0.603, -2.028, 1.885)$

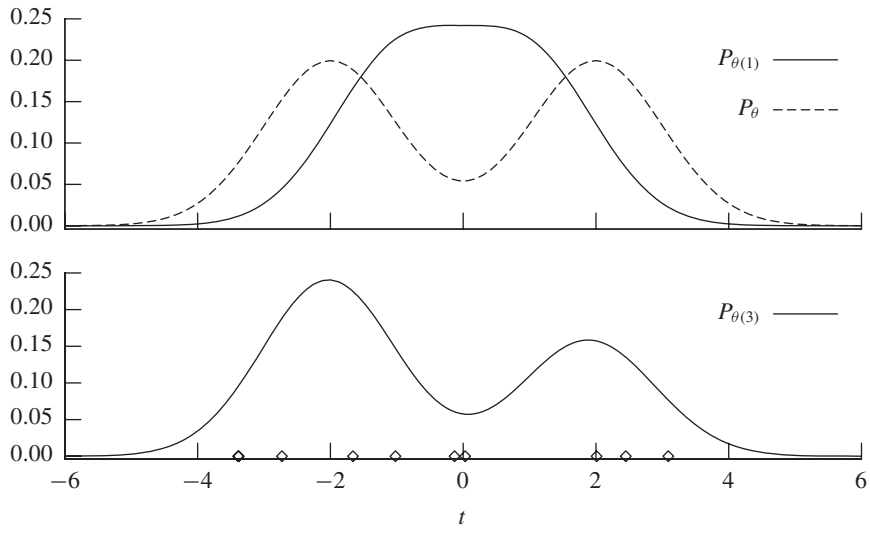


Figure 2.7. *Two iterations of the EM algorithm. I use the algorithm to search for the parameters of* (2.49) *that maximize the likelihood of the* 10 *simulated observations that appear above in the row labeled* $y(t)$*. The triple of rows labeled* $\theta(1)$ *report the weighting calculated using* (2.50) *used in* (2.51) *to recalculate the conditional means,* μ_1 *and* μ_2*, for* $\theta(2)$ *in the M step, and the next triple of rows, labeled* $\theta(2)$*, report the same quantities for the calculation of* $\theta(3)$*. The parameters of the first three models appear in the row just below the boxed table.* $\theta(3)$ *is the triple of parameters produced by two iterations of the EM algorithm,* $\lambda = 0.603$*,* $\mu_1 = -2.028$*,* $\mu_2 = 1.885$*. On the axes of the upper plot, I illustrate* $P(x|\theta)$ *as defined in* (2.49) *for two sets of model parameters: The dashed line depicts* $\theta = (0.5, -2, 2)$*, the distribution used to simulate the data, and the solid line depicts* $\theta(1) = (0.5, -1, 1)$*, the starting distribution I chose for the EM algorithm. On the bottom axes, I plot the simulated observations as marks on the abscissa and* $P(y|\theta(3))$ *as a solid line.*

The advantages of the EM algorithm are that it is easy to implement and it monotonically increases the likelihood. These often outweigh its slow linear convergence and the fact that it calculates neither the second derivative of the likelihood function nor any other indication of the reliability of the results it reports. Proving monotonicity is simple. If the likelihood is bounded, convergence of the likelihood follows directly from monotonicity, but convergence of the parameters does not follow. Also, the likelihood might converge to a local maximum. Papers by Baum et al. [29], Dempster, Laird, and Rubin [31], and Wu [54] analyze the issues. The next two subsections summarize and augment some of that analysis.

2.5.1 Monotonicity

Denoting the log likelihood of the observed data given the model parameters θ as

$$L(\theta) \equiv \log\left(P\left(\mathbf{y}|\theta\right)\right)$$

and the *cross entropy* of the unobserved data with respect to a model θ given a model θ as

$$H(\theta'||\theta) \equiv -\mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{S}|\mathbf{y},\theta')\right),$$

I can write the auxiliary function as

$$\begin{aligned} Q(\theta',\theta) &\equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{S},\mathbf{y}|\theta')\right) \\ &= \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{S}|\mathbf{y},\theta') + \log\left(P(\mathbf{y}|\theta')\right)\right) \\ &= L(\theta') - H(\theta'||\theta) \end{aligned}$$

or

$$L(\theta') = Q(\theta',\theta) + H(\theta'||\theta). \tag{2.52}$$

The fact that

$$H(\theta'||\theta) \geq H(\theta||\theta) \quad \forall\theta' \tag{2.53}$$

with equality *iff*

$$P(\mathbf{s}|\mathbf{y},\theta') = P(\mathbf{s}|\mathbf{y},\theta) \quad \forall\mathbf{s}$$

is called the *Gibbs inequality*.[9] It is a consequence of Jensen's inequality.

Given the model $\theta(n)$ after n iterations of the EM algorithm, I can write

$$L(\theta(n)) = Q(\theta(n),\theta(n)) + H(\theta(n)||\theta(n)). \tag{2.54}$$

If for some other model θ'

$$Q(\theta',\theta(n)) > Q(\theta(n),\theta(n)), \tag{2.55}$$

then the Gibbs inequality implies $H(\theta'||\theta(n)) \geq H(\theta(n)||\theta(n))$ and, consequently, $L(\theta') > L(\theta(n))$. Monotonicity of the log function further implies

$$P\left(\mathbf{y}|\theta'\right) > P\left(\mathbf{y}|\theta(n)\right).$$

Since the EM algorithm requires the inequality in (2.55), for $\theta' = \theta(n+1)$, the algorithm monotonically increases the likelihood.

[9] While most statisticians attribute this inequality to Kullback and Leibler [41], it appears earlier in Chapter XI, *Theorem* II of Gibbs [5].

2.5.2 Convergence

The EM algorithm operates on Θ, a set of allowed parameter vectors with the function $\mathcal{T} : \Theta \mapsto \Theta$ that implements an iteration of the algorithm, i.e.,

$$\theta(n+1) = \mathcal{T}(\theta(n)).$$

Wu [54] has observed that more than one value of θ' may maximize $Q(\theta', \theta)$. Consequently, he considered $\mathcal{T}$ to be a function that maps points to sets. However, there are many model classes—including all of the HMMs that I describe in this book—for which one can write algorithms that calculate a unique $\mathcal{T}(\theta)$. Thus I consider the case of $\mathcal{T}$ being a simple function. If there is a bound $\bar{P}$ on $P(\mathbf{y}|\theta)$ with

$$P(\mathbf{y}|\theta) \leq \bar{P} \quad \forall \theta \in \Theta,$$

the monotonicity of the sequence $(P(\mathbf{y}|\theta(1)), P(\mathbf{y}|\theta(2)), P(\mathbf{y}|\theta(3)), \ldots)$ and the bounded convergence theorem ensure that the limit $P^* \equiv \lim_{n\to\infty} P(\mathbf{y}|\theta(n))$ exists. The bounded convergence theorem does not promise that $P^* = \sup_{\theta\in\Theta} P(\mathbf{y}|\theta)$ or that it is even a local maximum, nor does it promise that $\theta^* \equiv \lim_{n\to\infty} \theta(n)$ exists.

It would be nice if every trajectory of $\mathcal{T}$ converged to a local maximum of the likelihood. Since Wu [54] provides an example of a trajectory that converges to a saddle point of the likelihood, it certainly is not true. In the following, I do, however, give assurance that, in typical circumstances, trajectories converge to local maxima.

I assume that Θ is compact and that the functions $L : \Theta \mapsto \mathbb{R}$, $H : \Theta \times \Theta \mapsto \mathbb{R}$, and $Q : \Theta \times \Theta \mapsto \mathbb{R}$ are C^2; i.e., each function and its first two derivatives exist and are continuous. For any starting point $\theta(1)$, I am interested in the *limit set* $\Omega(\theta(1))$ defined as the set of points to which subsequences converge; i.e., for all $\theta \in \Omega(\theta(1))$, there is a subsequence of integers $(N(1), N(2), \ldots)$ with $N(n+1) > N(n)$ for all n such that $\lim_{n\to\infty} \mathcal{T}^{N(n)}(\theta(1)) = \theta$. If $\mathcal{T}$ maps a compact neighborhood of $\theta(1)$ into itself, then $\Omega(\theta(1)) \neq \emptyset$. For discrete HMMs the assumptions are valid for all points in the interior of Θ. Continuity and the Gibbs inequality ensure that each element of a limit set has the same log likelihood L^* and that the cross entropy between pairs of limit points is a constant H^*. Consequently, the conditional distribution of the unobserved data is constant over all limit points; call it $P^*(\mathbf{S}|\mathbf{y})$. Also the continuity assumption ensures that at each point the first derivative of L is zero. In summary,

$$\forall \theta \in \Omega(\theta(1)) \begin{cases} L(\theta) = L^*, \\ H(\theta||\mathcal{T}(\theta)) = H(\theta||\theta) = H^*, \\ P(\mathbf{s}|\mathbf{y}, \theta) = P^*(\mathbf{s}|\mathbf{y}) \quad \forall \mathbf{s}, \\ \frac{\partial L(\theta)}{\partial \theta} = 0. \end{cases} \tag{2.56}$$

The limit set must be one of the following types: a single point θ^*, a periodic orbit, or an infinite number of points, none of which is isolated. Consider each possibility in turn.

$\Omega = \{\theta^*\}$ Is a Single Point

If N is the dimension of Θ, the Hessian of the log likelihood is the $N \times N$ matrix $\frac{\partial^2 L(\theta)}{\partial \theta^2}|_{\theta^*}$. Of the N eigenvalues of the Hessian, if N_+ are positive, N_z are zero, and N_- are negative, then

the triple (N_+, N_z, N_-) is called the *inertia* of the Hessian. The following three possibilities are exhaustive:

$N_- = N$. This is the desirable case. The first derivative is zero, and the second derivative is negative definite. These are the necessary and sufficient conditions for θ^* to be a local maximum of L. The analysis of $\mathcal{T}$ below shows that such limit points are linearly stable. Thus nonzero volumes of initial conditions in Θ converge to them.

$N_z > 0$. Although this can happen for neighborhoods of values for distributions such as the Laplacian ($P(x) = \frac{\lambda}{2} e^{-|x-\mu|}$), it is not generic[10] for any of the distributions I consider in this book.

$N_+ > 0$. Trajectories that converge to θ^* must stay on the stable manifold. Close to θ^*, if a trajectory has *any* component in the unstable subspace, it will diverge from θ^* exponentially. The analysis of $\mathcal{T}$ below shows that the dimension of the linearly unstable subspace is N_+. Thus if $N+ > 0$, convergence to θ^* is not generic.

The linear stability of $\mathcal{T}$ at an isolated fixed point θ^* is determined by the eigenvalues $\{\lambda_k : 1 \leq k \leq N\}$ of the $N \times N$ matrix $\frac{\partial \mathcal{T}(\theta)}{\partial \theta}|_{\theta^*}$. The fixed point is linearly unstable if any eigenvalue has magnitude greater than one ($|\lambda_k| > 1$). I write[11] the derivative of $\mathcal{T}$ in terms of derivatives of L and Q:

$$\left.\frac{\partial \mathcal{T}(\theta)}{\partial \theta}\right|_{\theta^*} = \mathbf{I} - \left[\left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*}\right]^{-1} \left[\left.\frac{\partial^2 L(\theta)}{\partial \theta^2}\right|_{\theta^*}\right]. \tag{2.58}$$

Note that $\operatorname{argmax}_\theta Q(\theta^*, \theta) = \theta^*$ because θ^* is a fixed point of $\mathcal{T}$, and generically the Hessian $[\frac{\partial^2 Q(\theta,\theta)}{\partial \theta^2}|_{\theta^*,\theta^*}]$ will be negative definite. It follows that the term $A \equiv [-\frac{\partial^2 Q(\theta,\theta)}{\partial \theta^2}|_{\theta^*,\theta^*}]^{-1}$ in (2.58) is positive definite.

[10] By *not generic*, I mean an event that occurs only on a set of Lebesgue measure zero.

[11] Equation (2.58), which appears in the original EM paper by Dempster et al., can be derived by considering the Taylor series expansion of $\frac{\partial Q(\theta,\theta)}{\partial \theta}$ about θ^*:

$$\left.\frac{\partial Q(\theta', \theta)}{\partial \theta'}\right|_{\theta_a, \theta_b} = \left.\frac{\partial Q(\theta', \theta)}{\partial \theta'}\right|_{\theta^*, \theta^*} + \left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*} (\theta_b - \theta^*) + \left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*} (\theta_a - \theta^*) + R,$$

where R is a remainder. Substituting $\theta(n)$ for θ_b and $\mathcal{T}(\theta(n))$ for θ_a, I find that to first order

$$0 = 0 + \left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*} (\theta(n) - \theta^*) + \left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*} (\mathcal{T}(\theta(n)) - \theta^*)$$

and

$$\left.\frac{\partial \mathcal{T}(\theta)}{\partial \theta}\right|_{\theta^*} = \left[-\left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*}\right]^{-1} \left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*}. \tag{2.57}$$

Direct differentiation of the definition of H yields

$$\left.\frac{\partial^2 H(\theta'||\theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*} = -\left.\frac{\partial^2 H(\theta'||\theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*}.$$

Combining this with $Q(\theta', \theta) = L(\theta) - H(\theta'||\theta)$, I find that

$$\left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*} = -\left.\frac{\partial^2 H(\theta'||\theta)}{\partial \theta' \partial \theta}\right|_{\theta^*, \theta^*} = \left.\frac{\partial^2 H(\theta'||\theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*} = \frac{\partial^2}{\partial \theta'^2} \left[L(\theta') - Q(\theta', \theta)\right]_{\theta^*, \theta^*},$$

and substituting back into (2.57) yields (2.58).

If A and B are symmetric and A is positive definite, then one can use Sylvester's inertia theorem[12] to show that the inertia of AB is equal to the inertia of B. At a saddle point of L, $B = [\frac{\partial^2 L(\theta)}{\partial \theta^2}|_{\theta^*}]$ will have at least one positive eigenvalue, and the product AB will also have at least one positive eigenvalue, λ^+. For the eigenvector v^+ of AB corresponding to λ^+,

$$(\mathbf{I} + AB)v^+ = (1 + \lambda^+)v^+.$$

Since $(1 + \lambda^+)$ is greater than one and is an eigenvalue of $\frac{\partial \mathcal{T}(\theta)}{\partial \theta}|_{\theta^*}$, the saddle is an unstable fixed point of $\mathcal{T}$, and convergence to it is not generic.

Similarly, if θ^* is an isolated local maximum of L with negative definite $B = [\frac{\partial^2 L(\theta)}{\partial \theta^2}|_{\theta^*}]$, then AB is also negative definite, and each eigenvalue of $\mathbf{I} + AB$ is less than one. Negative definiteness of B combined with the monotonicity of the EM algorithm also implies that every eigenvalue is greater than -1. Thus $|\lambda| < 1$ for each eigenvalue λ of $\frac{\partial \mathcal{T}(\theta)}{\partial \theta}|_{\theta^*}$, and the maximum is a linearly stable fixed point of $\mathcal{T}$. Linear stability implies that for every $\theta(1)$ in some neighborhood of θ^* the sequence $\big(\theta(1), \mathcal{T}(\theta(1)), \mathcal{T}^2(\theta(1)), \ldots\big)$ converges to θ^*.

In summary, convergence to a fixed point of $\mathcal{T}$ that is a local maximum of the likelihood is generic, and convergence to a saddle point of the likelihood is not generic.

Ω Is a Periodic Orbit

At a periodic limit point θ^* with period n, an analysis parallel to the derivation of (2.58) yields

$$\left.\frac{\partial \mathcal{T}^n(\theta)}{\partial \theta}\right|_{\theta^*} = \mathbf{I} - \left[\left.\frac{\partial^2 Q(\theta', \theta)}{\partial \theta'^2}\right|_{\theta^*, \theta^*}\right]^{-1} \left[\left.\frac{\partial^2 L(\theta)}{\partial \theta^2}\right|_{\theta^*}\right],$$

and I conclude by the same arguments as for the period one case that convergence to periodic limit points is generic if they are local maxima of the likelihood and not generic if they are saddle points of the likelihood.

Ω Has an Infinite Number of Points

I have not analyzed the possibility of a limit set with an infinite number of points, none of which is isolated. To detect such behavior, one could check numerical trajectories of $\mathcal{T}$ for convergence in both θ and L.

[12] Gerardo Lafferriere suggested the analysis using Sylvester's inertia theorem, which says that for any symmetric matrix M and invertible X both M and $X^\top MX$ have the same inertia. Because A^{-1} is positive definite, it can be factored as $A^{-1} = XX^\top$. For this factorization, $A = (X^\top)^{-1}X^{-1}$, and

$$X^\top ABX = X^{-1}BX. \tag{2.59}$$

The right-hand side of (2.59) is a similarity transformation which preserves the inertia of B, while the left-hand side has the inertia of AB. Thus the inertia of AB is the same as the inertia of B.

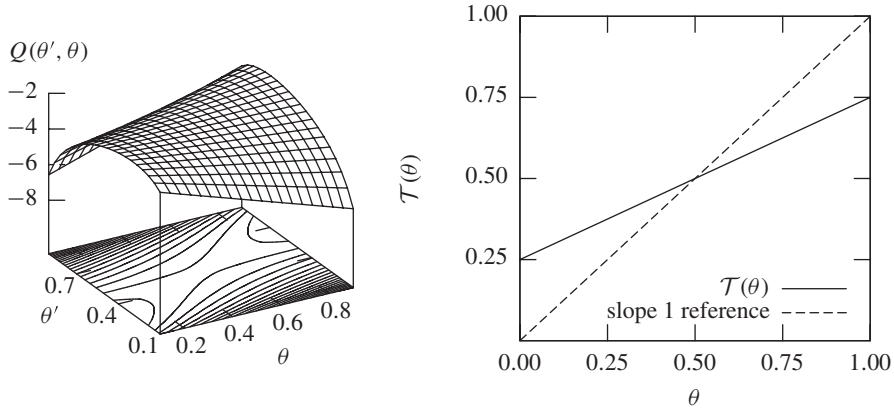


Figure 2.8. *An illustration of the EM algorithm for an experiment in which a coin is thrown four times; first, a head is observed* ($y(1) = 1$)*, then a tail is observed* ($y(2) = 0$)*, and finally two results are unobserved with s_h and s_t being the number of unobserved heads and tails, respectively. The goal is to find the maximum likelihood value of θ, the probability of heads. The log-likelihood function for the complete data is $L_\theta = (s_h + 1)\log(\theta) + (s_t + 1)\log(1-\theta)$. The auxiliary function $Q(\theta', \theta) = (1+2\theta)\log(\theta') + (1+2(1-\theta))\log(1-\theta')$ appears on the left, and the map $\mathcal{T}(\theta)$ appears on the right. Note that $\theta^* = \frac{1}{2}$ is the fixed point of $\mathcal{T}$ (where the plot intersects the slope* 1 *reference line) and it is stable because the slope of $\mathcal{T}$ is less than one.*

A Contrived Example

Fig. 2.8 illustrates stable convergence for the case of a coin being tossed four times and observed twice. By pretending that the unobserved data is important and handling it via the EM algorithm, I find that the algorithm implements an obviously stable *linear* map $\mathcal{T}$.

Chapter 3
Variants and Generalizations

HMMs are special cases of discrete-time state space models characterized by a state transition probability function and an observation probability function, i.e.,

$$P(S_{n+1}|S_n), \tag{3.1a}$$
$$P(Y_n|S_n). \tag{3.1b}$$

In Chapter 2, I described algorithms for fitting and using the probability distributions specified in (3.1) if both the set of possible states $\mathcal{S}$ and the set of possible observations $\mathcal{Y}$ have an unstructured finite number of discrete values. However, in many applications the measurements, and perhaps the states also, are thought of as being drawn from continuous vector spaces.

Since most experimental observations are measured and recorded digitally, one could argue that discrete approximations are adequate and attempt to use the algorithms of Chapter 2 anyway. That approach is disastrous because it precludes exploiting either the metric or topological properties of the space of measurements. Consider the histogram of the first 600 samples of Tang's laser data in Fig. 3.1. Neither 5 nor 93 occurs, but it seems more plausible that 93 will occur in the remainder of the samples because there are 14 occurrences between 90 and 96 and none between 2 and 8. To make more effective use of measured data, one usually approximates the probabilities by functions with a small number of free parameters. For many such families of *parametric models* one can use the algorithms of Chapter 2 with minor modifications.[13] For a practitioner, the challenge is to find or develop both a parametric family that closely matches the measured system and algorithms for fitting and using the models.

In this chapter, I will describe some model families with Gaussian observations. I will use the failure of the maximum likelihood approach with such models to motivate and develop *regularization*. Also, I will touch on the relationships between HMM model families and other kinds of models.

[13] At the 1988 ICASSP meeting, Poritz [21] reviewed several HMM variants.

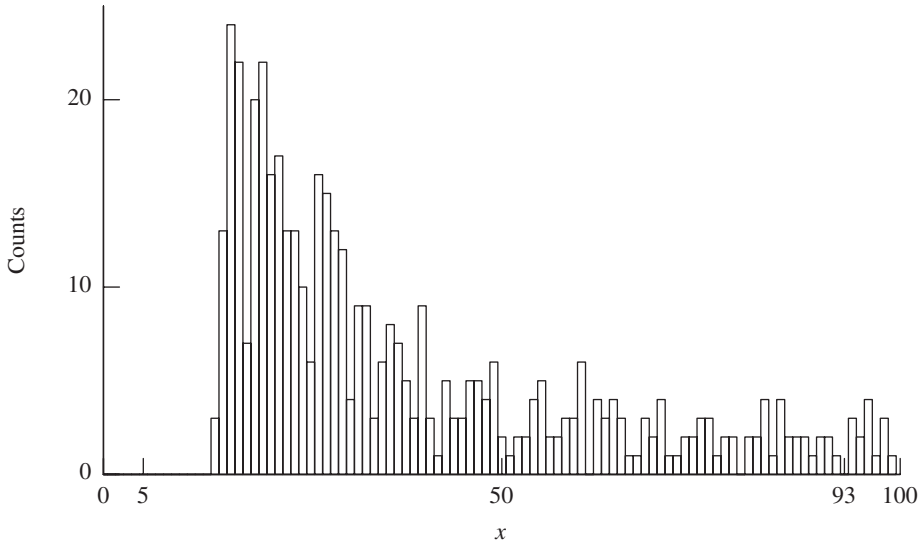


Figure 3.1. *Histogram of Tang's laser measurements. Even though neither* $y = 5$ *nor* $y = 93$ *occurs in* y_1^{600}*, it is more plausible that* $y = 93$ *would occur in future measurements because of what happens in the neighborhood. Discarding the numerical significance of the bin labels would preclude such an observation.*

3.1 Gaussian Observations

3.1.1 Independent Scalar Observations

A simple model for continuously distributed measurements is an HMM with an independent scalar Gaussian observation model associated with each state. In many cases it is adequate, but risky (see Fig. 3.3), to simply use the algorithms of Chapter 2 with minor modifications for Gaussian observations. Such algorithms performed satisfactorily for the exercises depicted in Fig. 3.2 in which I estimated an approximate state sequence and model parameters from a sequence of observations.

The code that generated the data for Fig. 3.2 implemented algorithms from Chapter 2 with the following modifications:

$P_{Y(t)|S(t)}(y|s)$. The Viterbi algorithm, the forward algorithm, and the backward algorithm all use the observation probability conditioned on the state. In each case one simply uses the value of the probability density conditioned on the state

$$P_{Y(t)|S(t)}(y|s) = \frac{1}{\sqrt{2\pi\sigma_s^2}} e^{-\frac{(y-\mu_s)^2}{2\sigma_s^2}}.$$

Reestimation. Reviewing the derivations in Section 2.4.1, one finds that the first two formulas in Table 2.1 (those for the initial state probability, $P_{S(1)|\theta(n+1)}(s|\theta(n+1))$, and the state transition probability, $P_{S(t+1)|S(t),\theta(n+1)}(\tilde{s}|s)$) still work with the new observation model. To derive reestimation formulas for the Gaussian observation model parameters, note that (2.31) becomes

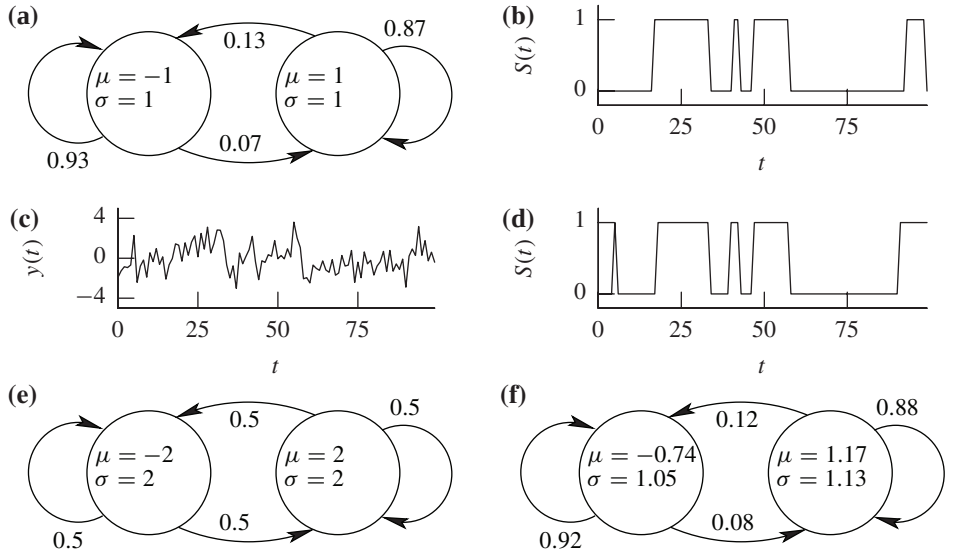


Figure 3.2. *An HMM with scalar Gaussian observations. A state diagram appears in* (a)*. The half-life of the first state is about* 10 *and the half-life of the second state is about five, i.e.,* $0.93^{10} \approx 0.87^5 \approx 0.5$*. A simulated state sequence and observation sequence appear in* (b) *and* (c)*, respectively. Using the model parameters from* (a) *and the observation sequence from* (c)*, the Viterbi algorithm estimates the state sequence that appears in* (d)*, which is satisfyingly similar to the state sequence in* (b)*. Finally, starting from the initial model depicted in* (e) *and using the observation sequence depicted in* (c)*,* 50 *iterations of the Baum–Welch algorithm produce the model depicted in* (f)*, which is satisfyingly similar to* (a)*.*

$$L_{\text{observation}}(y, s) \equiv \log P_{Y(1)|S(1),\theta'}\left(y|s, \theta'\right) \tag{3.2}$$

$$= -\frac{1}{2}\log(2\pi) - \log(\sigma_s) - \frac{(y-\mu_s)^2}{2\sigma_s^2} \tag{3.3}$$

and starting from (2.42) calculate

$$Q_{\text{observation}}(\theta', \theta) = \sum_{q_1^T \in \mathcal{S}^T} P_\theta\left(q_1^T | y_1^T\right) \sum_{t=1}^{T} L_{\text{observation}}(y(t), q(t)) \tag{3.4}$$

$$= \sum_{s\in\mathcal{S}} \sum_{t=1}^{T} L_{\text{observation}}(y(t), s) \sum_{q_1^T : q(t)=s} P_\theta\left(q_1^T | y_1^T\right) \tag{3.5}$$

$$= -\sum_{s\in\mathcal{S}} \sum_{t=1}^{T} w(s,t)\left(\frac{1}{2}\log(2\pi) + \log(\sigma_s) + \frac{(y(t)-\mu_s)^2}{2\sigma_s^2}\right). \tag{3.6}$$

Since the formulas

$$\mu_s = \sum_{t=1}^{T} w(s,t)y(t), \tag{3.7}$$

$$\sigma_s^2 = \sum_{t=1}^{T} w(s,t)\,(y(t)-\mu_s)^2 \tag{3.8}$$

maximize $Q_{\text{observation}}(\theta', \theta)$, I use them in place of the discrete observation reestimation formula of Chapter 2 (Table 2.1 and (2.45)).

3.1.2 Singularities of the Likelihood Function and Regularization

Running five iterations of the Baum–Welch algorithm on the observation sequence in Fig. 3.2(c) starting with the model in Fig. 3.3(a) produces the model in Fig. 3.3(b) in which the variance of the observations produced by the second state looks suspiciously small. In fact with additional iterations of the Baum–Welch algorithm that variance continues to shrink, and after the seventh iteration, the code stops with a floating point exception. The algorithm is pursuing a singularity in the likelihood function in which the second state fits the 56th observation exactly and the first state fits all of the other observations. If $\mu_2 = y(56)$, the likelihood $P_{Y(t)|S(t)}\,(y(56)|2)$ increases without limit as $\sigma_2 \to 0$, i.e.,

$$\lim_{\sigma_2 \to 0} P_{Y(t)|S(t)}\,(y(56)|2) = \infty.$$

Such singularities of likelihood are common among parametric probability density functions. A particularly simple example is the *Gaussian mixture model*

$$f(y) = \lambda \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(y-\mu_1)^2}{2\sigma_1^2}} + (1-\lambda)\frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(y-\mu_2)^2}{2\sigma_2^2}}, \tag{3.9}$$

which has the five parameters $\mu_1,\ \sigma_1,\ \mu_2,\ \sigma_2$, and λ. Assuming that the data are i.i.d., one might attempt a maximum likelihood fit to the observations in Fig. 3.2(e) with the likelihood function

$$g(\mu_1, \sigma_1, \mu_2, \sigma_2, \lambda) = \prod_{t=1}^{T} f(y(t)). \tag{3.10}$$

While it is possible to find a useful *local* maximum of g near

$$\mu_1 = -1,\ \ \sigma_1 = 1,\ \ \mu_2 = 1,\ \ \sigma_2 = 1,\ \ \lambda = \frac{2}{3},$$

the likelihood is higher near the singularities of g specified by the equations

$$\mu_s = y(t),$$
$$\sigma_s = 0$$

for each pair $(s,t) \in \{1,2\} \times \{1,2,\ldots,T\}$.

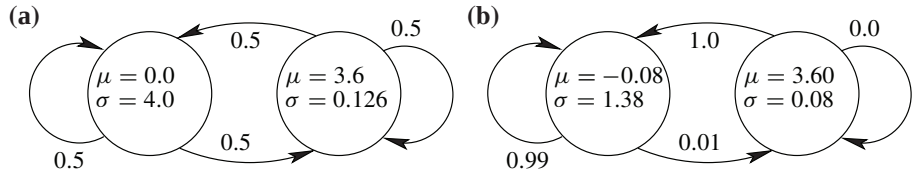


Figure 3.3. *An illustration of the trouble with maximum likelihood. Here I have used the same implementation of the Baum–Welch algorithm that I used to produce Fig.* 3.2(f)*, but rather than starting with the model in Fig.* 3.2(c)*, I started the algorithm with the initial model depicted in* (a) *above. Five iterations of the algorithm produced the suspicious model depicted in* (b) *above.*

If, as is the case here, I want to exclude parameter vectors for which the likelihood function is larger than its value at the solution I prefer, then likelihood does not really express my goal. *Regularization* refers to a variation on maximum likelihood that more accurately reflects what I want. In the next subsection, I explain how to use Bayesian *priors* to regularize MAP parameter estimates.

3.1.3 The EM Algorithm for MAP Estimation

Bayesian estimation starts by characterizing the acceptability of models $P_{\mathbf{Y}|\theta}$ in terms of an a priori probability distribution P_θ. Initial observations $\mathbf{y}_e$, called *evidence* or training data, modify the prior through Bayes' rule to yield the a posteriori distribution

$$P(\theta|\mathbf{y}_e) = \frac{P(\mathbf{y}_e, \theta)}{P(\mathbf{y}_e)} = \frac{P(\mathbf{y}_e|\theta)P(\theta)}{\int P(\mathbf{y}_e|\theta)P(\theta)\,d\theta}, \tag{3.11}$$

and the probability of future observations $\mathbf{y}_f$ is

$$P(\mathbf{y}_f|\mathbf{y}_e) = \int P(\mathbf{y}_f|\theta)P(\theta|\mathbf{y}_e)\,d\theta. \tag{3.12}$$

The parameter vector that maximizes the a posteriori distribution,

$$\theta_{\text{MAP}} \equiv \underset{\theta}{\text{argmax}}\, P(\theta|\mathbf{y}_e) \tag{3.13a}$$

$$= \underset{\theta}{\text{argmax}}\, P(\theta, \mathbf{y}_e), \tag{3.13b}$$

is called the MAP estimate. Using θ_{MAP} one may approximate[14] (3.12) with $P\left(\mathbf{y}_f|\theta_{\text{MAP}}\right)$.

A slight variation of the algebra in Section 2.5 produces an EM algorithm for MAP estimation. Dropping the subscript on $\mathbf{y}_e$, if I replace the auxiliary function of (2.19), i.e.,

$$Q_{\text{MLE}}(\theta', \theta) \equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)}\left(\log P(\mathbf{S}, \mathbf{y}|\theta')\right),$$

[14]Although it is every bit as reasonable to use the mean to characterize the a posteriori distribution as it is to use the maximum, I prefer the maximum because changing from MLE to MAP requires only minor modifications to the Baum–Welch algorithm. A strictly Bayesian approach would retain the entire a posteriori distribution in parameter space rather than characterizing the distribution by a single point estimate.

with

$$Q_{\text{MAP}}(\theta', \theta) \equiv \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)} \left(\log P(\mathbf{S}, \mathbf{y}, \theta')\right) \tag{3.14a}$$
$$= Q_{\text{MLE}}(\theta', \theta) + \mathbb{E}_{P(\mathbf{S}|\mathbf{y},\theta)} \left(\log P(\theta')\right) \tag{3.14b}$$
$$= Q_{\text{MLE}}(\theta', \theta) + \log P(\theta'), \tag{3.14c}$$

then the derivation of

$$Q_{\text{MAP}}(\theta', \theta) > Q_{\text{MAP}}(\theta, \theta) \Rightarrow P(\mathbf{y}, \theta') > P(\mathbf{y}, \theta)$$

is completely parallel to the argument on page 42 that concludes $Q_{\text{MLE}}(\theta', \theta) > Q_{\text{MLE}}(\theta, \theta)$. If in addition the components of $P(\theta)$ are independent, i.e.,

$$P(\theta) = P(\theta_{\text{initial}}) \cdot P(\theta_{\text{transition}}) \cdot P(\theta_{\text{observation}}),$$

then like the decomposition of Q_{MLE} in (2.25) I find that

$$Q_{\text{MAP}}(\theta', \theta) = Q_{\text{MAP, initial}}(\theta', \theta) + Q_{\text{MAP, transition}}(\theta', \theta) + Q_{\text{MAP, observation}}(\theta', \theta),$$

with

$$Q_{\text{MAP, initial}} = Q_{\text{MLE, initial}} + \log P(\theta_{\text{initial}}), \tag{3.15a}$$
$$Q_{\text{MAP, transition}} = Q_{\text{MLE, transition}} + \log P(\theta_{\text{transition}}), \tag{3.15b}$$
$$Q_{\text{MAP, observation}} = Q_{\text{MLE, observation}} + \log P(\theta_{\text{observation}}). \tag{3.15c}$$

With a suitable prior, the simple form of (3.15) makes it easy to convert a program that implements the Baum–Welch algorithm for maximum likelihood estimation into a program that implements MAP estimation.

3.1.4 Vector Autoregressive Observations

Overview of the Model

Rather than choosing a prior and developing algorithms for the independent scalar observations of Section 3.1.1, I will work on more general models with vector autoregressive Gaussian observations associated with each hidden state. If at time t such a system is in state s, then the mean of the conditional distribution for $y(t)$ is a linear function of the d previous observations y_{t-d}^{t-1} added to a fixed offset. Specifically, the conditional distributions for observations are n-dimensional vector Gaussians as follows:

Covariance. The covariance is a state dependent symmetric positive definite $n \times n$ matrix Σ_s.

Mean. Given that the system is in state s, the parameters $\left\{c_{s,i,\tau,j}, \bar{y}_{s,i}\right\}$ and the d previous observations determine the mean of the observation distribution through an *affine* function, i.e., the sum of a linear function and a fixed offset, with components

$$\mu\left(s, y_{t-d}^{t-1}\right) = \bar{y}_{s,i} + \sum_{\tau=1}^{d} \sum_{j=1}^{n} c_{s,i,\tau,j} y_j(t-\tau). \tag{3.16}$$

I implement this as a *linear* function of a *context vector* x consisting of d previous observations and a constant one, i.e.,

$$\begin{aligned} x(t) &\equiv \left(\overrightarrow{y(t-1)}, \overrightarrow{y(t-2)}, \ldots \overrightarrow{y(t-d)}, 1\right) \\ &\equiv (y_1(t-1), y_2(t-1), \ldots, y_n(t-1), y_1(t-2), \ldots, y_n(t-d), 1)\,. \end{aligned}$$

Using notation in which the ith component of the observation τ time steps before time t is $y_i(t-\tau)$, the kth component of the context at time t is

$$x_k(t) = \begin{cases} y_i(t-\tau), & 1 \le \tau \le d, \\ 1, & \tau = d+1, \quad i = 1, \end{cases} \tag{3.17}$$

where $k = d \cdot (\tau - 1) + i$, and

$$\mu_s\left(s, y_{t-d}^{t-1}\right) = A_s x(t), \tag{3.18}$$

where A_s is an $n \times (nd+1)$ matrix consisting of $\left\{c_{s,i,\tau,j}\right\}$ and $\left\{\bar{y}_{s,i}\right\}$.

Using this notation the model assumptions are (compare to (1.18) and (1.19), which describe the assumptions for HMMs with discrete observations) that the states follow a Markov process and that the conditional distribution of an observation given the state s is Gaussian with mean $A_s x(t)$ and covariance Σ_s, i.e.,

$$P(s(t+1)|s_{-\infty}^t, y_{-\infty}^t) = P\left(s(t+1)|s(t)\right), \tag{3.19}$$

$$P(y(t)|s_{-\infty}^t, y_{-\infty}^{t-1}) = \frac{1}{\sqrt{(2\pi)^n \left|\Sigma_{s(t)}\right|}} \exp\left(-\frac{1}{2} z_{s(t)}^\top(t) \Sigma_{s(t)}^{-1} z_{s(t)}(t)\right), \tag{3.20}$$

where

$$z_k(t) \equiv y(t) - A_k x(t).$$

The model accounts for relationships between an observation and its predecessors two ways; first, the observation at time $y(t)$ is related to the d previous observations through the matrix A_s, and it is also related to *all* previous observations through the state s.

Reestimation

A derivation like the one leading to (3.6) yields

$$Q_{\text{observation}}(\theta', \theta) = \frac{1}{2} \sum_{s \in \mathcal{S}} \sum_{t=1}^{T} w(s,t) \left[\log\left(\left|\Sigma_s^{-1}\right|\right) - n\log(2\pi) - z_s^\top(t) \Sigma_s^{-1} z_s(t)\right]. \tag{3.21}$$

Hence each term in the sum over s can be optimized separately. One can write code that maximizes $\sum_{t=1}^{T} w(s,t) \left[\log\left|\Sigma_s^{-1}\right| - z_s^\top(t) \Sigma_s^{-1} z_s(t)\right]$ using operations on vectors and matrices as follows:

1. Create a weighted *context* matrix X_s with columns $x(t)\sqrt{w(s,t)}$, where $w(s,t) = P_{S(t)|y_1^T}\left(s|y_1^T\right)$ and (3.17) defines $x(t)$.

2. Create a weighted *observation* matrix Y_s with columns $y(t)\sqrt{w(s,t)}$.

3. Solve

$$A_s = \underset{M}{\operatorname{argmin}} |Y_s - MX_s|^2 . \tag{3.22}$$

   (I use singular value decomposition methods here because they are stable and make diagnosing problems easy.)

4. Calculate a matrix of residuals

$$Z_s = Y_s - A_s X_s .$$

5. Calculate a new covariance matrix[15]

$$\Sigma_s = \frac{\sum_{t=1}^{T} w(s,t) z_s(t) z_s^\top(t)}{\sum_{t=1}^{T} w(s,t)} \tag{3.23}$$

   using $Z_s Z_s^\top = \sum_{t=1}^{T} w(s,t) z_s(t) z_s^\top(t)$.

Regularization

As (3.15) indicates, I can influence the a posteriori distributions of the initial states, the transitions, and the observations by the selection of the respective priors, $P(\theta_{\text{initial}})$, $P(\theta_{\text{transition}})$, and $P(\theta_{\text{observation}})$. Since singularities in the likelihood are associated with singular covariance matrices Σ_s, the prior for the covariance matrices is most urgent.

Following [34, 43], I use inverse Wishart distributions as priors for the inverse covariance matrices. See pages 150–155 of Schafer [15] for a description of these distributions. The inverse Wishart prior has the following probability density for an $n \times n$ inverse covariance matrix:

$$P_{\text{IW}}(\Sigma^{-1}) \equiv C \left|\Sigma^{-1}\right|^{\frac{m+n+1}{2}} e^{-\frac{1}{2}\mathbf{tr}(\Lambda\Sigma^{-1})},$$

[15] One may derive this formula by differentiating the right-hand side of (3.21) with respect to the elements of Σ_s^{-1} and setting the result equal to zero. The key is the observation that, for a symmetric positive definite matrix M,

$$\frac{\partial \log|M|}{\partial m_{i,j}} = \left[M^{-1}\right]_{i,j};$$

i.e., the derivative of the log of the determinant with respect to the i,jth element of M is the i,jth element of M^{-1}. Since the value of A_s that minimizes $\sum_{t=1}^{T} w(s,t) - z_s^\top(t)\Sigma_s^{-1} z_s(t)$ is independent of Σ_s^{-1}, it is correct to do step 3 before step 5.

where C is a normalization constant, m is called the *degrees of freedom*, and Λ is called the *scale*. The mean and the maximum of the inverse Wishart are

$$\mathbb{E}\Sigma^{-1} = \frac{1}{m-n-1}\Lambda^{-1}$$

and

$$\text{argmax } P_{\text{IW}}\left(\Sigma^{-1}\right) = \frac{1}{m+n+1}\Lambda^{-1},$$

respectively.

Assuming neutral priors for the coefficient matrices A_s and the form

$$\Lambda = \beta\mathbf{I},$$

one finds that

$$P(\theta_y) \propto \prod_s \left|\Sigma_s^{-1}\right|^{\frac{\alpha}{2}} e^{-\frac{1}{2}\mathbf{tr}(\beta\Sigma_s^{-1})}$$

(where $\alpha = n + m + 1$), and (3.15c) becomes

$$Q_{\text{observation}} = C + \sum_s \left(\frac{\alpha}{2}\log\left|\Sigma_s^{-1}\right| - \frac{1}{2}\mathbf{tr}(\beta\Sigma_s^{-1})\right) + \frac{1}{2}\sum_s\sum_{t=1}^{T} w(s,t)\left[\log\left|\Sigma_s^{-1}\right| - z_s^\top(t)\Sigma_s^{-1}z_s(t)\right],$$

where $z_s(t) \equiv y(t) - A_s x(t)$ and $w(s,t)$ is the probability of being in state s at time t given the data y_1^T. Reestimation of the regularized model is the same as reestimation for the unregularized model, except that (3.23) is replaced with

$$\Sigma_s = \frac{\beta\mathbf{I} + \sum_{t=1}^{T} w(s,t)z_s(t)z_s^\top(t)}{\alpha + \sum_{t=1}^{T} w(s,t)}. \tag{3.24}$$

Thus in the absence of new information, a covariance matrix is given the default value $\frac{\beta\mathbf{I}}{\alpha}$, and α is the weight of that default.

3.2 Related Models

Fig. 3.4 illustrates the strength of the HMMs with the vector autoregressive observations described in Section 3.1.4, namely, that optimization finds regions over which dynamics are approximately linear and assigns those regions to single states. The model class used to make the figure does not exploit all of the available features of the Lorenz data. In particular it does not use the Lorenz equations. Modeling techniques exist that can exploit knowledge of the nonlinear generating equations, e.g., *extended Kalman filters* and *particle filters*, that I mention below.

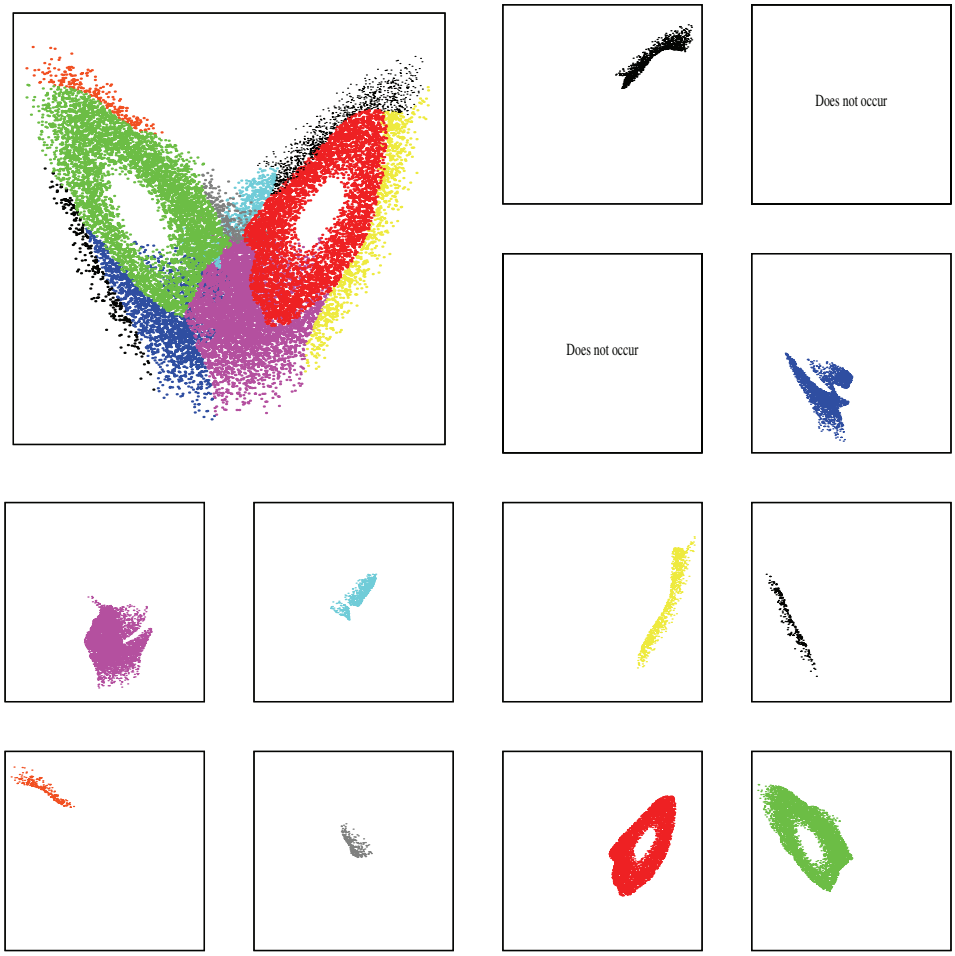


Figure 3.4. *Plots of decoded states using an HMM with vector autoregressive observations. Here the observations are a trajectory of three-dimensional state vectors from the Lorenz system. In each state the observation $y(t)$ is modeled as a Gaussian with a mean that is an affine (linear plus fixed offset) function of the observation $y(t-1)$. The empty boxes correspond to states that do not appear in the decoded sequence of states. In comparing with Fig.* 1.9*, which used a model with coarsely quantized observations, notice that large regions near the fixed points at the centers of the spirals are represented by a single state. These large regions occur because the dynamics are approximately linear over their extents.*

The following list briefly mentions a few of the many techniques that are related to the basic ideas I have described.

Nonstationary HMMs. The definition of a *stationary* model is that all probabilities are independent of shifts in time, i.e., $P_{Y_1^t} = P_{Y_{1+\tau}^{t+\tau}}$ for all (t, τ). Many processes, for example weather, are not even approximately stationary. By making HMMs with state transition probabilities that depend on time, many have created nonstationary HMMs for such applications. With Rechtsteiner in [46] I have implemented weather models with transition probabilities that depend on the time of year. In similar work, Hughes and Guttorp [38] described a local precipitation model with state transition probabilities driven by large scale atmospheric parameters.

Gaussian mixtures. In a Gaussian mixture, the probability density is a weighted average of Gaussian densities. I described a simple one-dimensional two component example in (3.9). Although a simple Gaussian mixture model does not use the notion of *state*, they are frequently used as observation models in HMMs. Note that a Gaussian mixture model is equivalent to the degenerate case of an HMM with Gaussian observations in which all state transition probabilities are identical.

Cluster weighted modeling. For an HMM, the state transition probabilities depend only on the state. Looking at Fig. 3.4, it seems that letting the current observation $y(t)$ influence the transition probabilities could be an improvement; i.e., the probability of the next state would depend on both the current state and the current observation. By ignoring the current state entirely, one obtains a *cluster weighted model* [35]; the observation at time $t + 1$ is a mixture whose component weights are determined by the observation at time t.

Kalman filter. The Kalman filter is essentially the forward algorithm applied to a model with continuous states $x(t)$ and observations $y(t)$ with the form

$$\begin{aligned} x(t+1) &= F(x(t), t) + \eta(t), \\ y(t) &= G(x(t)) + \epsilon(t), \end{aligned}$$

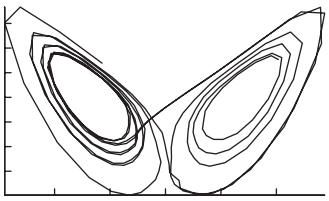
where the functions F and G are linear in x and the noise terms $\eta(t)$ and $\epsilon(t)$ are Gaussian. I will describe algorithms that operate with such models in the next chapter.

Extended Kalman filter. Using linear approximations to nonlinear functions F and G in a Kalman filter is called an *extended Kalman filter*. Linear approximations work well as long as the covariance of the state distributions are small compared to the second derivatives of F and G.

Unscented Kalman filter. Rather than use linear approximations to F and G, an unscented Kalman filter [39] uses exact functions and a collection of samples to estimate means and variances.

Particle filter. The idea of using the empirical distribution of many simulated trajectories (particles) to approximate the conditional distribution $P_{X(t)|Y_1^t}$ is called particle filtering [36, 40]. In the procedure, particles that seem inconsistent with measurements are eliminated, and new particles are created.

Chapter 4



Continuous States and Observations and Kalman Filtering

I often think of state space systems with continuously distributed states and observations in terms of equations such as

$$x(t) = F(x(t-1), t) + \eta(t), \tag{4.1a}$$
$$y(t) = G(x(t), t) + \epsilon(t), \tag{4.1b}$$

where $X \in \mathbb{R}^n$ is the state variable, $Y \in \mathbb{R}^m$ is the observation, and $\eta(t)$ and $\epsilon(t)$ are noise terms. Equations (4.1) define the conditional probability densities

$$P_{X(t+1)|X(t)} \tag{4.2}$$

and

$$P_{Y(t)|X(t)}. \tag{4.3}$$

Having each of the noise terms $\eta(t)$ and $\epsilon(t)$ in (4.1) be independent of all other noise terms is sufficient to ensure that the following assumptions hold:

1. Given the state at time t, the observation $Y(t)$ is independent of everything else.
2. The dynamics of the state variable X are Markov.

These are the same as the assumptions of (1.19) and (1.18), which characterize HMMs with discrete observations. In this chapter, I write forward and backward algorithms by replacing sums of finite probabilities with integrals over probability densities. If the functions F and G are linear in X and the noise terms $\eta(t)$ and $\epsilon(t)$ are independent and Gaussian, then the forward algorithm is called *Kalman filtering*.

In this chapter, I go over the forward and backward algorithms for continuous states and observations three times. First, in Section 4.1, I emphasize the parallel to Chapter 2 by simply replacing sums in the development of that chapter with integrals. By themselves, the results are not immediately useful because one must specify parametric forms for the probability densities and *do* the integrals to implement the algorithms. Next, in Section 4.2, I concisely present the algorithms one obtains when the functions F and G in (4.1) are linear and the noise terms $\eta(t)$ and $\epsilon(t)$ are Gaussian. I hope this concise presentation will

be useful for readers who want to implement the algorithms. Finally, for completeness, in Section 4.3, I demonstrate that in fact the integrals of Section 4.1 do yield the formulas of Section 4.2.

4.1 Algorithms with Integrals

4.1.1 Forward Algorithm

As the forward algorithm uses each successive observation, it calculates two quantities. First, it calculates the conditional probability of the observation at time t given earlier observations:

$$\gamma(t+1) \equiv P\left(y(t+1)|y_1^t\right).$$

Then it calculates the conditional distribution of states given the observations up to the present time:

$$\alpha(x,t) \equiv P_{X(t)|Y_1^t}\left(x|y_1^t\right).$$

The equations

$$P_{X(1),Y(1)}\left(x,\, y(1)\right) = P_{Y(1)|X(1)}\left(y(1)|x\right) P_{X(1)}\left(x\right), \tag{4.4}$$

$$\gamma(1) = P\left(y(1)\right) = \int P_{X(1),Y(1)}\left(x,\, y(1)\right) dx, \tag{4.5}$$

$$\alpha(x,\,1) \equiv P_{X(1)|Y(1)}\left(x|y(1)\right) = \frac{P_{X(1),Y(1)}\left(x,\, y(1)\right)}{P\left(y(1)\right)} \tag{4.6}$$

specify the calculation of $\gamma(1)$, the probability of the first observation, and of $\alpha(x,\,1)$, the conditional distribution of the first state given the first observation, that initialize the algorithm. Given $\gamma(1)$ and $\alpha(x,\,1)$, the forward algorithm executes the following recursion to calculate $\gamma(t)$ and $\alpha(x,\,t)$ for each of the remaining observations $y(t)$.

Forecast the state distribution. Find the conditional distribution of the state at the next time given the observations up to the present time:

$$P\left(x(t+1),\, x(t)|y_1^t\right) = P\left(x(t+1)|x(t),\, y_1^t\right) P\left(x(t)|y_1^t\right) \tag{4.7}$$

$$= P\left(x(t+1)|x(t)\right) P\left(x(t)|y_1^t\right) \tag{4.8}$$

$$= P\left(x(t+1)|x(t)\right) \alpha(x(t),\,t), \tag{4.9}$$

$$P\left(x(t+1)|y_1^t\right) = \int P\left(x(t+1),\, x(t)|y_1^t\right) dx(t) \tag{4.10}$$

$$= \int P\left(x(t+1)|x(t)\right) \alpha(x(t),\,t) dx(t). \tag{4.11}$$

Here I justify (4.7) by Bayes' rule, (4.8) by the model assumptions, and (4.10) by the definition of a marginal distribution.

Calculate the joint forecast. Leaving the state at the next time as a free parameter but fixing the next observation at $y(t)$, calculate the joint conditional distribution of the next state and observation given the observations up to the present time:

$$P\left(x(t+1), y(t+1)|y_1^t\right) = P\left(y(t+1)|x(t+1), y_1^t\right) P\left(x(t+1)|y_1^t\right) \tag{4.12}$$

$$= P\left(y(t+1)|x(t+1)\right) P\left(x(t+1)|y_1^t\right). \tag{4.13}$$

Here I justify (4.12) by Bayes' rule and (4.13) by the model assumptions.

Calculate the conditional probability of the observation. Integrate out $x(t+1)$ to get the conditional probability of the next observation given the observations up to the present time:

$$\gamma(t+1) \equiv P\left(y(t+1)|y_1^t\right) = \int P\left(x(t+1), y(t+1)|y_1^t\right) dx(t+1). \tag{4.14}$$

Update the conditional state distribution. Use Bayes' rule to combine (4.13) and (4.14) and get the conditional distribution of the state at time $t+1$ given all of the measurements up to time $t+1$:

$$\alpha(x(t+1), t+1) = P\left(x(t+1)|y_1^{t+1}\right) \tag{4.15}$$

$$= \frac{P\left(x(t+1), y(t+1)|y_1^t\right)}{P\left(y(t+1)|y_1^t\right)} \tag{4.16}$$

$$= \frac{P\left(x(t+1), y(t+1)|y_1^t\right)}{\gamma(t+1)}. \tag{4.17}$$

4.1.2 Backward Algorithm

Similarly, the backward algorithm finds a function called the backward forecast:

$$\beta(x,t) \equiv \frac{P_{Y_{t+1}^T|X(t)}\left(y_{t+1}^T|x\right)}{P\left(y_{t+1}^T|y_1^t\right)} = \frac{P_{X(t)|Y_1^T}\left(x|y_1^T\right)}{P_{X(t)|Y_1^t}\left(x|y_1^t\right)} \tag{4.18}$$

for each time t by starting with $\beta(x,T) = 1$ and following with the recursion

$$\beta(x,t-1) = \int \frac{\beta(x',t) P_{Y(t)|X(t)}\left(y(t)|x'\right) P_{X(t)|X(t-1)}\left(x'|x\right)}{P\left(y(t)|y_1^{t-1}\right)} dx'. \tag{4.19}$$

To justify (4.19) note that

$$\beta(x',t) = \frac{P_{Y_{t+1}^T|X(t)}\left(y_{t+1}^T|x'\right)}{P\left(y_{t+1}^T|y_1^t\right)} \tag{4.20}$$

$$= \frac{P_{Y_{t+1}^T|X(t),Y_1^t}\left(y_{t+1}^T|x', y_1^t\right) P\left(y(t)|y_1^{t-1}\right)}{P\left(y_t^T|y_1^{t-1}\right)} \tag{4.21}$$

and

$$\begin{aligned} &P_{Y(t)|X(t)}\left(y(t)|x'\right) P_{X(t)|X(t-1)}\left(x'|x\right) \\ &\quad = P_{Y(t)|X(t),Y_1^{t-1}}\left(y(t)|x', y_1^{t-1}\right) P_{X(t)|X(t-1),Y_1^{t-1}}\left(x'|x, y_1^{t-1}\right) \qquad (4.22) \\ &\quad = P_{Y(t),X(t)|X(t-1),Y_1^{t-1}}\left(y(t), x'|x, y_1^{t-1}\right) \qquad (4.23) \end{aligned}$$

and consider the integrand of (4.19):

$$\begin{aligned} I &= \frac{\beta(x',t) P_{Y(t)|X(t)}\left(y(t)|x'\right) P_{X(t)|X(t-1)}\left(x'|x\right)}{P\left(y(t)|y_1^{t-1}\right)} \qquad (4.24) \\ &= \frac{P_{Y_{t+1}^T|X(t),Y_1^t}\left(y_{t+1}^T|x', y_1^t\right)}{P\left(y_t^T|y_1^{t-1}\right)} P_{Y(t),X(t)|X(t-1),Y_1^{t-1}}\left(y(t), x'|x, y_1^{t-1}\right) \qquad (4.25) \\ &= \frac{P_{Y_t^T,X(t)|X(t-1)}\left(y_t^T, x'|x\right)}{P\left(y_t^T|y_1^{t-1}\right)}. \qquad (4.26) \end{aligned}$$

Integrating out x' leaves $\frac{P_{Y_t^T|X(t-1)}\left(y_t^T|x\right)}{P\left(y_t^T|y_1^{t-1}\right)} = \beta(x, t-1)$.

As in Chapter 2, I have chosen to define $\beta(x,t)$ so that the conditional distribution of the state at time t given all of the observations is

$$\alpha(x,t)\beta(x,t) = P_{X(t)|Y_1^T}\left(x|y_1^T\right). \qquad (4.27)$$

Defining and evaluating a *backward update* term

$$\begin{aligned} b(x,t) &\equiv \frac{\beta(x,t) P_{Y(t)|X(t)}\left(y(t)|x\right)}{P\left(y(t)|y_1^{t-1}\right)} \qquad (4.28) \\ &= \frac{P_{X(t)|Y_1^T}\left(x|y_1^T\right)}{P_{X(t)|Y_1^{t-1}}\left(x|y_1^{t-1}\right)} \\ &= \frac{P_{Y_t^T|X(t)}\left(y_t^T|x\right)}{P_{Y_t^T|Y_1^{t-1}}\left(y_t^T|y_1^{t-1}\right)} \end{aligned}$$

as an intermediate step can make evaluating the integral in (4.19) easier.

4.2 Linear Gaussian Systems

If the functions F and G in (4.1) are linear in x, the noise terms $\eta(t)$ and $\epsilon(t)$ are i.i.d. and Gaussian[16] with $\eta(t) \sim \mathcal{N}(0, \Sigma_\eta)$ and $\epsilon(t) \sim \mathcal{N}(0, \Sigma_\epsilon)$, and the distribution of the initial state $X(1)$ is also Gaussian, then I can write

$$\begin{aligned} X(1) &\sim \mathcal{N}\left(\mu_{X(1)}, \Sigma_{X(1)}\right), && (4.29a) \\ x(t) &= F(t)\cdot x(t-1) + \eta(t), \qquad \eta(t) \sim \mathcal{N}\left(0, \Sigma_\eta\right), && (4.29b) \\ y(t) &= G(t)\cdot x(t) + \epsilon(t), \qquad \epsilon(t) \sim \mathcal{N}\left(0, \Sigma_\epsilon\right), && (4.29c) \end{aligned}$$

[16] I assume *identical* distributions over time only to simplify the notation. The procedures generalize easily to time dependent noise terms.

where $F(t)$ and $G(t)$ are matrices. I use the notation $\mathcal{N}(\mu, \Sigma)$ for a Gaussian distribution with mean μ and covariance Σ, and I denote the value of a Gaussian probability density at v by $\mathcal{N}(\mu, \Sigma)|_v$; i.e., if V is an n-dimensional random variable $V \sim \mathcal{N}(\mu, \Sigma)$, then $P(v) = \mathcal{N}(\mu, \Sigma)|_v \equiv \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(v-\mu)^\top \Sigma^{-1}(v-\mu)}$.

In this section, I describe the forward and backward algorithms for linear Gaussian systems. I will give the key formulas in terms of the quantities listed below and defer the derivation of the formulas to Section 4.3. My notation emphasizes the similarity to the calculations for discrete processes in Chapter 2. In particular the Greek subscripts α and β denote the forward updated distribution and the backward forecast distribution, respectively, while the Roman subscripts a and b denote the forward forecast distribution and the backward updated distributions, respectively. The assumptions of (4.29) imply that the distributions described by these parameters are also Gaussian.

$\mu_\alpha(t)$ and $\Sigma_\alpha(t)$ (note *Greek* subscript) are the parameters of the *updated* state distribution,[17] i.e.,

$$P\left(x(t)|y_1^t\right) = \mathcal{N}(\mu_\alpha(t), \Sigma_\alpha(t))|_{x(t)}.$$

$\mu_a(t)$ and $\Sigma_a(t)$ (note *Roman* subscript) are the parameters of the one-step *forecast* of the state distribution,[18] i.e.,

$$P\left(x(t)|y_1^{t-1}\right) = \mathcal{N}(\mu_a(t), \Sigma_a(t))|_{x(t)}.$$

$\mu_\gamma(t)$ and $\Sigma_\gamma(t)$ are the parameters of the conditional probability of the observation at time t given all previous observations, i.e.,

$$P\left(y(t)|y_1^{t-1}\right) = \mathcal{N}\left(\mu_\gamma(t), \Sigma_\gamma(t)\right)\Big|_{y(t)}.$$

Neither the forward nor the backward Kalman filter uses the γ terms, but they are useful for calculating the likelihood of model parameters.

$N_\beta(t)$, $\mu_\beta(t)$, and $\Sigma_\beta(t)$ are the parameters[19] of the backward forecast function $\beta(x, t)$, which as a ratio of Gaussian functions itself is an unnormalized Gaussian. I define $N_\beta(t)$ by

$$\frac{\beta(x,t)}{N_\beta(t)} = \mathcal{N}\left(\mu_\beta(t), \Sigma_\beta(t)\right)\Big|_x.$$

$\mu_b(t)$ and $\Sigma_b(t)$ are the mean[20] and covariance of the backward update function $b(x, t)$ (see (4.28)), which is an intermediate term in the backward algorithm. Notice that the parameters of the forward forecast have Roman subscripts, while the parameters of the backward forecast have Greek subscripts.

$\mu_{\alpha\beta}(t)$ and $\Sigma_{\alpha\beta}(t)$ are the mean and covariance of the best estimate of the state at time t given all of the observations, i.e., $P_{X(t)|Y_1^T}\left(x|y_1^T\right) = \mathcal{N}\left(\mu_{\alpha\beta}(t), \Sigma_{\alpha\beta}(t)\right)\Big|_x$.

[17] For the quantities that I call $\mu_\alpha(t)$ and $\Sigma_\alpha(t)$, Maybeck [11] uses the notation $\hat{x}(t_i^+)$ and $P(t_i^+)$ and Kailath, Sayed, and Hassibi [7] use $\hat{x}_{i|i}$ and $P_{i|i}$, respectively.

[18] For the quantities that I call $\mu_a(t)$ and $\Sigma_a(t)$, Maybeck [11] uses the notation $\hat{x}(t_i^-)$ and $P(t_i^-)$ and Kailath, Sayed, and Hassibi [7] use $\hat{x}_{i+1}$ and P_{i+1}, respectively.

[19] On page 342, Kailath, Sayed, and Hassibi [7] use $\hat{x}_i^b$ to denote the quantity that I call $\mu_\beta(t)$.

[20] Kailath, Sayed, and Hassibi [7] use $\hat{x}_{i|i}^b$ to denote the quantity that I call $\mu_b(t)$.

4.2.1 Kalman Filter: The Forward Algorithm

The following two step recursion is called Kalman filtering,[21] and it implements the forward algorithm:

Calculate the forecast of the distribution of the state:

$$\mu_a(t) = F(t) \cdot \mu_\alpha(t-1), \tag{4.30a}$$

$$\Sigma_a(t) = F(t) \cdot \Sigma_\alpha(t-1) \cdot (F(t))^\top + \Sigma_\eta. \tag{4.30b}$$

Update the distribution of the current state using $y(t)$:

$$(\Sigma_\alpha(t))^{-1} = (\Sigma_a(t))^{-1} + (G(t))^\top \Sigma_\epsilon^{-1} G(t), \tag{4.31a}$$

$$\mu_\alpha(t) = \mu_a(t) + \Sigma_\alpha(t) (G(t))^\top \Sigma_\epsilon^{-1} \big[y(t) - G(t)\mu_a(t)\big]. \tag{4.31b}$$

Note the following:

- Equations (4.31) are usually presented in the equivalent but computationally more efficient form

$$\mu_\alpha(t) = \mu_a(t) + K(t) \left(y(t) - G(t)\mu_a(t)\right),$$
$$\Sigma_\alpha(t) = \left(\mathbf{I} - K(t)G(t)\right) \Sigma_a(t),$$

  where

$$K(t) \equiv \Sigma_a(t) (G(t))^\top \left(G(t)\Sigma_a(t) (G(t))^\top + \Sigma_\epsilon\right)^{-1}$$

  is called *the Kalman gain matrix* (see (4.44)). This, the *Kalman form*, is more efficient than (4.31a) because (4.31a) requires inverting a matrix whose size is the dimension of the state space, while the Kalman form requires inverting a matrix whose size is the, often *much* smaller, dimension of the observation space.

- One can calculate the log likelihood of a model by summing the increments

$$\log\left(P\left(y_1^T\right)\right) = \sum_{t=1}^{T} \log\left(P\left(y(t)|y_1^{t-1}\right)\right)$$

  and calculating each increment as

$$P\left(y(t)|y_1^{t-1}\right) = \mathcal{N}\left(\mu_\gamma(t), \Sigma_\gamma(t)\right)\big|_{y(t)},$$

  where

$$\mu_\gamma(t) = G(t)\mu_a(t),$$
$$\Sigma_\gamma(t) = G(t)\Sigma_a(t) (G(t))^\top + \Sigma_\epsilon,$$
$$\log\left(P\left(y(t)|y_1^{t-1}\right)\right) = -\frac{1}{2}\Big(n\log(2\pi) + \log\left(\left|\Sigma_\gamma(t)\right|\right) + \left(y(t) - \mu_\gamma(t)\right)^\top \left(\Sigma_\gamma(t)\right)^{-1} \left(y(t) - \mu_\gamma(t)\right)\Big).$$

[21] There are many longer presentations of Kalman filters, e.g., Maybeck [11], Kailath, Sayed, and Hassibi [7], O.L.R. Jacobs, *Introduction to Control Theory,* Oxford University Press, 1993, and R.G. Brown and P.Y.C. Hwang, *Introduction to Random Signals and Applied Kalman Filters*, Wiley, 1992.

4.2.2 The Backward Algorithm

One calculates $\mu_\beta(t)$ and $\Sigma_\beta(t)$ as defined in (4.18) with the following recursion that goes through the observations in reverse order.

Update the distribution of the current state using $y(t)$:

$$(\Sigma_b(t))^{-1} = \left(\Sigma_\beta(t)\right)^{-1} + (G(t))^\top (\Sigma_\epsilon)^{-1} G(t), \tag{4.32a}$$

$$\mu_b(t) = \mu_\beta(t) + \Sigma_b(t) (G(t))^\top \Sigma_\epsilon^{-1}\big[y(t) - G(t)\mu_\beta(t)\big]. \tag{4.32b}$$

Forecast of the distribution of the state backwards in time:

$$\mu_\beta(t-1) = (F(t))^{-1} \mu_b(t), \tag{4.33a}$$

$$\Sigma_\beta(t-1) = (F(t))^{-1} \left(\Sigma_\eta + \Sigma_b(t)\right) \left((F(t))^{-1}\right)^\top. \tag{4.33b}$$

Note the following:

- As for the forward recursion, the update formulas are usually presented in a computationally more efficient form using a gain matrix (see (4.45)).
- Ideally one would initialize the backward algorithm with

$$\left(\Sigma_\beta(T)\right)^{-1} = 0,$$
$$\mu_\beta(T) = 0,$$

  but that would make $(\Sigma_b(T))^{-1}$ noninvertible and preclude using (4.33b) to evaluate $\Sigma_\beta(T-1)$. One can address the problem by initializing $\Sigma_\beta^{-1}(T)$ with small values or by using the *inverse covariance form* of the algorithm (see Section 4.3.4).

4.2.3 Smoothing

The conditional distribution of the state at time t, given all of the data, is Gaussian and therefore specified by its covariance and mean, i.e.,

$$P\left(x(t)|y_1^T\right) = \mathcal{N}\left(\mu_{\alpha\beta}(t), \Sigma_{\alpha\beta}(t)\right)\Big|_{x(t)}, \tag{4.34}$$

where

$$\left(\Sigma_{\alpha\beta}(t)\right)^{-1} = (\Sigma_\alpha(t))^{-1} + \left(\Sigma_\beta(t)\right)^{-1} \tag{4.35a}$$

and

$$\mu_{\alpha\beta}(t) = \Sigma_{\alpha\beta}(t) \left((\Sigma_\alpha(t))^{-1} \mu_\alpha(t) + \left(\Sigma_\beta(t)\right)^{-1} \mu_\beta(t)\right) \tag{4.35b}$$

are combinations of the *forward update* parameters and the *backward prediction* parameters. Such use of all of the observations to estimate the state sequence is called *smoothing*. Note that combining forward update parameters and backward update parameters, i.e., α and b, for smoothing is an error.

Viterbi Smoothing

Given a sequence of observations, the a posteriori distribution of the state sequences, $P(x_1^T|y_1^T)$, is Gaussian and has a marginal distribution at each time given by (4.34). The maximum of a Gaussian is its mean. Since the coordinates of the maximum of a Gaussian are simply the maxima of the marginals of the coordinates, the MAP estimate of the state sequence is provided by the means of (4.34).

Thus, rather than imitating the forward and backward algorithms of Chapter 2 to estimate a smoothed state sequence, one can implement the generalization of Viterbi decoding described in Section 2.3.1. The forward pass through the observations for that algorithm is implemented exactly by Kalman filtering. Rather than make a backward pass through the data, the backtrack phase of the algorithm uses the derived functions $B(s', t)$ defined in Section 2.3.1.

The results of the two smoothing approaches are the same because for linear Gaussian systems the MAP state sequence is the same as the sequence of MAP states. This is a striking contrast with the example of Section 2.3.2 in which the sequence of MAP states was impossible.

4.3 Algorithm Derivations and Details

This section connects the integrals that describe the forward and backward algorithms ((4.4)–(4.28)) to the Kalman formulas ((4.30)–(4.35)).

4.3.1 Forward Kalman Filter

The basic loop consists of steps 2 through 5 below. In the first iteration, use the model parameters that describe $P(x(1))$ and proceed directly to step 3.

1. **Use $P(x(1))$.** Initialize the recursion using the mean $\mu_a(1)$ and covariance $\Sigma_a(1)$ of $P(x(1))$, which are parameters of the model, and entering the loop at step 3 to calculate $\mu_\gamma(1)$ and $\Sigma_\gamma(1)$ by setting $t = 1$.

2. **Calculate the state forecast, $P\left(x(t)|y_1^{t-1}\right)$.** The distribution of the forecast is Gaussian and therefore determined by its mean and covariance. Thus rather than evaluating the integral in (4.11) to find the distribution $P\left(x(t)|y_1^{t-1}\right)$, one can specify the distribution completely by calculating its first two moments. Since

$$x(t) = F(t)x(t-1) + \eta(t)$$

the mean is

$$\begin{aligned}
\mu_a(t) &= \mathbb{E}_{P\left(x(t-1),\eta(t)|y_1^{t-1}\right)}\left[F(t)X(t-1) + \eta(t)\right] \\
&= \mathbb{E}_{P\left(x(t-1),\eta(t)|y_1^{t-1}\right)}F(t)X(t-1) + \mathbb{E}_{P\left(x(t-1),\eta(t)|y_1^{t-1}\right)}\eta(t) \\
&= \mathbb{E}_{P\left(x(t-1)|y_1^{t-1}\right)}F(t)X(t-1) + \mathbb{E}_{P(\eta)}\eta(t) \qquad (4.36) \\
&= F(t)\mu_\alpha(t-1).
\end{aligned}$$

The key step in the above sequence is (4.36), which I justify by observing that the distribution of the noise $\eta(t)$ is independent of time and independent of $x(\tau)$ and $y(\tau)$ for all earlier times τ. Similarly one calculates the variance by

$$\begin{aligned}\Sigma_a(t) &= \mathbb{E}_{P\left(x(t-1),\eta(t)|y_1^{t-1}\right)}\big[\,(F(t)X(t-1)+\eta(t)-\mu_a(t-1)) \\ &\quad\times (F(t)X(t-1)+\eta(t)-\mu_a(t-1))^\top\,\big] \qquad (4.37)\\ &= F(t)\Sigma_\alpha(t-1)F(t)^\top+\Sigma_\eta. \qquad (4.38)\end{aligned}$$

Thus (4.30) implements the integral of (4.11).

3. **Calculate $P\left(y(t)|y_1^{t-1}\right)$.** Using

$$y(t) = G(t-1)x(t)+\epsilon(t-1)$$

calculate the mean and covariance of the distribution of the forecast observation as follows:

$$\begin{aligned}\mu_\gamma(t) &= \mathbb{E}_{P\left(x(t),\epsilon(t)|y_1^{t-1}\right)}G(t)X(t)+\epsilon(t)\\ &= G(t)\mu_a(t), \qquad (4.39)\\ \Sigma_\gamma(t) &= \mathbb{E}_{P\left(x(t),\epsilon(t)|y_1^{t-1}\right)}\big(G(t)X(t)+\epsilon(t)-\mu_\gamma(t)\big)\\ &\quad\big(G(t-1)X(t)+\epsilon(t)-\mu_\gamma(t)\big)^\top\\ &= G(t)\Sigma_a(t)G(t)^\top+\Sigma_\epsilon. \qquad (4.40)\end{aligned}$$

4. **Calculate $P\left(x(t),\,y(t)|y_1^{t-1}\right)$.** The conditional distribution of the joint variable

$$z(t)=\begin{bmatrix}x(t)\\ y(t)\end{bmatrix}$$

is a Gaussian characterized by

$$\mu_z=\begin{bmatrix}\mu_a(t)\\ \mu_\gamma(t)\end{bmatrix}$$

and[22]

$$\Sigma_z \equiv \begin{bmatrix}A & C\\ C^\top & B\end{bmatrix} = \begin{bmatrix}\Sigma_a(t) & \Sigma_a(t)(G(t))^\top\\ G(t)\Sigma_a(t) & \Sigma_\gamma(t)\end{bmatrix}. \qquad (4.41)$$

One finds the off diagonal terms of Σ_z in (4.41) as follows:

$$\begin{aligned}C &= \mathbb{E}_{P\left(x(t),\epsilon(t)|y_1^{t-1}\right)}\big(x(t)-\mu_a(t)\big)\big(y(t)-\mu_\gamma(t)\big)^\top\\ &= \mathbb{E}_{P\left(x(t)|y_1^{t-1}\right)}\big(x(t)-\mu_a(t)\big)\big(x(t)-\mu_a(t)\big)^\top(G(t))^\top\\ &\quad+\mathbb{E}_{P\left(x(t),\epsilon(t)|y_1^{t-1}\right)}\big(x(t)-\mu_a(t)\big)\big(\epsilon(t)\big)\\ &= \Sigma_a(t)\big(G(t)\big)^\top. \qquad (4.42)\end{aligned}$$

[22] I have introduced the blocks A, B, and C to match the matrix identities in (A.5), which are written in terms of

$$\begin{bmatrix}A & C\\ C^\top & B\end{bmatrix}^{-1} \equiv \begin{bmatrix}D & F\\ F^\top & E\end{bmatrix}.$$

5. **Calculate the state update, $P\left(x(t)|y_1^t\right)$.** From the actual value of $y(t)$ and the joint distribution of $x(t)$ and $y(t)$ given y_1^{t-1}, use (A.7) to write the parameters of $P\left(x(t)|y_1^t\right)$ as

$$\Sigma_\alpha(t) = D^{-1}, \tag{4.43a}$$
$$\mu_\alpha(t) = \mu_a(t) + CB^{-1}\left(y(t) - \mu_\gamma(t)\right). \tag{4.43b}$$

Note the following facts and justifications:

$$\begin{aligned}
E^{-1} &= B - C^\top A^{-1} C && \text{(see (A.5b))}\\
&= \Sigma_\gamma(t) - G(t)\Sigma_a(t)\left(G(t)\right)^\top && \text{(see (4.41))}\\
&= \Sigma_\epsilon && \text{(see (4.40)),}\\
CB^{-1} &= D^{-1}A^{-1}CE && \text{(see (A.5c)}\\
&= \Sigma_\alpha(t)\left(G(t)\right)^\top E && \text{(see (4.42))}\\
&= \Sigma_\alpha(t)\left(G(t)\right)^\top (\Sigma_\epsilon)^{-1},\\
D &= A^{-1} + A^{-1}CEC^\top A^{-1} && \text{(see (A.5a)).}
\end{aligned}$$

Thus

$$\begin{aligned}
(\Sigma_\alpha(t))^{-1} &= (\Sigma_a(t))^{-1} + (G(t))^\top \Sigma_\epsilon^{-1} G(t),\\
\mu_\alpha(t) &= \mu_a(t) + \Sigma_\alpha(t)\left(G(t)\right)^\top \Sigma_\epsilon^{-1}\left[y(t) - G(t)\mu_a(t)\right],
\end{aligned}$$

which are the update equations in (4.31).

As presented in (4.30) and (4.31), the forward algorithm requires two matrix inversions per iteration. First, one must invert $(\Sigma_\alpha(t-1))^{-1}$ for (4.30b); then one must invert $\Sigma_a(t-1)$ for (4.31a). Each of these is an $n \times n$ matrix, where n is the dimension of the state x. One can avoid these inversions either by keeping track of state space covariances or by keeping track of *inverse* state space covariances. Either choice improves the numerical speed and accuracy of the algorithm. Kalman's form keeps track of state space covariances and uses the *Kalman gain matrix* which in the notation of (4.41) is $K = CB^{-1}$. To get (4.43) into Kalman's form, note that, by (A.5a), $D^{-1} = A - CB^{-1}C^\top$, and thus

$$\Sigma_\alpha(t) = \left(\mathbf{I} - K(t)G(t)\right)\Sigma_a(t), \tag{4.44a}$$
$$\mu_\alpha(t) = \mu_a(t) + K(t)\left(y(t) - \mu_\gamma(t)\right). \tag{4.44b}$$

Using (4.40), (4.41), and (4.42), one finds the Kalman gain matrix in terms of known quantities to be

$$\begin{aligned}
CB^{-1} &\equiv K(t)\\
&= \Sigma_a(t)\big(G(t)\big)^\top\big(\Sigma_\gamma(t)\big)^{-1}\\
&= \Sigma_a(t)\big(G(t)\big)^\top\big(G(t)\Sigma_a(t)G(t)^\top + \Sigma_\epsilon\big)^{-1}.
\end{aligned}$$

The formula requires inversion of an $m \times m$ matrix, where m, the dimension of the observation y, is usually less than the dimension of the state x.

4.3.2 Backward Recursion

Equation (4.19) states the backward recursion as the integral

$$\beta(x, t-1) = \frac{1}{\gamma(t)} \int \beta(x', t) P_{Y(t)|X(t)} \left(y(t)|x'\right) P_{X(t)|X(t-1)} \left(x'|x\right) dx'.$$

In the context of linear Gaussian systems, $P\left(y(t)|x(t)\right)$ is the Gaussian $\mathcal{N}\left(G(t)x(t), \Sigma_\epsilon\right)\big|_{y(t)}$, $P\left(x(t)|x(t-1)\right)$ is the Gaussian $\mathcal{N}\left(F(t)x(t-1), \Sigma_\eta\right)\big)\big|_{x(t)}$, and $\frac{\beta(x,t)}{N_\beta(t)}$ is the Gaussian $\mathcal{N}\left(\mu_\beta(t), \Sigma_\beta(t)\right)\big|_x$. Thus

$$\beta(x, t-1) = \frac{N_\beta(t)}{\gamma(t)} \int \mathcal{N}\left(\mu_\beta(t), \Sigma_\beta(t)\right)\big|_{x'} \mathcal{N}\left(G(t)x', \Sigma_\epsilon\right)\big|_{y(t)} \mathcal{N}\left(F(t)x, \Sigma_\eta\right)\big|_{x'} dx'.$$

Since, for smoothing, the only parameters of β required are μ_β and Σ_β, I adopt the following strategy to calculate those parameters without evaluating the entire integral:

1. Drop time from notation in calculations.
2. Forget normalization and work only with exponents.
3. Combine $y(t)$ with $\mathcal{N}\left(\mu_\beta(t), \Sigma_\beta(t)\right)\big|_{x'}$ and $\mathcal{N}\left(G(t)x', \Sigma_\epsilon\right)\big|_{y(t)}$ to get an *updated distribution* for $x(t)$ with mean $\mu_b(t)$ and covariance $\Sigma_b(t)$.
4. Write the exponent of the product $\mathcal{N}\left(\mu_\beta(t), \Sigma_\beta(t)\right)\big|_{x'} \cdot \mathcal{N}\left(F(t)x, \Sigma_\eta\right)\big|_{x'}$ as

   $$Q = (x'-\mu)^\top \Sigma^{-1}(x'-\mu) + (x-\mu_\beta(t-1))^\top \left(\Sigma_\beta(t-1)\right)^{-1} (x-\mu_\beta(t-1)) + R,$$

   where Q is quadratic and the remainder R depends on neither x' nor x.
5. Since the integration over x' eliminates the term $(x'-\mu)^\top \Sigma^{-1}(x'-\mu)$ and R affects only normalization, the surviving parameters, $\mu_\beta(t-1)$ and $\Sigma_\beta(t-1)$, describe the function $\beta(x, t-1)$ up to normalization.

I begin with step 3 in the list above by examining the quadratic form in the exponent of $\mathcal{N}\left(\mu_\beta, \Sigma_\beta\right)\big|_x \cdot \mathcal{N}\left(Gx, \Sigma_\epsilon\right)\big|_y$:

$$\begin{aligned}
&(x-\mu_\beta)^\top \Sigma_\beta^{-1}(x-\mu_\beta) + (y-Gx)^\top \Sigma_\epsilon^{-1}(y-Gx) \\
&= x^\top \left(\Sigma_\beta^{-1} + G^\top \Sigma_\epsilon^{-1} G\right) x - 2x^\top \left(\Sigma_\beta^{-1}\mu_\beta + G^\top \Sigma_\epsilon^{-1} y\right) + \mu_\beta^\top \Sigma_\beta^{-1}\mu_\beta + y^\top \Sigma_\epsilon^{-1} y,
\end{aligned}$$

$$\begin{aligned}
\Sigma_b &= \left(\Sigma_\beta^{-1} + G^\top \Sigma_\epsilon^{-1} G\right)^{-1} \\
&= \Sigma_\beta - \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} G\Sigma_\beta, \\
\mu_b &= \Sigma_b \left(\Sigma_\beta^{-1}\mu_\beta + G^\top \Sigma_\epsilon^{-1} y\right) \\
&= \Sigma_b \left(\left(\Sigma_b^{-1} - G^\top \Sigma_\epsilon^{-1} G\right)\mu_\beta + G^\top \Sigma_\epsilon^{-1} y\right) \\
&= \mu_\beta + \Sigma_b G^\top \Sigma_\epsilon^{-1}(y - G\mu_\beta) \\
&= \mu_\beta + \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} (y - G\mu_\beta),
\end{aligned}$$

where I justify the last equation by observing that

$$\begin{aligned}\Sigma_b G^\top \Sigma_\epsilon^{-1} &= \Sigma_\beta G^\top \Sigma_\epsilon^{-1} - \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} G\Sigma_\beta G^\top \Sigma_\epsilon^{-1} \\ &= \Sigma_\beta G^\top \left(\mathbf{I} - \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} G\Sigma_\beta G^\top\right) \Sigma_\epsilon^{-1} \\ &= \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} \left(G\Sigma_\beta G^\top + \Sigma_\epsilon - G\Sigma_\beta G^\top\right) \Sigma_\epsilon^{-1} \\ &= \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1}.\end{aligned}$$

Thus

$$\Sigma_b = (\mathbf{I} - K_b G)\,\Sigma_\beta, \tag{4.45a}$$

$$\mu_b = \mu_\beta + K_b(y - G\mu_\beta), \tag{4.45b}$$

where

$$K_b \equiv \Sigma_\beta G^\top \left(G\Sigma_\beta G^\top + \Sigma_\epsilon\right)^{-1} \tag{4.45c}$$

is called the backward Kalman gain matrix.

Next, I address step 4 in the list above using the abbreviations

$$\begin{aligned}\tilde{\Sigma} &\equiv (\Sigma_b^{-1} + \Sigma_\eta^{-1})^{-1} \\ &= \Sigma_\eta - \Sigma_\eta(\Sigma_\eta + \Sigma_b)^{-1}\Sigma_\eta, \\ \tilde{\mu} &\equiv \tilde{\Sigma}\left(\Sigma_b^{-1}\mu_b + \Sigma_\eta^{-1}Fx\right)\end{aligned}$$

to analyze $Q_{x,x'}$, the exponent of $\mathcal{N}\left(\mu_b, \Sigma_b\right)|_{x'} \cdot \mathcal{N}\left(F(t)x, \Sigma_\eta\right)\Big|_{x'}$:

$$\begin{aligned}Q_{x,x'} &= (x' - \mu_b)^\top \Sigma_b^{-1}(x' - \mu_b) + (x' - Fx)^\top \Sigma_\eta^{-1}(x' - Fx) \\ &= (x')^\top(\Sigma_b^{-1} + \Sigma_\eta^{-1})x' - 2(x')^\top(\Sigma_b^{-1}\mu_b - \Sigma_\eta^{-1}Fx) + \mu_b^\top\Sigma_b^{-1}\mu_b + x^\top F^\top\Sigma_\eta^{-1}Fx \\ &= (x' - \tilde{\mu})^\top\tilde{\Sigma}^{-1}(x' - \tilde{\mu}) - \tilde{\mu}^\top\tilde{\Sigma}^{-1}\tilde{\mu} + \mu_b^\top\Sigma_b^{-1}\mu_b + x^\top F^\top\Sigma_\eta^{-1}Fx\end{aligned}$$

Note that $(x' - \tilde{\mu})^\top\tilde{\Sigma}^{-1}(x' - \tilde{\mu})$ goes away with the integration over x', and

$$\begin{aligned}\tilde{\mu}^\top\tilde{\Sigma}^{-1}\tilde{\mu} &= \left(\Sigma_b^{-1}\mu_b + \Sigma_\eta^{-1}Fx\right)^\top \left(\Sigma_b^{-1} + \Sigma_\eta^{-1}\right)^{-1} \left(\Sigma_b^{-1}\mu_b + \Sigma_\eta^{-1}Fx\right) \\ &= x^\top F^\top\Sigma_\eta^{-1}\tilde{\Sigma}^{-1}\Sigma_\eta^{-1}Fx + 2xF^\top\Sigma_\eta^{-1}\tilde{\Sigma}^{-1}\Sigma_b^{-1}\mu_b + \mu_b^\top\Sigma_b^{-1}\tilde{\Sigma}^{-1}\Sigma_b^{-1}\mu_b \\ &= x^\top F^\top\left(\Sigma_\eta^{-1} - \left(\Sigma_\eta + \Sigma_b\right)^{-1}\right)Fx + 2xF^\top\left(\Sigma_\eta + \Sigma_b\right)^{-1}\mu_b \\ &\quad + \mu_b^\top\Sigma_b^{-1}\tilde{\Sigma}^{-1}\Sigma_b^{-1}\mu_b.\end{aligned}$$

After integrating, the exponent of $\mathcal{N}\left(\mu_\beta, \Sigma_\beta\right)\Big|_x$ is

$$\begin{aligned}Q_x &= x^\top F^\top(\Sigma_\eta + \Sigma_b)^{-1}Fx - 2xF^\top\left(\Sigma_\eta + \Sigma_b\right)^{-1}\mu_b + \text{ scalar terms}, \\ Q_x &= (x - \mu_b)^\top F^\top(\Sigma_\eta + \Sigma_b)^{-1}F\,(x - \mu_b) + \text{ scalar terms}.\end{aligned} \tag{4.46}$$

Finally, I implement step 5 and pick out the following terms from (4.46):

$$\begin{aligned}\Sigma_\beta(t-1) &= (F(t))^{-1}(\Sigma_\eta + \Sigma_b(t))\left((F(t))^{-1}\right)^\top, \\ \mu_\beta(t-1) &= (F(t))^{-1}\mu_b(t).\end{aligned}$$

4.3.3 Smoothing

I have defined $\alpha(x,t)$ and $\beta(x,t)$ so that

$$P_{X(t)|Y_1^T}\left(x|y_1^T\right) = \alpha(x,t)\beta(x,t).$$

(See (4.27).) In fact $P_{X(t)|Y_1^T}$ is Gaussian, and I denote its parameters $\mu_{\alpha\beta}(t)$ and $\Sigma_{\alpha\beta}(t)$. Examining the exponential terms in $\alpha(x,t)\beta(x,t)$, I find (suppressing the time indices) that

$$\begin{aligned}
&(x-\mu_\alpha)^\top \Sigma_\alpha^{-1}(x-\mu_\alpha) + \left(x-\mu_\beta\right)^\top \Sigma_\beta^{-1}\left(x-\mu_\beta\right) \\
&= x^\top\left((\Sigma_\alpha(t))^{-1} + \left(\Sigma_\beta(t)\right)^{-1}\right)x - 2x^\top\left((\Sigma_\alpha(t))^{-1}\mu_\alpha(t) + \left(\Sigma_\beta(t)\right)^{-1}\mu_\beta(t)\right) \\
&\quad + \mu_\alpha^\top(\Sigma_\alpha(t))^{-1}\mu_\alpha + \mu_\beta^\top\left(\Sigma_\beta(t)\right)^{-1}\mu_\beta,
\end{aligned}$$

which implies

$$\left(\Sigma_{\alpha\beta}(t)\right)^{-1} = (\Sigma_\alpha(t))^{-1} + \left(\Sigma_\beta(t)\right)^{-1} \tag{4.47a}$$

and

$$\mu_{\alpha\beta}(t) = \Sigma_{\alpha\beta}(t)\left((\Sigma_\alpha(t))^{-1}\mu_\alpha(t) + \left(\Sigma_\beta(t)\right)^{-1}\mu_\beta(t)\right). \tag{4.47b}$$

4.3.4 Inverse Covariance Form

If the covariance of the state distribution is singular, one must propagate inverse covariances rather than covariances (see page 239 of Maybeck [10]). Kailath, Sayed, and Hassibi call this *The Information Form* (see page 332 of [7]). The procedure is useful when one makes the backwards pass through the data for smoothing because the initial inverse covariance is zero.

4.3.5 Extended Kalman Filter

Recall (4.1) from the beginning of the chapter:

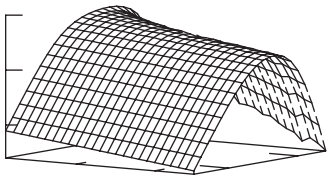
$$\begin{aligned}
x(t+1) &= F(x(t),t) + \eta(t), \\
y(t) &= G(x(t),t) + \epsilon(t),
\end{aligned}$$

and note that if the functions F and G are linear and the noise terms $\eta(t)$ and $\epsilon(t)$ are independent and Gaussian, then the Kalman filter implements the forward algorithm. If, on the other hand, the functions F and G are nonlinear but the errors of first order approximations to them are small compared to the size of the noise, then one can reasonably apply the same algorithm to the approximations. The resulting procedure is called an *extended Kalman filter*. While, as I noted in Section 3.2, there are more robust alternatives, the simplicity of extended Kalman filters explains their frequent use. In Section 1.1, I applied an extended Kalman filter to laser measurements for the following purposes:

- Estimate model parameters.

- Estimate state space trajectory.
- Forecast measurements.

In the next chapter, I will compare the likelihood of a model implemented as an extended Kalman filter to a performance bound obtained from Lyapunov exponent estimates.



Chapter 5
Performance Bounds and a Toy Problem

Having developed algorithms for fitting model parameters, one might reasonably ask how well the resulting models perform. In this chapter, I argue that the exercise of fitting models to data from chaotic dynamical systems is interesting because *Lyapunov exponent* calculations give a quantitative benchmark against which to compare model performance. The idea is that the stretching or local instability of dynamics, which Lyapunov exponents characterize, limits the predictability of sequences of observations. I will start by examining a toy example derived from the Lorenz system that informally introduces the main ideas. From there I will review definitions of entropy and Lyapunov exponents and results from information theory and ergodic theory that connect the ideas. Finally, I explain a simple calculation that can determine that a proposed model is fundamentally suboptimal.

I suppose that there is a *true* model of the stochastic process that assigns probabilities to sequences of observations. Many of the terms that I define are *expected values* with respect to those probabilities, and I find that sample sequences converge to those expected values. I use $P_{*|\mu}$ to denote these *true* probabilities,[23] and I use them to define expected values without delving into theoretical questions about their existence.

Lorenz Example

As an example, I have simulated a version of the Lorenz system (see (1.1)) modified to fit the form of (4.1),

$$\begin{aligned} x(t+1) &= F(x(t), t) + \eta(t), \\ y(t) &= G(x(t), t) + \epsilon(t)). \end{aligned}$$

I have used the extended Kalman filter described in Chapter 4 to obtain parametric probability functions $P\left(y(t)|y_1^{t-1}, \theta\right)$ that approximate $P\left(y(t)|y_1^{t-1}, \mu\right)$, i.e., the conditional

[23] Following Kolmogorov, modern probability theory is cast as a subfield of measure theory. The measure theory literature uses the Greek letter μ for a function or *measure* that maps sets to $\mathbb{R}^+$. In earlier chapters, I have used $P_{*|\theta}$ to denote parametric families of distributions. I introduce the mongrel notation $P_{*|\mu}$ here to make the notation for comparisons between a true distribution and a parametric model natural. The meaning of the Greek letter μ here is not related to its use to denote the mean of a distribution.

distribution of the measurement at time t given all previous measurements. My code for generating sequences of measurements has the following characteristics:

State.

$$x \equiv \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Time step. I obtain the map F by numerically integrating (1.1) for time intervals of length τ_s with an absolute error tolerance of 10^{-7}.

i.i.d. state noise.

$$\eta(t) \sim \mathcal{N}\left(0, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \sigma_\eta^2\right).$$

Measurement function. A simple projection

$$G(x) = x_1 = \begin{bmatrix} 1, 0, 0 \end{bmatrix} \cdot x.$$

i.i.d. measurement noise.

$$\epsilon(t) \sim \mathcal{N}(0, \sigma_\epsilon^2).$$

Quantization. The observations are quantized with a resolution Δ. I analyze quantized measurements rather than continuous measurements because they provide a *finite* rather than *infinite* amount of information, and they are characterized by *coordinate invariant* probability mass functions rather than *coordinate dependent* probability density functions.

Recall that $\mu_\gamma(t)$ and $\sigma_\gamma(t)$ completely characterize $P(y(t)|y_1^{t-1}, \theta)$ with

$$P(y(t)|y_1^{t-1}, \theta) = \mathcal{N}\left(\mu_\gamma(t), \sigma_\gamma^2(t)\right)\Big|_{y(t)}.$$

(I use a lowercase sigma here because the observations are scalars.) I obtain affine maps for the approximation $F(x + \delta, t) \approx [DF(x)]\,\delta + F(x)$ by numerically integrating both the Lorenz system and the tangent equations. I use those approximations with (4.30a) and (4.31) on page 64 to implement the recursive calculation of $\mu_\gamma(t)$ and $\sigma_\gamma(t)$ described by (4.7)–(4.17) on pages 60–61.

Figs. 5.1 and 5.2 depict a simulation in which dynamical stretching, i.e., the linear instability of $[DF(x)]$, occasionally limits predictability. I chose the parameters, specified in the caption of Fig. 5.1, so that dynamical noise and measurement quantization are negligible compared to the effects of measurement noise and dynamical stretching. In the middle plot of Fig. 5.1 notice that while for most times the forecast deviation of the predictions $\sigma_\gamma(t)$ is very close to the size of the measurement noise (0.01), occasionally the forecast deviations are many times larger. The log likelihood per time step which appears in the bottom plot of the figure is low when either the forecast deviations are large or when the difference between the mean of the forecast and the actual observation is much larger than the predicted deviation, i.e., $\sigma_\gamma^2(t) \ll \big(y(t) - \mu_\gamma(t)\big)^2$.

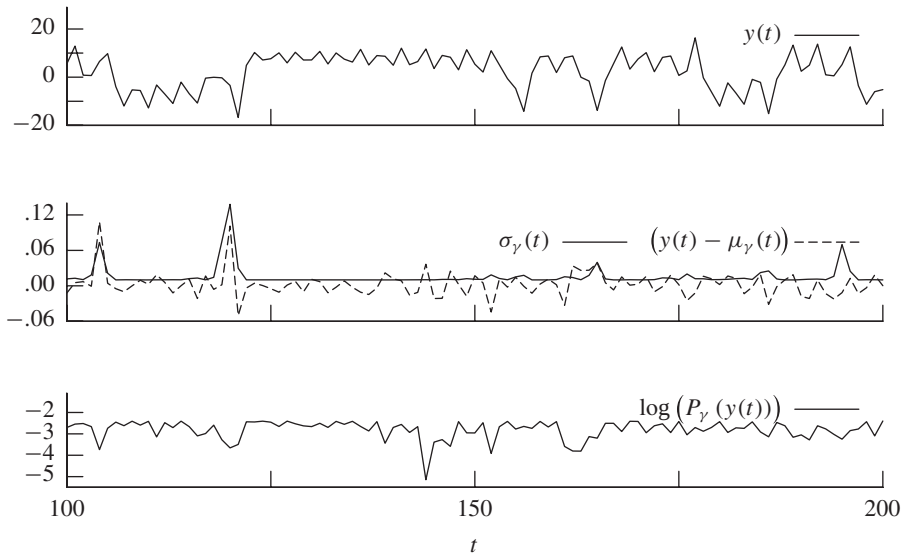


Figure 5.1. *Extended Kalman filter for one-step forecasting with simulation parameters*

$\tau_s = 0.25$	*Sample interval*
$\sigma_\eta = 10^{-6}$	*Standard deviation of state noise*
$\sigma_\epsilon = 0.01$	*Standard deviation of measurement noise*
$\Delta = 10^{-4}$	*Measurement quantization*

A time series of observations appears in the upper plot. The middle plot characterizes the one-step forecast distributions $P_\gamma\left(y(t)\right) \equiv P\left(y(t)|y_1^{t-1}, \theta\right) = \mathcal{N}\left(\mu_\gamma(t), \sigma_\gamma^2(t)\right)\Big|_{y(t)}$*; the first trace is the standard deviations of the forecasts, and the second trace is the difference between the actual observation and the mean of the forecast. The logs of the likelihoods of the forecasts,* $\log(P_\gamma\left(y(t)\right))$*, appear in the bottom plot.*

The largest excursion of $\sigma_\gamma(t)$ in Fig. 5.1 occurs between $t = 117$ and $t = 122$. Fig. 5.2 illustrates the stretching action of the map $[DF]$ that occurs in that interval; the dynamics map the smaller ellipse in the plot on the left into the larger ellipse in the plot on the right.

Fig. 5.3 illustrates the behavior of $-\hat{h}$, the sample average of the log likelihood of forecasts, for a group of simulations with parameters that are quite different from those in Figs. 5.1 and 5.2. Given model parameters θ and a sample sequence y_1^T of length T, I define

$$\begin{aligned} -\hat{h} &\equiv \frac{1}{T}\sum_{t=1}^{T} \log\left(P\left(y(t)|y_1^{t-1}, \theta\right)\right) \\ &= \frac{1}{T}\log\left(P\left(y_1^T|\theta\right)\right). \end{aligned}$$

The negative of this sample log likelihood is an estimate of the *cross entropy rate*

$$h(\mu||\theta) \equiv \lim_{T\to\infty} -\frac{1}{T}\mathbb{E}_\mu \log\left(P\left(Y_1^T|\theta\right)\right),$$

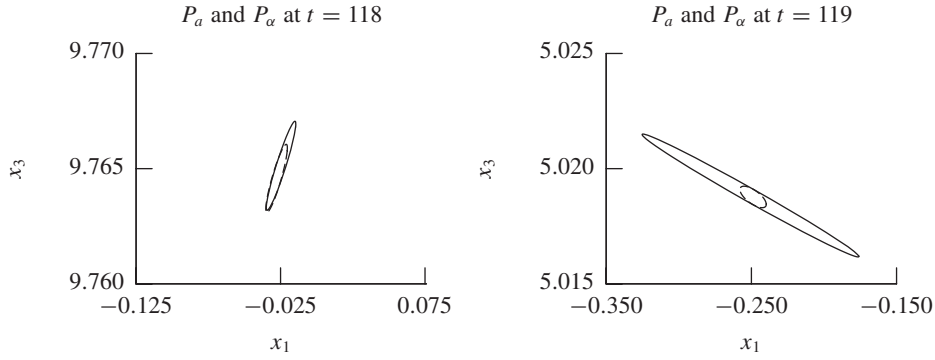


Figure 5.2. *These plots illustrate dynamical stretching increasing the variance of the conditional distribution in state space corresponding to time steps* 118 *and* 119 *in Fig.* 5.1. *In each plot, the larger ellipse represents the* forecast *state distribution* $P_a\,(x(t)) \equiv P\left(x(t)|y_1^{t-1}, \theta\right) = \mathcal{N}\,(\mu_a, \Sigma_a)|_{x(t)}$, *and the smaller ellipse represents the* updated *state distribution* $P_\alpha\,(x(t)) \equiv P\left(x(t)|y_1^{t}, \theta\right) = \mathcal{N}\,(\mu_\alpha, \Sigma_\alpha)|_{x(t)}$. *For each distribution, an ellipse depicts the level set* $(x-\mu)^\top \Sigma^{-1}(x-\mu) = 1$ *in the* $x_1 \times x_3$ *plane. Since the observations provide information about the value of* x_1, *the updated distributions vary less in the* x_1 *direction than the corresponding forecasts. To aid comparisons, the* x_1 *range is* 0.2 *and the* x_3 *range is* 0.01 *in each plot. In the* x_1 *direction, the standard deviation of the updated distribution* $P_\alpha(x(t))$ *at* $t = 118$ *(the smaller of the two ellipses on the left) is* 0.007. *The dynamics map that distribution to the forecast distribution* $P_a(x(t))$ *at* $t = 119$ *(the larger of the two ellipses on the right) for which the standard deviation in the* x_1 *direction is more than* 10 *times larger.*

which in turn is bounded from below by the *entropy rate*. I discuss both entropy rate and cross entropy rate in Section 5.2, and in Section 5.3, I review the *Pesin formula*. That formula says (with qualifications that include this context) that the largest *Lyapunov exponent* λ_1, a characterization of the stretching action of the dynamics, is equal to the entropy rate, a characterization of the predictability. For good models, the log likelihood of forecasts should fall off with a slope of $-\lambda_1$ as the sample time τ_s increases, and for the best possible model

$$h(\tau_s) = \lambda_1 \tau_s. \tag{5.1}$$

I have chosen the parameters[24] specified in the caption of Fig. 5.3 with a large measurement quantization size so that the log likelihood is limited primarily by dynamical

[24]In making Fig. 5.3, I wanted simulations close to the bound of (5.1). I found that at larger values of τ_s and Δ, extended Kalman filters performed better if provided with models that have larger state noise than the noise actually used to generate the data, i.e., $\tilde{\sigma}_\eta > \sigma_\eta$. I believe that the effect is the result of the larger errors that occur as the affine approximation $F(x+\delta) \approx [DF(x)]\delta + F(x)$ fails to track the nonlinearities of the Lorenz system over larger intervals in state space. By making $\tilde{\sigma}_\eta$ larger, the errors are accommodated as state noise. I chose the state noise of the generating process to be an order of magnitude larger than the absolute integration tolerance of 10^{-7}. I then chose the quantization level and sample times to be as large as possible but still small enough that I could have $\tilde{\sigma}_\eta = \sigma_\eta$ without losing performance. That led to the values $\tilde{\sigma}_\eta = \sigma_\eta = 10^{-6}$, $\Delta = 10^{-4}$, and $0 < \tau_s \leq 0.5$.

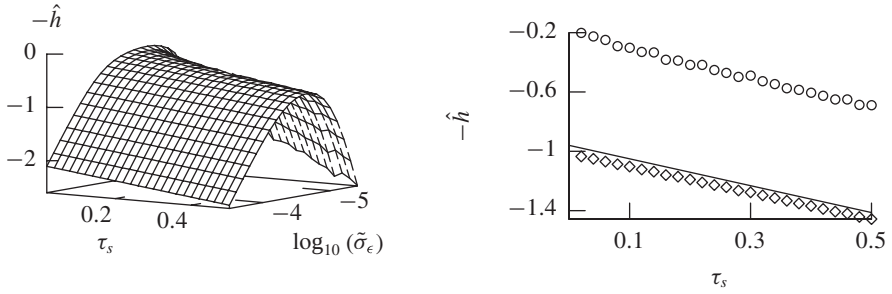


Figure 5.3. *Average log likelihood of one-step forecasts as a function of time step τ_s and filter parameter $\tilde{\sigma}_\epsilon$. To simulate measurements for this figure, I used the parameters*

$\sigma_\eta = 10^{-6}$ *Standard deviation of state noise*
$\sigma_\epsilon = 10^{-10}$ *Standard deviation of measurement noise*
$\Delta = 10^{-4}$ *Measurement quantization*
$T = 5,000$ *Number of samples*

For both plots, the vertical axis is the average log likelihood of the one-step forecast $-\hat{h} \equiv \frac{1}{T}\sum_{t=1}^{T} \log\left(P\left(y(t)|y_1^{t-1},\theta\right)\right)$. On the left, I plot $-\hat{h}$ as a function of both τ_s, the time step, and $\tilde{\sigma}_\epsilon$, the standard deviation of the measurement noise model used by the Kalman filter. On the right, "$\circ$"indicates the performance of filters that use measurement noise models that depend on the sampling time through the formula $\tilde{\sigma}_\epsilon(\tau_s) = 10^{0.4\tau_s - 4.85}$, which closely follows the ridge top in the plot on the left, "$\diamond$"indicates the performance of filters that use $\tilde{\sigma}_\epsilon = 10^{-4}$, i.e., the measurement quantization level, and the solid line traces (5.2) *in the text.*

stretching and the Gaussian assumption for $P\left(y(t)|y_1^{t-1},\theta\right)$. Notice that the overall slope of the plot on the left in Fig. 5.3 is consistent with the estimate $\hat{\lambda}_1 = 0.906$ (base e) obtained using the Benettin procedure described in Section 5.4.

In the plot on the right in Fig. 5.3, notice that for a class of filters in which the standard deviation of the model measurement noise $\tilde{\sigma}_\epsilon$ is set to the quantization size Δ the log likelihood closely follows the approximation

$$\hat{h}(\tau_s) = \log\left(\operatorname{erf}\left(\frac{1}{2\sqrt{2}}\right)\right) + \lambda_1 \tau_s, \tag{5.2}$$

where *erf* is the error function.[25] I explain the nonzero intercept in (5.2) by observing that in the limit of small sampling interval ($\tau_s \to 0$) and zero noise ($\sigma_\eta \to 0$ and $\sigma_\epsilon \to 0$) only one discrete observation $y(t) = \bar{y}$ is possible given a history y_1^{t-1}. For data drawn from that limiting case, a Kalman filter with parameters $\tilde{\sigma}_\epsilon = \Delta$ and $\sigma_\eta \to 0$ would make a forecast with a density $P(y(t)|y_1^{t-1}) = \mathcal{N}\left(\bar{y}, (\Delta)^2\right)\big|_{y(t)}$. Integrating that density over the

[25]The error function is defined by $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

quantization interval yields

$$\begin{aligned} P_{y(t)|\theta}(\bar{y}) &= \int_{\bar{y}-\frac{\Delta}{2}}^{\bar{y}+\frac{\Delta}{2}} \frac{1}{\sqrt{2\pi(\Delta)^2}} e^{-\frac{(y-\bar{y})^2}{2(\Delta)^2}} dy \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}s^2} ds \\ &= \operatorname{erf}\left(\frac{1}{2\sqrt{2}}\right) \\ &\approx 0.3829, \\ \log\left(P_{y(t)|\theta}(\bar{y})\right) &\approx -0.9599. \end{aligned}$$

Given the simplicity of the analysis, (5.2) fits the simulations in Fig. 5.3 remarkably well.

5.1 Fidelity Criteria and Entropy

Stochastic models are fit to an enormous variety of measured phenomena, and the most appropriate measure of the fidelity of a model to measurements depends on the application. Such phenomena include long and short term weather, financial markets, computer data, electric power demand, and signals and noise in communication or instrumentation. In many cases one makes decisions based on a model, and those decisions change the *cost* of future events. The expected cost of basing decisions on a model $P_{*|\theta}$ depends in a complicated fashion on many factors, including how the cost of acting on a decision depends on lead time and which aspects of the modeled phenomenon are important. For the Lorenz example at the beginning of this chapter, I implicitly assumed *stationarity* and *ergodicity* and characterized model quality in terms of the average of the log of the likelihood. For a stationary ergodic system, the log likelihood is tied to $D(\mu||\theta)$, the *relative entropy* of the model $P_{*|\theta}$ given $P_{*|\mu}$ (see (5.5)). Although relative entropy is not an appropriate performance measure for every application, it is a common tool for problems in information theory and statistics. See Cover and Thomas [2] for many of these, including the application of relative entropy to the theory of gambling. Relative entropy is exactly the right performance measure for data compression. The arithmetic coding algorithm (see the review by Witten, Neal, and Cleary [24]) for compressing a sequence of symbols uses a model $P_{*|\theta}$ to make decisions that affect the cost in a manner that depends on the symbol values that actually occur. The relative entropy $D(\mu||\theta)$ is the expected value of the number of bits wasted by the algorithm if it uses a model $P_{*|\theta}$ for decisions when in fact $P_{*|\mu}$ is true. More accurate models lead to better compression.

5.1.1 Definitions

Now, to solidify the discussion, I make some formal definitions.

Stochastic Process

I am interested in sequences of states X_1^T and measurements Y_1^T, each of which can be thought of as a *random function* on the domain $\{1, 2, \ldots, T\}$, i.e., a *stochastic process*.

Entropy of a Discrete Random Variable

If a discrete random variable U takes on the values $u_1, u_2, \ldots, u_n$ with probabilities $P(u_1), P(u_2), \ldots, P(u_n)$, then the *entropy* of U is

$$H(U) \equiv -\mathbb{E}\log\left(P(U)\right) = -\sum_{k=1}^{n} P(u_k)\log\left(P(u_k)\right). \tag{5.3}$$

Entropy quantifies the uncertainty in U before its value is known and the information or degree of surprise in discovering its value. If the base of the logarithm in (5.3) is 2, then the units of $H(U)$ are called *bits*. I will use natural logarithms with the Euler constant e as the base. For natural logarithms the units of $H(U)$ are called *nats*.

Differential Entropy of a Continuous Random Variable

If U is a continuous random variable with a probability density function P, then the *differential entropy* of U is

$$\tilde{H}(U) \equiv -\mathbb{E}\log\left(P(U)\right) = -\int P(u)\log\left(P(u)\right)du. \tag{5.4}$$

Notice that the differential entropy depends on the coordinates used to describe U.

Conditional Entropy

The *conditional entropy* of U given V is

$$H(U|V) \equiv -\mathbb{E}\log\left(P(U|V)\right) = -\sum_{i,j} P(u_i, v_j)\log\left(P(u_i|v_j)\right).$$

Factoring the probability of sequences

$$P_{z_1^T} = P_{z(1)}\prod_{t=2}^{T} P_{z(t)|z_1^{t-1}}$$

is equivalent to analyzing entropy into the sum

$$H(Z_1^T) = H(Z(1)) + \sum_{t=2}^{T} H\left(Z(t)|Z_1^{t-1}\right).$$

Relative Entropy of Two Probability Functions

The *relative entropy* between two probability functions $P_{*|\mu}$ and $P_{*|\theta}$ with the same domain $\mathcal{Z}$ is

$$\begin{aligned} D(\mu||\theta) &\equiv \mathbb{E}_\mu \log\left(\frac{P(Z|\mu)}{P(Z|\theta)}\right) \\ &= \sum_{z\in\mathcal{Z}} P(z|\mu)\log\left(\frac{P(z|\mu)}{P(z|\theta)}\right). \end{aligned} \tag{5.5}$$

The relative entropy is coordinate independent. The relative entropy between two probability functions $P_{*|\mu}$ and $P_{*|\theta}$ is never negative and is zero iff the functions are the same on all sets with finite probability. I use $D(\mu||\theta)$ to characterize the fidelity of a model $P_{*|\theta}$ to a *true* distribution $P_{*|\mu}$.

Cross Entropy of Two Probability Functions

While some authors use the terms *relative entropy* and *cross entropy* interchangeably to mean the quantity $D(\mu||\theta)$ defined in (5.5), I define the cross entropy to be

$$\begin{aligned} H(\mu||\theta) &\equiv -\mathbb{E}_\mu \log\left(P(Z|\theta)\right) \\ &= -\sum_{z\in\mathcal{Z}} P(z|\mu)\log(P(z|\theta)) \end{aligned} \tag{5.6}$$

and note that

$$D(\mu||\theta) = H(\mu||\theta) - H(\mu).$$

The cross entropy is the negative expected log likelihood of a model. It is greater than the entropy unless the model $P_{*|\theta}$ is the same as $P_{*|\mu}$ for all sets with finite probability.

Stationary

A stochastic process is *stationary* if probabilities are unchanged by constant shifts in time; i.e., for any two integers $T \geq 1$ and $\tau \geq 0$,

$$P_{Z_1^T} = P_{Z_{1+\tau}^{T+\tau}}.$$

Ergodic

Roughly, in an *ergodic* process you can get anywhere from anywhere else. Let $\mathcal{X}$ be the set of states for a stationary stochastic process with probabilities $P_{*|\mu}$. The process is ergodic if for any two subsets of $\mathcal{X}$, A and B with $P(A|\mu) > 0$ and $P(B|\mu) > 0$, there is a time T such that the probability of going from set A to set B in time T is greater than zero. Birkhoff's ergodic theorem says that, for an ergodic process, time averages converge to expected values with probability one.

Entropy Rate

For a discrete stochastic process X, the *entropy rate* is

$$h(X) \equiv \lim_{T\to\infty} \frac{1}{T} H(X_1^T). \tag{5.7}$$

If the process is stationary,

$$h(X) = \lim_{T\to\infty} H(X(T)|X_1^{T-1}). \tag{5.8}$$

If the process is stationary and Markov,

$$h(X) = H(X(T+1)|X(T)) \quad \forall T. \tag{5.9}$$

And if the process is ergodic,

$$h(X) = \lim_{T\to\infty} -\frac{1}{T} \log\left(P\left(x_1^T|\mu\right)\right) \tag{5.10}$$

with probability one.

I similarly define the relative entropy rate and the cross entropy rate. For an ergodic process X with true probabilities $P_{*|\mu}$ and model probabilities $P_{*|\theta}$, the cross entropy rate is

$$\begin{aligned} h(\mu||\theta) &\equiv \lim_{T\to\infty} -\frac{1}{T}\mathbb{E}_\mu \log\left(P\left(X_1^T|\theta\right)\right) \\ &= \lim_{T\to\infty} -\frac{1}{T} \log\left(P\left(x_1^T|\theta\right)\right) \end{aligned} \tag{5.11}$$

with probability one.

Entropy Rate of a Partition $\mathcal{B}$

Let $\mathcal{X}$, the set of states for a stationary stochastic process with probabilities $P_{*|\mu}$, be a continuum, and let $\mathcal{B} = \{\beta_1, \beta_2, \ldots, \beta_n\}$ be a partition of $\mathcal{X}$ into a finite number of nonoverlapping subsets. By setting $b(t)$ to the index of the element of $\mathcal{B}$ into which $x(t)$ falls, I can map any sequence x_1^T into a sequence b_1^T, thus defining a discrete stationary stochastic process B. Applying the definition of entropy rate to the process B yields the definition of the entropy rate as a function of partition $\mathcal{B}$. Suppose, in particular, that on some set $\mathcal{X}$ the map $F : \mathcal{X} \mapsto \mathcal{X}$ and the probability $P_{*|\mu}$ define an ergodic process, that $\mathcal{B}$ is a partition of $\mathcal{X}$, and that the model probability function $P_{*|\theta}$ assigns probabilities to sequences of partition indices b_1^T. I define the entropy rate $h(\mathcal{B}, F, \mu)$ and the cross entropy rate $h(\mathcal{B}, F, \mu||\theta)$ as follows:

$$\begin{aligned} h(\mathcal{B}, F, \mu) &\equiv \lim_{T\to\infty} \frac{1}{T} H(B_1^T) \\ &= \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\mu \log\left(P\left(B_1^T|\mu\right)\right), \\ h(\mathcal{B}, F, \mu||\theta) &\equiv \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\mu \log\left(P\left(B_1^T|\theta\right)\right). \end{aligned} \tag{5.12}$$

Kolmogorov–Sinai Entropy h_{KS}

As before, suppose that on some set $\mathcal{X}$ the map $F : \mathcal{X} \mapsto \mathcal{X}$ and the probability $P_{*|\mu}$ define an ergodic process. The least upper bound over all partitions $\mathcal{B}$ on the entropy rate $h(\mathcal{B}, F, \mu)$ is called the *Kolmogorov–Sinai entropy*:

$$h_{KS}(F, \mu) \equiv \sup_{\mathcal{B}} h(\mathcal{B}, F, \mu). \tag{5.13}$$

5.2 Stretching and Entropy

Here I will outline the theory that connects ideas from dynamics to ideas from probability. The main results say that average dynamical stretching (Lyapunov exponents) is proportional to average uncertainty (entropy) in measurement sequences. First, I will give some examples; then I will quote definitions and theorems without proof.

5.2.1 Maps of the Unit Circle

Two x mod One

The map of the unit interval $[0, 1)$ into itself defined by

$$x(n+1) = F_2(x(n)) \tag{5.14}$$
$$\equiv 2x \mod 1 \tag{5.15}$$

is continuous if one identifies the points 0 and 1. Notice that if I use the partition

$$\mathcal{B}_2 = \left\{ \beta_0 = \left[0, \frac{1}{2}\right), \beta_1 = \left[\frac{1}{2}, 1\right) \right\}, \tag{5.16}$$

a symbol sequence b_0^∞ provides the coefficients of a base two power series that identifies the starting point, i.e.,

$$x(0) = \sum_{t=0}^{\infty} b(t) \left(\frac{1}{2}\right)^{t+1}.$$

The partition $\mathcal{B}_2$ achieves the supremum in (5.13) and assigning a uniform probability measure μ to the interval yields

$$h_{KS}(F_2, \mu) = h(\mathcal{B}_2, F, \mu) = \log(2).$$

Three x mod One

Analogously, by defining the map

$$x(n+1) = F_3(x(n)) \tag{5.17}$$
$$\equiv 3x \mod 1, \tag{5.18}$$

and using the partition

$$\mathcal{B}_3 = \left\{ \beta_0 = \left[0, \frac{1}{3}\right), \beta_1 = \left[\frac{1}{3}, \frac{2}{3}\right), \beta_2 = \left[\frac{2}{3}, 1\right) \right\} \tag{5.19}$$

and uniform probability measure μ, I find that

$$h_{KS}(F_3, \mu) = h(\mathcal{B}_3, F_3, \mu) = \log(3).$$

This is the result that motivated Kolmogorov and Sinai to define the entropy h_{KS}. They were addressing *the isomorphism problem*, e.g., "Is there a relabeling of points in the unit interval that makes F_2 the same as F_3?" Since the characteristic h_{KS} is independent of the coordinates or labeling used, the fact that $h_{KS}(F_3, \mu) \neq h_{KS}(F_2, \mu)$ provided a negative answer to the question.

Notice that the Kolmogorov–Sinai entropy is equal to the average of the log of the slope of the map. Specifically, the slope of F_2 is 2 and $h_{KS}(F_2, \mu) = \log(2)$, while the slope of F_3 is 3 and $h_{KS}(F_3, \mu) = \log(3)$. The rule that entropy is proportional to the log of the average slope is not true in general. The next example provides a counterexample and suggests a correction factor.

Dynamics on a Cantor Set

While every point in the entire unit interval can be represented as a base three power series, i.e.,

$$\forall x \in [0, 1),\ \exists d_0^\infty :\ x = \sum_t^\infty d(t) \left(\frac{1}{3}\right)^{t+1} \text{ with } d(t) \in \{0, 1, 2\} \quad \forall t,$$

the middle third Cantor set consists of the points in the unit interval that can be represented as base three power series that exclude the digit "1." The symbol sequences produced by applying the map F_3 and partition $\mathcal{B}_3$ to the middle third Cantor set are the sequences of coefficients in the base three expansions of the starting points; i.e., they consist exclusively of 0's and 2's. Given any finite sequence d_0^n, I define the set of infinite coefficient sequences $\left\{d_0^n, \ldots\right\}$ as those that begin with the sequence d_0^n. Now I define a probability measure μ_c in terms of such sets of infinite sequences,

$$\mu_c\left(\left\{d_0^n, \ldots\right\}\right) \equiv \begin{cases} 2^{-(n+1)} & \text{if "1" does not appear in } d_0^n, \\ 0 & \text{if "1" does appear in } d_0^n. \end{cases} \tag{5.20}$$

With this measure I find that

$$h_{KS}(F_3, \mu_c) = \log(2).$$

The following isomorphism or relabeling of the unit interval connects (F_2, μ) to (F_3, μ_c):

1. Find the binary expansion b_0^∞ of the original point x.
2. Create a new sequence d_0^∞ by replacing every occurrence of "1" in b_0^∞ with "2."
3. Map x to y, where y is described by the base three expansion d_0^∞.

The Hausdorff dimension[26] of the middle third Cantor set is $\delta = \frac{\log(2)}{\log(3)}$, and that is the factor that is missing in the formula connecting entropy and stretching:

$$\begin{aligned} h_{KS}(F_3, \mu_c) &= \log(2) \\ &= \frac{\log(2)}{\log(3)} \log(3) \\ &= \delta \log(\text{stretching factor}). \end{aligned} \tag{5.21}$$

Now I turn to the definitions and theorems that express the above idea precisely.

5.3 Lyapunov Exponents and Pesin's Formula

Vixie [27] has reviewed the work of Ruelle [49], Pesin [45], Young [25], and others, who established the relationship between smooth dynamics and entropy. Here I reiterate a few of those results using the following notation:

X An n-dimensional manifold

$F : X \mapsto X$ An invertible C^2 (continuous with continuous first and second derivatives) map of the manifold into itself

μ A probability measure on X that is invariant under F

x A point on the manifold

$TX(x)$ The tangent space of X at x

v An element of $TX(x)$

I define the asymptotic growth rate of the direction v at x as

$$\lambda(F, x, v) \equiv \lim_{t\to\infty} \frac{1}{t} \log\left(\left\|[DF^t(x)]v\right\|\right). \tag{5.22}$$

Oseledec's theorem [25, 8, 9] says that at almost every x the limit exists for every v and that although the value of the limit depends on v, as v varies, it takes on only $r \leq n$ discrete values called the *spectrum of Lyapunov exponents*:

$$\lambda_1(F, x) > \lambda_2(F, x) > \cdots > \lambda_r(F, x). \tag{5.23}$$

The tangent space $TX(x)$ is the direct sum of subspaces $E_i \subset TX(x)$ associated with each exponent, i.e.,

$$TX(x) = \bigoplus_{i=1}^{r} E_i,$$

where for each $v \in E_i$

$$\lambda(F, x, v) = \lambda_i(F, x).$$

[26] I sometimes refer to sets and characteristics of sets with noninteger dimensions as *fractal*.

The dimension of E_i is called the *multiplicity* m_i of the exponent λ_i. If μ is ergodic with respect to F, then the spectrum $\{\lambda_i\}$ is the same almost everywhere.

If each exponent has unit multiplicity, the intuitive picture of a time dependent singular value decomposition

$$U(t) \cdot S(t) \cdot V(t)^\top = DF^t(x),$$

where $U(t)$ and $V(t)$ are orthogonal and $S(t)$ is diagonal with positive entries, may help. As $t \to \infty$, the growth rate of $s_i(t)$ is given by λ_i, and the ith column of $V(t)$ spans E_i.

I want to use Pesin's formula [45], which implies that if μ is smooth and ergodic, then the entropy is equal to the sum of the positive Lyapunov exponents, i.e.,

$$h_{KS}(F, \mu) = \sum_{i:\lambda_i > 0} m_i \lambda_i. \tag{5.24}$$

In light of the correction for fractal dimension in (5.21) and the ubiquity of fractal measures in chaotic systems, I should review Ledrappier and Young's explanation (see [25] for an overview) of the effect of fractal measures on Pesin's formula.

Ledrappier and Young's formula is given in terms of the dimensions of the conditional measures on the nested family of *unstable foliations* of F. For a point $x \in X$ and i such that $\lambda_i > 0$, I define

$$W^i(x) \equiv \left\{ y \in X \text{ such that } \lim_{t\to\infty} \sup \frac{1}{t} \log\left(d\left(F^{-t}(x), F^{-t}(y)\right)\right) < -\lambda_i \right\}. \tag{5.25}$$

For an intuitive picture, consider a trajectory $x(t)$ that passes through x at time $t = 0$; any trajectory $y(t)$ that has separated from $x(t)$ at a rate of at least λ_i passes through the manifold $W^i(x)$ at time $t = 0$.

Let δ_i be the Hausdorff dimension of the conditional measure that μ defines on $W^i(x)$. For an ergodic μ, δ_i will be constant almost everywhere. Further, let γ_i be the incremental dimension

$$\gamma_i \equiv \begin{cases} \delta_1, & i = 1, \\ \delta_i - \delta_{i-1}, & i > 1. \end{cases}$$

Now Ledrappier and Young's formula is

$$h_{KS}(F, \mu) = \sum_{i:\lambda_i > 0} \lambda_i \gamma_i. \tag{5.26}$$

Note that Pesin's formula holds if the measure μ is smooth in the unstable directions. Such measures are called Sinai–Ruelle–Bowen (SRB) measures. Tucker has found that the Lorenz system has an SRB measure and says that numerical simulations of Lorenz's system are "real" [52].

5.3.1 A Theoretical Bound on Model Likelihood

Now I have the terms that I need to discuss theoretical bounds on the expected log likelihood of models of discrete observations of a chaotic dynamical system. Given X, F, and μ as

described above, if the multiplicity of each exponent is $m_i = 1$, then I know that

$$h(\mathcal{B}, F, \mu) \equiv -\lim_{t\to\infty} \mathbb{E}_\mu \log\left(P\left(b(t)|b_1^{t-1}, \mu\right)\right), \tag{5.27}$$

$$h(\mathcal{B}, F, \mu||\theta) \equiv -\lim_{t\to\infty} \mathbb{E}_\mu \log\left(P\left(b(t)|b_1^{t-1}, \theta\right)\right), \tag{5.28}$$

$$h(\mathcal{B}, F, \mu) \leq h(\mathcal{B}, F, \mu||\theta) \qquad (\text{equality} \iff \theta = \mu \text{ a.e.}), \tag{5.29}$$

$$h(\mathcal{B}, F, \mu) \leq h_{KS}(F, \mu) \qquad (\text{equality} \iff \mathcal{B} \text{ generating}), \tag{5.30}$$

$$h_{KS}(F, \mu) = \sum_{i:\lambda_i>0} \lambda_i \gamma_i, \tag{5.31}$$

$$h_{KS}(F, \mu) \leq \sum_{i:\lambda_i>0} \lambda_i \qquad (\mu \text{ smooth on } W^i \Rightarrow \text{equality}) \tag{5.32}$$

with the following justifications:

(5.27) and (5.28): Definition.

(5.29): The Gibbs inequality, (2.53).

(5.30): The definition of $h_{KS}(F, \mu)$ is that it is the *supremum* over all partitions $\mathcal{B}$.

(5.31): This is Ledrappier and Young's formula (5.26).

(5.32): Because in (5.31) $0 \leq \gamma_i \leq 1$ for all i.

Thus I have the following two theorems:

Theorem 5.1 (Lyapunov exponent bound on likelihood). *If μ is ergodic and smooth in the unstable directions and $\mathcal{B}$ is a generating partition, then for any model θ of the stochastic process B consisting of F, μ, and $\mathcal{B}$*

$$h(\mathcal{B}, F, \mu||\theta) \geq \sum_{i:\lambda_i>0} \lambda_i. \tag{5.33}$$

Theorem 5.2 (entropy gap). *If μ is ergodic (not necessarily smooth in the unstable directions), then for an optimal model θ of the stochastic process B consisting of F, μ, and $\mathcal{B}$ ($\mathcal{B}$ not necessarily generating)*

$$h(\mathcal{B}, F, \mu||\theta) = h(\mathcal{B}, F, \mu) \leq \chi \equiv \sum_{i:\lambda_i>0} \lambda_i, \tag{5.34}$$

and if for some other model ν

$$h(\mathcal{B}, F, \mu||\nu) \geq \chi, \tag{5.35}$$

then the model ν is not optimal.

In the next section, I will describe a numerical procedure for estimating Lyapunov exponents, and in the following section I will argue that one can reasonably use (5.35) with numerical simulations to quantitatively characterize the nonoptimality of a model.

5.4 Benettin's Procedure for Calculating Lyapunov Exponents Numerically

I begin reviewing Benettin's procedure [30] for estimating Lyapunov exponents by using the Lorenz system as an example. The Lorenz system is

$$\dot{x} = F(x) = \begin{bmatrix} s(x_2 - x_1) \\ x_1(r - x_3) - x_2 \\ x_1 x_2 - b x_3 \end{bmatrix}.$$

Note that

$$DF(x) = \begin{bmatrix} -s & s & 0 \\ r - x_3 & -1 & -x_1 \\ x_2 & x_1 & -b \end{bmatrix},$$

where $(DF(x))_{i,j} \equiv \frac{\partial F_i(x)}{\partial x_j}$. Let Φ denote solutions to the differential equation with

$$x(\tau) \equiv \Phi(x(0), \tau).$$

Lyapunov exponents are defined (recall (5.22)) in terms of the long time behavior of the derivative matrix

$$\mathcal{D}(x(0), \tau) \equiv D_{x(0)} \Phi(x(0), \tau).$$

Interchanging the order of differentiation with respect to $x(0)$ and τ and applying the chain rule yields a *linear* differential equation for $\mathcal{D}$:

$$\begin{aligned} \dot{\mathcal{D}}(x(0), \tau) &= \frac{d}{d\tau} D_{x(0)} \Phi(x(0), \tau) \\ &= DF(x)|_{x = \Phi(x(0), \tau)} \, \mathcal{D}(x(0), \tau). \end{aligned}$$

Thus, given initial conditions $x(0)$ and $\mathcal{D}(0) = \mathbf{I}$ one can use an off-the-shelf routine to find $\begin{bmatrix} x(\tau) \\ \mathcal{D}(\tau) \end{bmatrix}$ by integrating

$$\begin{bmatrix} \dot{x}(\tau) \\ \dot{\mathcal{D}}(\tau) \end{bmatrix} = \begin{bmatrix} F(x) \\ [DF(x)]\mathcal{D} \end{bmatrix}. \tag{5.36}$$

Given a computer with infinite precision, for a range of time intervals τ, one could do the following:

1. Integrate (5.36) to obtain $\mathcal{D}(\tau)$.
2. Perform singular value decompositions (SVDs)

$$U(\tau) S(\tau) V^{\top}(\tau) = \mathcal{D}(\tau), \tag{5.37}$$

   where $U(\tau)$ and $V(\tau)$ are orthogonal and $S(\tau)$ is diagonal.

3. Look for approximate convergence of the finite time Lyapunov exponent estimates:

$$\tilde{\lambda}_i(\tau) \equiv \frac{1}{\tau} \log(S_{i,i}(\tau)). \tag{5.38}$$

On a real computer, the procedure fails because the ratio of the largest and smallest singular values $\frac{S_1(\tau)}{S_d(\tau)}$ grows exponentially with τ and becomes larger than the precision of the machine.

Rather than using an SVD for each τ in step 2 above, one could use a QR decomposition:

$$Q(\tau)R(\tau) = \mathcal{D}(\tau). \tag{5.39}$$

A QR decomposition factors the matrix $\mathcal{D}(\tau)$ into a product of two matrices, the first of which, $Q(\tau)$, is orthogonal and the second of which, $R(\tau)$, is upper triangular. One could use the intuitive Gram–Schmidt procedure, but other algorithms behave better numerically (see, e.g. [6] or [14]). Although the diagonal elements of $R(\tau)$ are not equal to the diagonal elements of $S(\tau)$, the finite time estimates

$$\hat{\lambda}_i(\tau) \equiv \frac{1}{\tau} \log\left(\left|R_{i,i}(\tau)\right|\right), \tag{5.40}$$

and the $\tilde{\lambda}_i(\tau)$ defined in (5.38) converge to the same values.[27]

Using (5.40) does not address the problem of finite machine precision for long time intervals τ, but Benettin et al. [30] recommend calculating $\log(|R_{i,i}(\tau)|)$ by breaking the interval into N smaller steps of duration $\Delta\tau$ in a way that does address finite precision. Letting $A(n)$ denote the one time step derivative

$$A(n) \equiv D\Phi(x((n-1)\Delta\tau), \Delta\tau) \tag{5.41}$$

the chain rule implies

$$D\Phi(x(0), N\Delta\tau) = \prod_{n=1}^{N} A(n).$$

If, for each n, one calculates[28] the pair $(Q(n), r(n))$ defined by

$$\begin{aligned} Q(0) &= \mathbf{I}, \\ Q(n)r(n) &\equiv A(n)Q(n-1), \end{aligned}$$

where $Q(n)$ and $r(n)$ are obtained by a QR factorization of the product $A(n)Q(n-1)$, then induction yields

$$\prod_{n=1}^{N} A(n) = Q(N) \prod_{n=1}^{N} r(n).$$

[27] In the SVD of (5.37), the first column of $V(\tau)$ specifies the direction of the initial vector in the tangent space with the largest stretching. The exponential stretching rate is the Lyapunov exponent λ_1. However, with probability one, a randomly chosen vector will have the same stretching rate. The estimate $\hat{\lambda}_1(\tau)$ of (5.40) is based on the stretching rate of the first standard basis vector, e.g., $[1, 0, 0]$. Similar arguments using the growth rates of areas, volumes, hypervolumes, etc., support using the estimates $\hat{\lambda}_i(\tau)$ of (5.40) for $i = 2, 3, \ldots$.

[28] To calculate $Q(n)$ and $r(n)$ for each n, one can do either of the following:

1. Integrate (5.36) for a time interval $\Delta\tau$ with the initial condition $\begin{bmatrix} x((n-1)\Delta\tau) \\ Q(n-1) \end{bmatrix}$ to obtain $\begin{bmatrix} x(n\Delta\tau) \\ A(n)Q(n-1) \end{bmatrix}$ and then calculate a QR factorization of $A(n)Q(n-1)$, the second component of the result.
2. Proceed as above but use the identity matrix instead of $Q(n-1)$ as the second component of the initial condition for the integration which yields the result $\begin{bmatrix} x(n\Delta\tau) \\ A(n) \end{bmatrix}$ and then calculate a QR factorization of the product $A(n)Q(n-1)$.

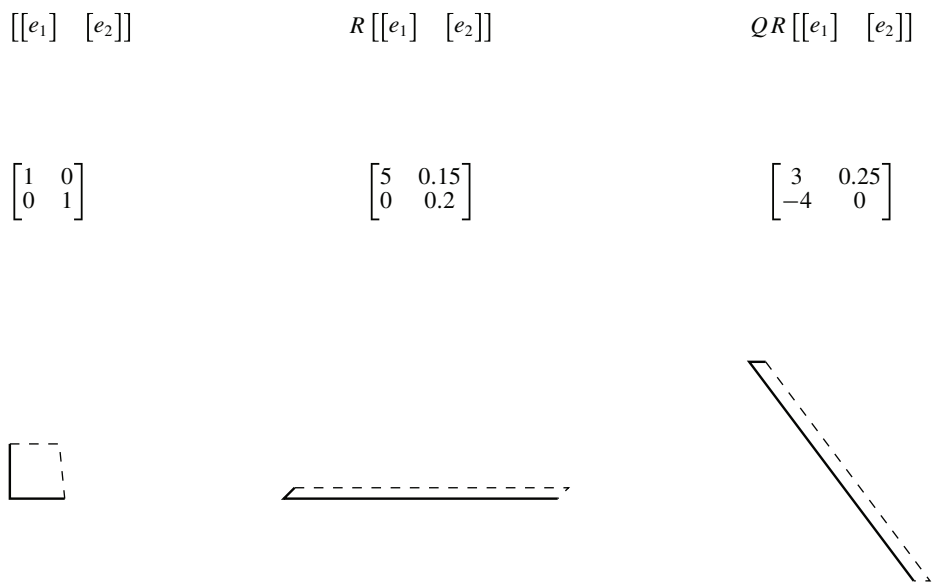


Figure 5.4. *The action of the Q and R factors of a matrix on a unit square. Here* $A = \begin{bmatrix} 3 & 0.25 \\ -4 & 0 \end{bmatrix}$, $Q = \begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix}$, *and* $R = \begin{bmatrix} 5 & 0.15 \\ 0 & 0.2 \end{bmatrix}$. *R stretches the x component by a factor of five and shears y components in the x direction and shrinks them by a factor of five with a net effect of preserving areas. Q simply rotates the stretched figure. Each parallelepiped in the bottom row is constructed from the columns of the corresponding matrix in the middle row. The algebraic formulas for those vectors appear in the top row. Note that R determines the changes in length and area and that Q does not affect either.*

Since $\prod_{n=1}^{N} r(n)$ is upper triangular, I have the QR factorization

$$D\Phi(x(0), N\Delta\tau) = Q(N)R(N), \tag{5.42}$$

$$R(N) = \prod_{n=1}^{N} r(n). \tag{5.43}$$

And since $R_{i,i}(N) = \prod_{n=1}^{N} r_{i,i}(n)$,

$$\log\left(\left|R_{i,i}(n)\right|\right) = \sum_{n=1}^{N} \log\left(\left|r_{i,i}(n)\right|\right). \tag{5.44}$$

Substituting this result into (5.40) constitutes the Benettin procedure. The action of a matrix on a unit square is factored into components Q and R and sketched in Fig. 5.4. Results of applying the procedure to the Lorenz system appear in Fig. 5.5.

5.5 A Practical Performance Bound

Consider the following two cases:

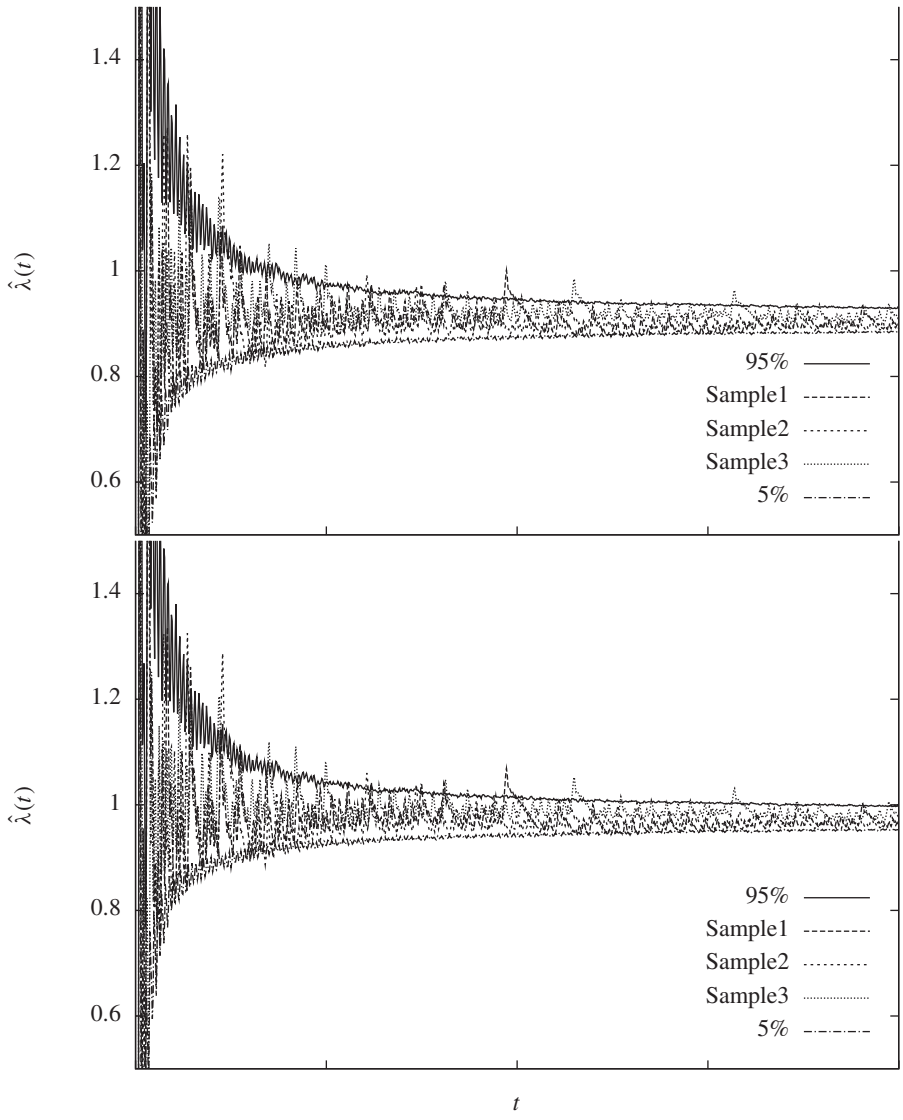


Figure 5.5. *Lyapunov exponent calculation for the Lorenz system. In the upper part, the three lighter traces are plots of* $\frac{1}{T}\sum_{t=1}^{T}\log\left(\left|r_{1,1}(t)\right|\right)$*, and the heavier traces the* 5% *and* 95% *limits on* 1,000 *separate runs. The lower part is the same, except that* $\left|r_{1,1}(t)\right|$ *is augmented by a noise term with amplitude* $\frac{\sigma_\eta}{\Delta} = 0.01$ *(see* (5.49)*). The shapes of the traces are almost unchanged, except for a uniform shift up of about* 0.03*. I conclude that a test model that is the same as the generating model, except that the state noise is* $\frac{\sigma_\eta}{\Delta} = 0.01$*, would have a cross entropy of about* 0.936 *nats, while the largest Lyapunov exponent is* $\hat{\lambda}_1 \approx 0.906$ *nats.*

- state space dynamics perturbed by noise;
- simulated dynamics perturbed by numerical truncation.

The definition of Kolmogorov–Sinai entropy in the two cases yields extremely different answers. If the perturbations are random noise, then the supremum of $h(\mathcal{B}, F, \mu)$ over $\mathcal{B}$ does not exist and h_{KS} is unbounded. On the other hand, if the perturbations are numerical truncation and the process is a digital simulation, then all observation sequences converge to periodic cycles and $h_{KS} = 0$. Thus, the *strict* definition of the Kolmogorov–Sinai entropy is useless as a bound on the cross entropy of models in numerical simulations. Here I argue, however, that numerical Lyapunov exponent estimates nonetheless provide a *practical* reference for the performance of models.

When working on a new model building procedure that takes *training* samples $\left\{y_{\tau_1}^{T_1}, y_{\tau_2}^{T_2}, \ldots, y_{\tau_N}^{T_N}, \right\}$ and produces a family of conditional probability functions $P\left(y(t)|y_1^{t-1}, \theta\right)$ with parameters θ, I recommend numerically estimating the *entropy gap* (see Theorem 5.2) $\delta_{\mu||\theta} = h(\mathcal{B}, F, \mu||\theta) - \sum_{i:\lambda_i>0} \lambda_i$ to characterize the fidelity of the resulting models θ to generating processes. As a debugging tool, it is reasonable to choose some parameters θ' for a model class, use that model to generate training data, and then verify that as the size of the training data set increases the proposed model building procedure recovers the parameters θ'. However, such a test fails to consider how well the proposed procedure and model class work on the realistic case of data generated by processes outside the model class. Even though the test I propose does not provide *correct* model parameters against which to compare fitted parameters, it does provide a reference against which to compare model performance. Specifically, I advocate the following numerical experiment for evaluating a model building procedure:

1. Use a numerical simulation of a chaotic dynamical system to generate training data and testing data. For simplicity, consider a system with a single positive exponent λ_1.
2. Quantize the data to a few thousand levels.
3. Run the Benettin procedure on the system, to estimate Lyapunov exponents.
4. Substitute the estimated exponents into Ledrappier and Young's formula, (5.26), with $\gamma_1 = 1$ to get $\hat{h}(F, \mu) = \hat{\lambda}_1$, an estimated entropy rate. If the partition $\mathcal{B}$ is fine enough, $\hat{h}(\mathcal{B}, F, \mu) = \hat{\lambda}_1$ will be a good estimate.
5. Produce $P_{*|\theta}$ by applying the new model building procedure to the training data.
6. Estimate the cross entropy by evaluating the likelihood of the model on long sequences of testing data:

$$\hat{h}(\mathcal{B}, F, \mu||\theta) = \frac{-1}{T} \sum_{t=1}^{T} \log\left(P\left(y(t)|y_1^{t-1}, \theta\right)\right). \tag{5.45}$$

7. Calculate an entropy gap by subtracting the two estimates:

$$\hat{\delta}_{\mu||\theta} = \hat{h}(\mathcal{B}, F, \mu||\theta) - \hat{h}(\mathcal{B}, F, \mu).$$

   For an optimal model, expect the gap to be zero. If the gap is much larger than zero, conclude that the new procedure is suboptimal.

The test is reasonable only if, subject to some constraints, $\mathbb{E}\hat{\delta}_{\mu||\theta} \geq 0$ is a tight bound and the variance of $\hat{\delta}_{\mu||\theta}$ is small. Below, I argue that a model that uses knowledge of the generating process and has smooth probability densities in state space achieves the bound with equality, and thus the bound is tight.

In this class of models, the probability measure for the generating process is not necessarily ergodic or even stationary; it is derived from a uniform density over a box that covers possible starting conditions, and it includes a little bit of noise in the dynamics so that even in the long time limit it does not become fractal. Because the probability is smooth, the models cannot exploit fractal properties that might exist in the system being modeled, and consequently γ, the Ledrappier and Young correction to the Pesin formula, is irrelevant. More specifically, I consider a model class with the following properties:

Probability density. The probability density for the initial state $P(x(1)|\theta)$ is a uniform distribution on a cube in X that has length $L_{\text{i.c.}}$ on each side.

State noise. The model has noise in the state dynamics,

$$x(t+1) = F(x(t)) + \eta(t), \tag{5.46}$$

where $\eta(t)$ are i.i.d. Gaussian with $\eta(t) \sim \mathcal{N}(0, \mathbf{I}\sigma_\eta^2)$. I suppose that F is the same as the *true* function of the system being *modeled* but that noise in the system being modeled is smaller or zero.

Measurement function. I let the measurement function be the same for the model as for the true system, i.e., a discrete partition with resolution Δ. I have in mind a uniform quantization of a single component of X such as used for Fig. 5.3.

The only difference between the true system and the model is that the state noise in the model may be larger than the state noise in the true system. With this framework, I can draw samples randomly from a true distribution $P_{*|\mu}$ and consider model probabilities $P_{*|\theta}$ without having to find a stationary distribution.

In sidestepping the key issue of a stationary distribution, I have sacrificed ergodicity, which is the basis of the definition of a Lyapunov exponent as a global property. Empirically, however, the convergence of the Benettin procedure is similar for almost any initial condition (see Fig. 5.5). Defining the Lyapunov exponent estimate for a given initial condition x_0 and duration t as

$$\lambda(x_0, \tau) \equiv \frac{1}{t} \log \left(\prod_{t=1}^{\tau} r_{1,1}(x_0, t) \right)$$

and the *range* as the smallest interval $\hat{\lambda} \pm \epsilon$ such that for almost every initial condition x_0 there is an integer τ such that for every $t \geq \tau$

$$\hat{\lambda} - \epsilon \leq \lambda(x_0, t) \leq \hat{\lambda} + \epsilon,$$

I can characterize the empirical observation from Fig. 5.5 (and similar plots for longer times)[29] by saying that $\hat{\lambda} = 0.906 \pm 0.001$.

[29] Many others have estimated the Lyapunov exponents of the Lorenz attractor. For example, Viswanath [53] suggests that $\lambda_1 = 0.905630 \pm 0.00005$.

In the limit of small noise $\sigma_\eta \to 0$, one might calculate $P(y_1^T|\theta)$ for any sequence of observations as the probability of the set of initial conditions that are consistent with y_1^T, i.e., the preimage of y_1^T:

$$P(y_1^T|\theta) = \int_{\{x: Y_1^T(x)=y_1^T\}} P(x|\theta)dx.$$

The volume of such preimages is typically in the range[30]

$$\frac{\Delta e^{-\hat{\lambda}T}}{O(T)} < \text{Vol} < \Delta e^{-\hat{\lambda}T},$$

and since the density of initial conditions is smooth, for large T one finds that

$$\frac{1}{T}\log\left(P(y_1^T|\theta)\right) \approx -\hat{\lambda}. \tag{5.47a}$$

Rather than going backwards in time to analyze the preimage of y_1^T, one can think about the forward image of the volume of initial conditions under the map $\Phi(T)$. To first order, the distribution is a uniform probability density over a parallelepiped that extends a distance $L_{\text{i.c.}}e^{T\lambda(x_0,T)}$ in the direction of the first column of the orthogonal matrix $Q(t)$ in (5.42). The measurement partition divides the image into elements that have a characteristic size of Δ, again yielding

$$\frac{1}{T}\log\left(P(y_1^T|\theta)\right) \approx -\hat{\lambda}. \tag{5.47b}$$

Typically, the stretching is so large that the image of the volume of allowed initial conditions does not resemble a parallelepiped, but the exponential nature of the stretching is all that matters, and in the small noise limit

$$h(\mathcal{B}, F, \mu||\theta) \approx \hat{\lambda}. \tag{5.48}$$

The effect of state noise, $\sigma_\eta > 0$, on (5.48) depends on the size of the noise compared to the quantization, i.e., $\frac{\sigma_\eta}{\Delta}$, the expansion rate, and the uniformity of the expansion rate. At each time step, the noise term σ_η in the model roughly augments the stretching in each state space direction by an amount $\frac{\sigma_\eta}{\Delta}$. One can estimate an upper bound on the total effect by replacing $\left|r_{i,i}(n)\right|$ with $\left|r_{i,i}(n)\right| + \frac{\sigma_\eta}{\Delta}$ for each i and n in (5.44) of the Benettin procedure, i.e.,

$$\hat{\lambda}_{\text{aug},i} \equiv \frac{1}{N}\sum_{n=1}^{N}\log\left(\left|r_{i,i}(n)\right| + \frac{\sigma_\eta}{\Delta}\right). \tag{5.49}$$

Notice that knowledge of $\hat{\lambda}_i$ and $\frac{\sigma_\eta}{\Delta}$ is not sufficient to calculate $\hat{\lambda}_{\text{aug},i}$. If the stretching were uniform with $\left|r_{i,i}(n)\right| = e^{\lambda}$ for all n, the augmented result would be $\hat{\lambda}_{\text{aug},i} = \log\left(e^{\lambda} + \frac{\sigma_\eta}{\Delta}\right)$,

[30] Since the partition boundaries induced by all but the first few observations are almost perpendicular to the unstable direction, they rarely intersect. The last observation, $y(T)$, induces boundaries with a spacing of about $\Delta e^{-\hat{\lambda}T}$. Although earlier observations serve primarily to distinguish the multiple preimages of the last observation, they may also further subdivide those preimages. The number of such subdivisions is much less than T.

but the result increases without bound[31] as $\left|r_{i,i}(n)\right|$ varies more with n. In Fig. 5.5, I compare $\hat{\lambda}_{\text{aug},1}$ with $\hat{\lambda}_1$ for the Lorenz system. For noise with an amplitude $\frac{\sigma_\eta}{\Delta} = 0.01$, the figure indicates an augmentation of $\hat{\lambda}_{\text{aug},1} - \hat{\lambda}_1 \approx 0.03$, which, although roughly 10 times the augmentation that uniform stretching would produce, is still a small effect. From the figure, I conclude that the Benettin procedure produces a robust practical upper bound on model performance.

5.6 Approaching the Bound

Although the *slope* of the plot in Fig. 5.3 (the log likelihood per time step attained by extended Kalman filters) matches the entropy bound, and the explanation of the nonzero intercept seems adequate, an example of a model with a likelihood close to the bound without any offset would be more satisfying. To find such a model, I returned to the source of coarsely quantized Lorenz observations that I used for Fig. 1.9 in the introduction. That figure illustrates the association of each of the 12 discrete hidden states of an HMM with particular regions in $\mathbb{R}^3$, the state space of the Lorenz system. Recall that I generated the observations by integrating the Lorenz system with a time step of $\tau_{\texttt{sample}} = 0.15$ and quantized the first component into one of four levels. Although the cross entropy of that 12 state model is not very close to the bound based on the Lyapunov exponent estimate ($\hat{h}(\mathcal{B}, F, \mu) = \hat{\lambda}_1 = 0.136$ nats $= 0.196$ bits), it seems plausible that, by using HMMs with more states, I might get higher likelihoods. In fact it is true, but it requires a surprisingly large number of states.

My first attempt was to train a sequence of models with ever more hidden states. I initialized each model randomly and ran many iterations of the Baum–Welch algorithm on quantized observations. Even with many iterations, I did not build any promising models.

In my second attempt, I exploited my knowledge of the Lorenz dynamics in $X = \mathbb{R}^3$ as follows:

1. Generate training data x_1^T and y_1^T by integrating the Lorenz system. Here $x(t)$ is a point in the original state space, and $y(t)$ is a quantized observation that can take one of four values.

2. Generate testing data y_{T+1}^{T+N} by continuing the integration.

3. Find a set of discrete states $\{s_1, s_2, \ldots, s_m\}$ by partitioning the original space with a uniform grid of resolution Δ_x. I constructed states only for those partition elements that were occupied by at least one member of x_1^T.

4. Build an initial HMM using the training sequence. I set the state transition probabilities by counting the frequency with which the partitioned X sequence made each possible transition. Similarly, I set the observation model by counting the frequency with which each partition element was associated with each possible observation.

[31] If, for example,

$$\left|r_{i,i}(n)\right| = \begin{cases} \delta^{N-1} e^{\lambda}, & n = 0, \\ \frac{1}{\delta} e^{\lambda} & \text{otherwise,} \end{cases}$$

then $\hat{\lambda}_i = \lambda$, but $\lim_{\delta \to 0} \hat{\lambda}_{\text{aug},i} = \infty$.

5. Estimate the cross entropy ($\hat{h}(\mathcal{B}, F, \mu||\theta)$; see (5.45)) of the initial model by calculating its log likelihood per time step on the testing data.

6. Improve the model with several Baum–Welch iterations.

7. Estimate the cross entropy of the trained model by calculating its log likelihood per time step on the testing data.

As hoped, I found that as I reduced the resolution, the number of states increased and the cross entropy estimates decreased. I found, however, that the computational cost of training models with large numbers of states was prohibitive. Although the cross entropy estimates in step 7 were lower than the estimates in step 5, it was computationally cheaper to attain any given cross entropy value by simply reducing the state partition resolution enough, omitting steps 6 and 7, and using the estimate from step 5.

Since there is no training, i.e., Baum–Welch iterations, in this abbreviated procedure, I could calculate the likelihood with a variant of the forward algorithm that does not store or return α or γ values for the entire sequence. In fact, given typical observations y_1^t up to time t, only a small fraction of the states have nonzero probability. Code that exploits these features is many orders of magnitude cheaper than code for the vanilla forward algorithm.

Results of the abbreviated procedure appear in Fig. 5.6. For the rightmost point on the curve I found a model with 1,490,202 hidden states and a cross entropy of 0.153 nats

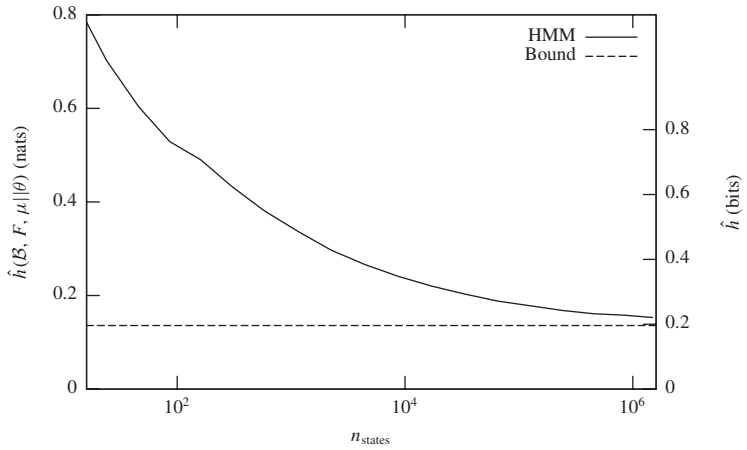
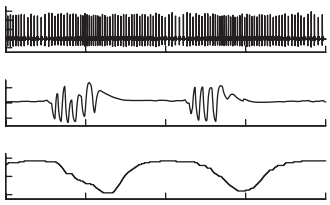


Figure 5.6. *Entropy gap, $\hat{\delta}_{\mu||\theta}$, vs. number of states in HMMs. The upper trace plots estimates of cross entropy $\hat{h}(\mathcal{B}, F, \mu||\theta)$ for a sequence of HMMs vs. the number of discrete states in the models. I built the models using actual Lorenz state space trajectories as described in the text. The lower trace is an estimate of the entropy rate, $\hat{h}(F, \mu) = \hat{\lambda}_1$, of the true process based on Lyapunov exponents estimated by the Benettin procedure. The distance between the curves is the* entropy gap $\hat{\delta}_{\mu||\theta}$. *The gap seems to be going to zero, suggesting that an HMM with enough states might perform at least as well as any other model based on any other technology. Each model was built using the same sample trajectory of* 8,000,000 *points in the original state space, and the cross entropy estimates are based on a test sequence of* 10,000 *observations.*

or 0.221 bits. By connecting this model to a simple data compression routine, one could compress the test data (or presumably any other long sequence from the source) down to 0.221 bits per sample, which is 13% more than the 0.196 bits per sample that the Lyapunov exponent estimate suggests is possible. Address space limits of 32-bit Python made it difficult to build larger models and extend the plot to the right.



Chapter 6
Obstructive Sleep Apnea

The challenge for the *Computers in Cardiology* meeting in 2000 (CINC2000) was a competition to identify *obstructive sleep apnea* on the basis of ECGs alone. My colleague James McNames at Portland State University and I won the prize for the best minute by minute analysis of data. I prepared our first entry using HMMs, and James prepared the subsequent (and winning) entries by hand using spectrograms[32] to visualize the data. In preparing the first entry, I discovered that using a sequence of states from the Viterbi algorithm to deduce a sequence of classifications produced obvious errors. After some thought, I realized that a variant of the Viterbi algorithm that directly decoded sequences of *classifications* rather than sequences of *states* would yield better classification results. Although I did not implement such an algorithm in 2000, I have done so in preparing this book.

While the more appropriate algorithm does not perform as well as James' eye, it does perform well enough to place among the top scores that appear on the PhysioNet Web site.[33] In this final chapter, which is available as technical report LA-UR-07-1969 from Los Alamos National Laboratory, I will describe the algorithm for decoding sequences of classification and narrate in a less formal style of presentation my attempt at using it to *re-win* the contest.

6.1 The Challenge and the Data

The PhysioNet Web site[34] announced the challenge and described the data as follows:

Introduction: *Obstructive sleep apnea (intermittent cessation of breathing) is a common problem with major health implications, ranging from excessive daytime drowsiness to serious cardiac arrhythmias. Obstructive sleep apnea is associated with increased risks of high blood pressure, myocardial infarction, and stroke, and with increased mortality rates. Standard methods for detecting and quantifying sleep apnea are based on respiration monitoring, which often disturbs or interferes with sleep and is generally expensive. A*

[32] A spectrogram is display of power in Fourier spectral bands as a function of time. The x-axis is time, the y-axis is frequency, and the image intensity at point (t, f) is the power in that frequency estimated in a window centered at that time.

[33] `http://www.physionet.org/challenge/2000/top-scores.shtml`.

[34] `http://www.physionet.org/challenge/2000`.

number of studies during the past 15 *years have hinted at the possibility of detecting sleep apnea using features of the electrocardiogram. Such approaches are minimally intrusive, inexpensive, and may be particularly well-suited for screening. The major obstacle to use of such methods is that careful quantitative comparisons of their accuracy against that of conventional techniques for apnea detection have not been published.*

We therefore offer a challenge to the biomedical research community: demonstrate the efficacy of ECG-based methods for apnea detection using a large, well-characterized, and representative set of data. The goal of the contest is to stimulate effort and advance the state of the art in this clinically significant problem, and to foster both friendly competition and wide-ranging collaborations. We will award prizes of US$500 *to the most successful entrant in each of two events.*

Data for development and evaluation: *Data for this contest have kindly been provided by Dr. Thomas Penzel of Philipps-University, Marburg, Germany [available on the website].*

The data to be used in the contest are divided into a learning set and a test set of equal size. Each set consists of 35 *recordings, containing a single ECG signal digitized at* 100 *Hz with* 12*-bit resolution, continuously for approximately 8 hours (individual recordings vary in length from slightly less than* 7 *hours to nearly* 10 *hours). Each recording includes a set of reference annotations, one for each minute of the recording, that indicate the presence or absence of apnea during that minute. These reference annotations were made by human experts on the basis of simultaneously recorded respiration signals. Note that the reference annotations for the test set will not be made available until the conclusion of the contest. Eight of the recordings in the learning set include three respiration signals (oronasal airflow measured using nasal thermistors, and chest and abdominal respiratory effort measured using inductive plethysmography) each digitized at* 20 *Hz, and an oxygen saturation signal digitized at* 1 *Hz. These additional signals can be used as reference material to understand how the apnea annotations were made, and to study the relationships between the respiration and ECG signals. [...]*

Data classes: *For the purposes of this challenge, based on these varied criteria, we have defined three classes of recordings:*

Class A (Apnea): *These meet all criteria. Recordings in class A contain at least one hour with an apnea index of* 10 *or more, and at least* 100 *minutes with apnea during the recording. The learning and test sets each contain* 20 *class A recordings.*

Class B (Borderline): *These meet some but not all of the criteria. Recordings in class B contain at least one hour with an apnea index of* 5 *or more, and between* 5 *and* 99 *minutes with apnea during the recording. The learning and test sets each contain* 5 *class B recordings.*

Class C (Control): *These meet none of the criteria, and may be considered normal. Recordings in class C contain fewer than* 5 *minutes with apnea during the recording. The learning and test sets each contain* 10 *class C recordings.*

Events and scoring: *Each entrant may compete in one or both of the following events:*

1. Apnea screening: *In this event, your task is to design software that can classify the* 35 *test set recordings into class A (apnea) and class C (control or normal) groups, using the ECG signal to determine if significant sleep apnea is present. [...]*

2. Quantitative assessment of apnea: *In this event, your software must generate a minute-by-minute annotation file for each recording, in the same format as those provided with the learning set, using the ECG signal to determine when sleep apnea occurs. Your annotations will be compared with a set of reference annotations to determine your score. Each annotation that matches a reference annotation earns one point; thus the highest possible score for this event will be approximately* 16800 *(*480 *annotations in each of* 35 *records). It is important to understand that scores approaching the maximum are very unlikely, since apnea assessment can be very difficult even for human experts. Nevertheless, the scores can be expected to provide a reasonable ranking of the ability of the respective algorithms to mimic the decisions made by human experts.*

6.1.1 The Data

Briefly, one can fetch the following records from PhysioNet:

a01–a20: The *a records* from individuals that display *apnea*

b01–b05: The *b records*[35] from individuals diagnosed as *borderline*

c01–c10: The *c records* from *control* or normal individuals

a01er–a04er and b01er and c01er: Identical to $a01$–$a04$ and $b01$ and $c01$ except augmented with respiration and SpO_2 (percent of arterial hemoglobin saturated with oxygen) signals

summary_of_training: Expert classifications of each minute in the a, b, and c records

x01–x35: The test set;[36] records without classification

In 2000, using both the data and software to view it from PhysioNet, I saw striking oscillations in the apnea time series. The patients stop breathing for tens of seconds, gasp a few breaths, and stop again. Each cycle takes about 45 seconds and can go on for most of the night. I have plotted two periods of such an oscillation from record $a03$ in Fig. 6.1. The reference or *expert* classifications provided with the data indicate that these are the last oscillations in the first apnea episode of the night. Over the entire night's record of 8 hours and 39 minutes, the patient had apnea for a total of 4 hours and 5 minutes in 11 separate episodes. In that time, more than once a minute, he was waking up enough to start breathing. Ten and a half minutes after the end of the first apnea episode, the record, as plotted in Fig. 6.2, looks normal.

[35] The amplitude of the $b05$ record varies dramatically over different segments of time. I found it unusable and discarded it entirely.

[36] The records $x33$ and $x34$ are so similar that I suspect they are simultaneous recordings from different ECG leads. I did not explicitly exploit the similarity in my analysis.

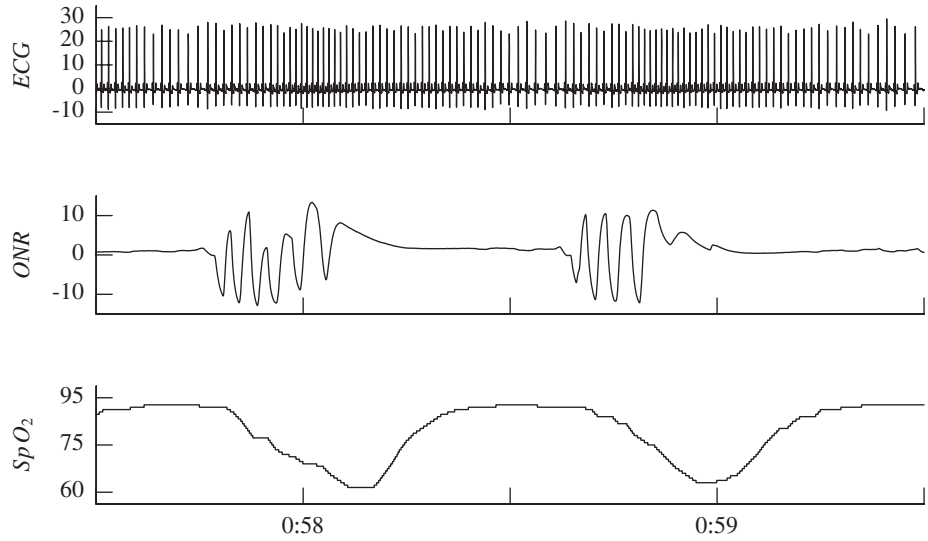


Figure 6.1. *A segment of record a*03*. Two cycles of a large apnea induced oscillation in SpO_2 are drawn in the lower plot. The middle plot is the oronasal airflow signal, and the upper plot is the ECG (units of both ONR and ECG are unknown). The time axis is marked in* hours:minutes*. Notice the increased heart rate just after* 0:58 *and just before* 0:59*.*

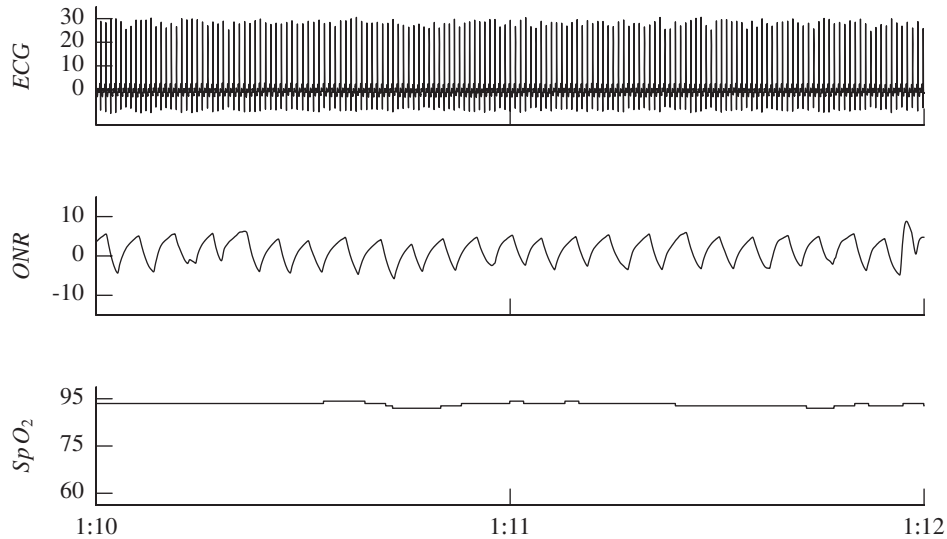


Figure 6.2. *A segment of record a*03 *taken during a period of normal respiration. The signals are the same as in Fig.* 6.1*.*

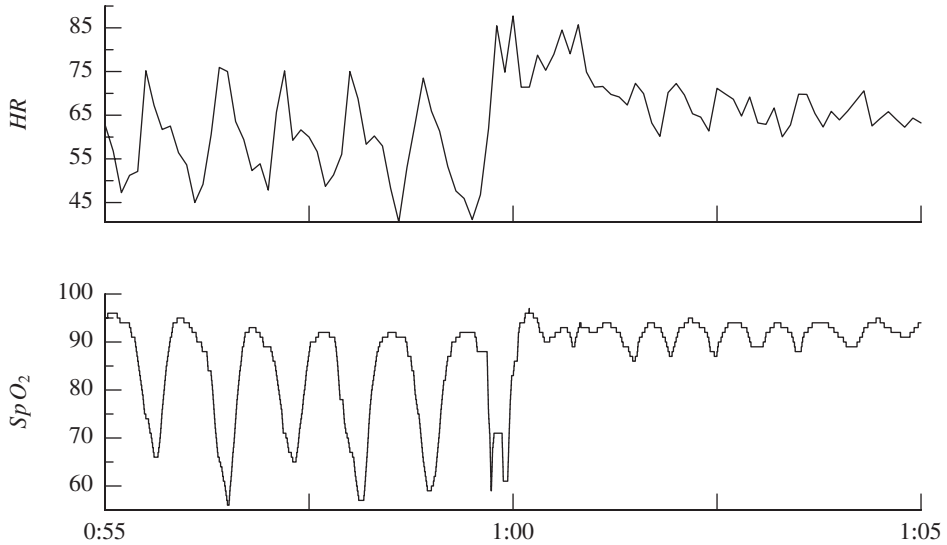


Figure 6.3. *A segment of record a*03 *at the end of an episode of apnea with indications in both the SpO_2 signal and the heart rate* (HR) *signal. The expert marked the time before* 1:00 *as apnea and the time afterwards as normal.*

In plots like Fig. 6.1, the heart rate visibly increases at the end of the gasping phase and then decreases during the phase of interrupted respiration. That heart rate oscillation is the key I used in my initial attempts to classify periods of apnea. In Fig. 6.3, I have plotted both a heart rate derived from the ECG and the SpO_2 signal. The oscillations in the signals track each other, and the expert classifies only the region of large heart rate oscillation as apnea.

6.2 First Classification Algorithms and Two Useful Features

6.2.1 Using Information from Experts to Train

I first thought about the CINC2000 challenge when Andreas Rechtsteiner told me that he was going to use it as a final project for a class that James was teaching. I suggested that he try a two state HMM with autoregressive observation models, i.e., a model like those described in Section 3.1.4 but with scalar observations. The performance of Andreas' two state model was not memorable.

To use the expert classification information in training, Andreas and I found that we need only modify the observation model. The technique is not limited to HMMs with only two states or two classes; it applies to an arbitrary number of classes and to arbitrary numbers of states associated with each class. At each time t, let $c(t)$ denote classification information from an expert about which states are possible. Specifically $c(t)$ is a vector

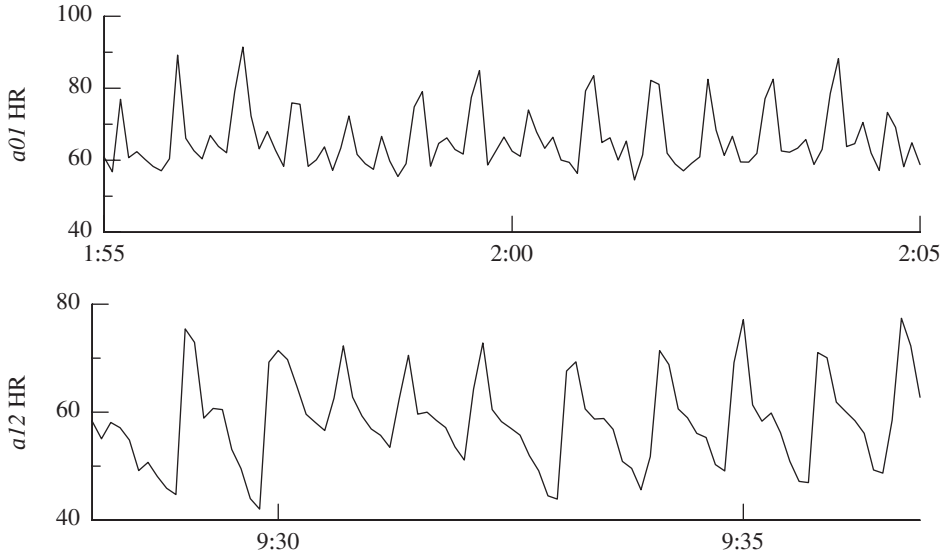


Figure 6.4. *Nonlinear effects: The upper plot seems to be a period two oscillation. The lower plot is approximately sawtooth.*

that indicates that some states are possible and that the others are impossible. We simply replaced $P_{Y(t)|S(t),Y_1^{t-1}}$ with $P_{Y(t),C(t)|S(t),Y_1^{t-1}}$ wherever it occurs in the Baum–Welch algorithm. Roughly, the modification forces the system into states associated with the right class during training.

6.2.2 Nonlinear Dynamics

After Andreas finished his class, I looked more carefully at heart rate time series and the classification errors of a two state model. I saw evidence of *nonlinear dynamics* such as the period two waveform and the sawtooth in Fig. 6.4. Superpositions of sinusoids can describe such patterns with the right amplitudes and relative phases of the harmonics. Although a linear model can favor the correct frequencies and amplitudes of the component sinusoids, linear models cannot detect or enforce phase relationships. The ability to distinguish phase relationships makes it possible to build nonlinear models that are more discriminating than any linear model.

Excited by these observations, I built an HMM with many chains of states. Each chain modeled a different form of heart rate oscillation. I trained the model on the expertly classified records and then used the model to classify those same records. For each record, I used the Viterbi algorithm to find the MAP state sequence given the observations. Then I derived a classification sequence from the decoded state sequence by selecting class C at time t if the decoded state satisfied $s(t) \in C$. The technique performed terribly. The sequences of

classifications said every minute was *normal* even though (actually *because*) I had modeled the different kinds of *apnea* oscillations so carefully.

An analysis of the values of the conditional probabilities of states given the entire sequence of observations, $P_{S(t)|Y_1^T}\left(s_i|y_1^T\right) = \alpha(s_i,t)\beta(s_i,t)$, revealed the problem and suggested a solution. Because the model had many more apnea states than normal states, the state probability was rarely concentrated enough in a single apnea state for that state to be decoded. To address the problem, I solved for the sequence of MAP classifications, i.e.,

$$\tilde{C}(t) \equiv \underset{C}{\operatorname{argmax}} \sum_{i:s_i \in C} P_{S(t)|Y_1^T}\left(s_i|y_1^T\right), \tag{6.1}$$

instead of solving for the MAP sequence of states.

In general if there are more than two classes, (6.1) can yield unlikely or even impossible sequences of classes. (See the example in Section 2.3.2.) Section 6.3 describes an algorithm that rather than finding the *sequence MAP classifications* finds the *MAP sequence of classifications*, i.e.,

$$\hat{C}_1^T \equiv \underset{C_1^T}{\operatorname{argmax}} \sum_{s_1^T : s(t) \in C(t), 1 \le t \le T} P\left(s_1^T|y_1^T\right). \tag{6.2}$$

Using (6.1) and the HMM with many chains of states, I classified the test data to prepare a contest entry. When James submitted the entry we found that it classified 78.91% of the minutes correctly.

6.2.3 The Excellent Eye of Dr. McNames

To diagnose the errors, James printed spectrograms of every record. Although the spectrograms discarded the phase information that I hoped was important, they clearly indicated the oscillations that I had tried to capture in my complicated HMMs as intense bands of power at frequencies below 2 cpm (cycles per minute). In addition to those low frequency oscillations, James noticed bands of power between 10 and 20 cpm in the spectrograms (see Fig. 6.5). That higher frequency power was evidence of respiration. Using both of those features, he classified the test data by hand. First, he classified each entire record for *event* 1. Since almost none of the minutes in the normal records is apnea, he classified each minute in those records as normal. His third attempt at *event* 2 was the best entry at the close of the contest.

The question that motivated the contest is, "Is the information in an ECG alone sufficient to classify apnea?" James' work answered the question affirmatively. For attempts to do the classification automatically, his work suggests the following points:

1. Classify each entire record first. Then, using that classification, classify each minute in the record.

2. Heart rate oscillations at about 1.3 cpm indicate apnea.

3. A clean phase jitter signal at about 14 cpm indicates normal respiration.

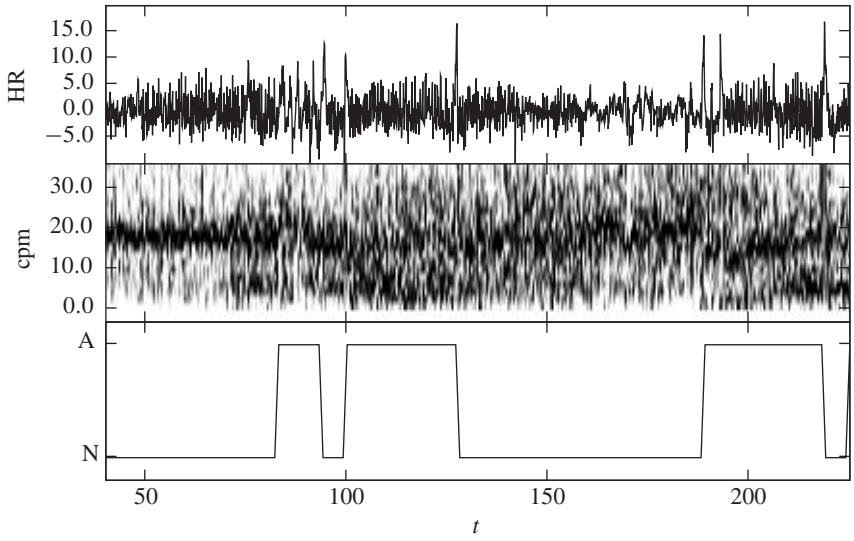


Figure 6.5. *Information about respiration in high frequency phase variations. This is the a*11 *record roughly between minutes* 40 *and* 225*. The upper plot is heart rate (bandpass filtered* 0.09–3.66 *cpm), the middle plot is a spectrogram of the phase jitter in the heart rate, and the lower plot is the expert classification. A single band of spectral power between about* 10 *and* 20 *cpm without much power below the band in the spectrogram indicates normal respiration.*

6.3 Decoding Sequences of Classifications

In this section, I present a modified Viterbi algorithm that finds the MAP sequence of classifications given a model and a sequence of observations. Before a brief discussion, let me introduce the following notation and equations.

Notation

S^* The set of state sequences of arbitrary length. Likewise, C^* is the set of classification sequences of arbitrary length.

$g : S^* \times C^* \mapsto \{0, 1\}$ A masking function, $g(s_1^t, c_1^t) = 1$ is equivalent to "s_1^t is consistent with c_1^t." The principal application is

$$P\left(y(t), c(t)|s(t)\right) = P\left(y(t)|s(t)\right) g(s(t), c(t)).$$

$\hat{c}_1^t(c)$ The best classification sequence ending in c:

$$\hat{c}_1^t(c) \equiv \underset{c_1^t : c(t)=c}{\operatorname{argmax}} P\left(c_1^t, y_1^t\right).$$

$v(t, c)$ The *utility* of the best sequence ending in c:

$$v(t, c) \equiv \log\left(P\left(y_1^t, \hat{c}_1^t(c)\right)\right).$$

$f(t+1, s, c')$ The probability of the state s at time $t+1$ and output $y(t+1)$ given class c' at time t, the best classification sequence leading to c', and all previous outputs:

$$f(t+1, s, c') \equiv P\left(s, y(t+1)|y_1^t, \hat{c}_1^t(c')\right).$$

$\phi(t, s, c)$ The probability of state s at time t given the best classification sequence ending in c at time t and y_1^t:

$$\phi(t, s, c) \equiv P\left(s|y_1^t, \hat{c}_1^t(c)\right).$$

$c'(c, t+1)$ The best $c(t)$ given $c(t+1) = c$:

$$c'(c, t+1) \equiv \underset{\bar{c}}{\operatorname{argmax}}\, P\left(y_1^{t+1}, \hat{c}_1^t(\bar{c}), c\right).$$

Equations

$$\hat{c}_1^t(c) = \underset{c_1^t : c(t)=c}{\operatorname{argmax}} \sum_{s_1^t} g(s_1^t, c_1^t) P\left(s_1^t, y_1^t\right), \tag{6.3}$$

$$\hat{c}_1^{t+1}(c) = \left(\left[\hat{c}_1^t(c'(c, t+1))\right], c\right), \tag{6.4}$$

$$f(t+1, s, c') = \sum_{s'} \phi(t, s', c') P(s|s') P(y(t+1)|s), \tag{6.5}$$

$$v(t+1, c) = v(t, c'(c, t+1)) + \log\left(\sum_s g(s, c) f(t+1, s, c'(c, t+1))\right), \tag{6.6}$$

$$\phi(t+1, s, c) = \frac{f(t+1, s, c'(c, t+1))}{\sum_{\bar{s}} g(\bar{s}, c) f(t+1, \bar{s}, c'(c, t+1))}. \tag{6.7}$$

Equation (6.4) says that the best classification sequence ending in c at time $t+1$ consists of c concatenated with the best sequence ending in c' at time t, where c' is the best predecessor of c.

Equation (6.5) follows from the model assumptions

$$P(s|s', y_1^t, \hat{c}_1^t(c')) = P(s|s')$$

and

$$P(y(t+1)|s, s', y_1^t, \hat{c}_1^t(c')) = P(y(t+1)|s)$$

and Bayes' rule.

To derive (6.7) use[37]

$$f(t+1,s,c') = \frac{P(s,\, y_1^{t+1},\, c_1^t)}{P(y_1^t,\, c_1^t)}, \tag{6.8}$$

$$\sum_{\bar{s}} g(\bar{s},c)\, f(t+1,\bar{s},c'(c,t+1)) = \frac{P(y_1^{t+1},\, c_1^{t+1})}{P(y_1^t,\, c_1^t)}, \tag{6.9}$$

$$P(c|s,\, y_1^{t+1},\, c_1^t) = g(s,c) \tag{6.10}$$

$$= \frac{P(s,\, y_1^{t+1},\, c_1^{t+1})}{P(s,\, y_1^{t+1},\, c_1^t)} \tag{6.11}$$

and note that

$$\begin{aligned}\frac{f(t+1,s,c')}{\sum_{\bar{s}} g(\bar{s},c)\, f(t+1,\bar{s},c')} P(c|s,\, y_1^{t+1},\, c_1^t) &= \frac{P(s,\, y_1^{t+1},\, c_1^t)\; P(y_1^t,\, c_1^t)\; P(s,\, y_1^{t+1},\, c_1^{t+1})}{P(y_1^t,\, c_1^t)\; P(y_1^{t+1},\, c_1^{t+1})\; P(s,\, y_1^{t+1},\, c_1^t)} \\ &= P(s|y_1^{t+1},\, c_1^{t+1}) \equiv \phi(t+1,s,c).\end{aligned}$$

Algorithm

The algorithm makes a single pass through the sequence of observations going forward in time. To move from time t to time $t+1$, consider each possible classification c_{next}, and find the best predecessor classification c_{best} at time t. Then collect and store the best classification sequence ending in that classification, its log probability, and the conditional probabilities of the hidden states given that classification sequence. At the last time step T, each possible final classification $C(T)$ will be associated with the best sequence ending in that classification and the log likelihood of that sequence. The algorithm simply selects the sequence with the highest log likelihood. Fig. 6.6 provides pseudocode for the algorithm.

6.4 Assembling the Pieces

As a final example for this book, I chose to revisit the CINC2000 challenge by combining the following pieces:

Low frequency oscillations of heart rate: Oscillations like those before 1:00 in the upper plot of Fig. 6.3 indicate apnea.

High frequency phase modulation: Strong bands of spectral power between 10 and 20 cpm indicate normal respiration.

[37] I use the following abbreviations in this derivation:

$$\begin{aligned} c'(c,t+1) &\Rightarrow c', \\ \hat{c}_1^t(c') &\Rightarrow c_1^t, \\ \hat{c}_1^{t+1}(c) &\Rightarrow c_1^{t+1}. \end{aligned}$$

Initialize:
    for each c
        $v_{\text{next}}(c) = \log\left(\sum_s g(s, C) P_{Y(1),S(1)}(y(1), s)\right)$

Iterate:
    for t from 1 to T
        Swap $v_{\text{next}} \leftrightarrow v_{\text{old}}$
        for each c_{next}

            # Find best predecessor
            $c_{\text{best}} = \text{argmax}_{c_{\text{old}}}\left(v_{\text{old}}(c_{\text{old}}) + \log\left(\sum_s g(s, c_{\text{best}}) f(t+1, s, c_{\text{best}})\right)\right)$

            # Update v
            $v_{\text{next}}(c_{\text{next}}) = v_{\text{old}}(c_{\text{best}}) + \log\left(\sum_s g(s, c_{\text{best}}) f(t+1, s, c_{\text{best}})\right)$

        # Update predecessor array
        Predecessor$[c_{\text{next}}, t] = c_{\text{best}}$

        # Update ϕ
        for s in c_{next}
            Assign $\phi_{\text{next}}(s, c_{\text{next}})$ using Eqn. (6.7)

Backtrack:
    $c_1^t = \hat{c}_1^t(\bar{c})$, where $\bar{c} = \text{argmax}_c\, v_{\text{next}}(c)$ at $t = T$

Figure 6.6. *Pseudocode for the Viterbi algorithm for class sequences.*

Two pass classification: First, classify each entire record. Then classify each minute in the record using a model selected on the basis of the first classification.

Viterbi decoding of classification: For a given model and sequence of observations, use the algorithm of Section 6.3 to solve for a sequence of classifications.

6.4.1 Extracting a Low Pass Filtered Heart Rate

From the raw ECG, I derived two kinds of sequences of observations. At each sample time the code extracts a scalar *low pass heart rate signal* that conveys information about the low frequency heart rate oscillations and a *respiration vector* that conveys information about high frequency phase oscillations. I used the following signal processing steps to derive the low pass heart rate signal:

1. Clip the raw ECG signal. The ECG signals have a few time intervals during which the values have extremely large magnitudes. I suspect that the electrical leads were physically disturbed at those times. I eliminated those useless values as follows:

   (a) For each second, find the lowest, l, and highest, h, value of the ECG signal.
   (b) Sort these two collections of extreme values.
   (c) Find X_l, the largest value that is smaller than 90% of the l values, and define the lower bound[38] as $B_l = 1.5 * X_l$.
   (d) Find X_h, the smallest value that is larger than 90% of the h values, and define the upper bound as $B_h = 1.5 * X_h$.
   (e) For every second that has an ECG value outside of the range $[B_l, B_h]$ set all ECG values in that minute to 0.
2. Find the time of each heart beat by applying the PhysioNet program *wqrs* to the clipped ECG signal. I call these times the *Rtimes*.
3. Create a sequence of raw pairs [*time, rr-interval*], where *time* is an *Rtime* and *rr-interval* is the time delay to the next *Rtime*.
4. Cull the sequence of pairs by removing entries that have extreme *rr-interval* values. For various reasons, the code misses some heart beats. Without culling, the missed beats introduce impulsive noise into the heart rate signal that is at least twice as large as the real signal.
5. Create a heart rate signal, hr, sampled uniformly at 2 Hz by interpolating between times in the sequence of remaining pairs.
6. Subtract the mean heart rate from hr and then use the FFT twice to eliminate frequency components below 0.09 cpm and above 3.66 cpm. Call the resulting signal hr_l.
7. Save hr_l sampled 10 times per minute.

6.4.2 Extracting Respiration Information

I also created a *respiration* signal with a sample every tenth of a minute where each sample is a vector with three components. As a first step, I used the following procedure to derive periodograms[39] from the same *Rtimes* sequence I used for the *low pass heart rate* signal:

1. Read the *Rtimes* data.
2. Calculate a *deviation* time series sampled uniformly at 2 Hz. The deviation is the difference between the actual value of an *Rtime* and the value that is midway between its nearest neighbors.
3. Calculate 10 periodograms per minute using FFTs and Gaussian windows with a support of 512 seconds and $\sigma = 25$ seconds.
4. Smooth each periodogram over frequencies.

[38] Since X_l is negative, the lower bound is even more negative, i.e., $B_l < X_l$.

[39] Periodograms are calculated by applying the FFT to a sample sequence and squaring the resulting coefficients. The result is a Fourier transform of the autocorrelation of a finite segment of a single sequence. Periodograms are often used, e.g., by averaging, to estimate the Fourier *power spectral density* (PSD) as a function of frequency.

5. Normalize each smoothed periodogram to be a vector F_j with unit length.

6. Save the vector F_j and the normalization factor Z_j.

I used *Fisher linear discriminant analysis* to reduce the information in each periodogram to two scalar values. To implement the analysis, I collected the following three classes of periodogram vectors: C, those drawn from c records; N, those drawn from a records during minutes that are labeled *normal*; and A, those drawn from a records during minutes that are labeled *apnea*.

A basis of Fisher linear discriminants $\mathbf{W}$ maximizes the separation of classes compared to the spread within classes as expressed by the objective function

$$J(\mathbf{W}) \equiv \frac{\left|\mathbf{W}^\top \mathbf{S}_B \mathbf{W}\right|}{\left|\mathbf{W}^\top \mathbf{S}_W \mathbf{W}\right|},$$

where $\mathbf{S}_B$ is the *between* class scatter and $\mathbf{S}_W$ is the *within* class scatter. My classes were C, N, and A. Using the notation $E(j) = i$ to mean "The minute corresponding to periodogram F_j was in class i.", I can write my calculation of $\mathbf{S}_B$ and $\mathbf{S}_W$ as follows:

$$\begin{aligned}
m_i &= \frac{1}{n_i} \sum_{j:E(j)=i} F_j, \\
\mathbf{S}_i &= \sum_{j:E(j)=i} (F_j - m_i)^\top (F_j - m_i), \\
\mathbf{S}_W &= \sum_i \mathbf{S}_i, \\
n &= \sum_i n_i, \\
m &= \frac{1}{n} \sum_i n_i m_i, \\
\mathbf{S}_B &= \sum_i n_i (m_i - m) \times (m_i - m)^\top.
\end{aligned}$$

While the classic solution (see page 123 of Duda, Hart, and Stork [3]) is to choose a basis of eigenvectors corresponding to the largest eigenvalues for the generalized eigenvalue problem

$$\mathbf{S}_B w_i = \lambda_i \mathbf{S}_W w_i,$$

I solved the simple eigenvalue problem

$$Q \Lambda Q^\top = (\mathbf{S}_W + 100\mathbf{I})^{-1} \mathbf{S}_B \tag{6.12}$$

and chose as $\mathbf{W}$ the two eigenvectors (two columns of Q) that correspond to the two largest eigenvalues (elements of Λ). In (6.12), $100\mathbf{I}$ is a regularization term that roughly quenches the effect of small or zero eigenvalues of $\mathbf{S}_W$. Every tenth of a minute, I recorded the three numbers $(\mathbf{W}F_j, Z_j)$ as a *respiration* vector. See results of the analysis in Fig. 6.7.

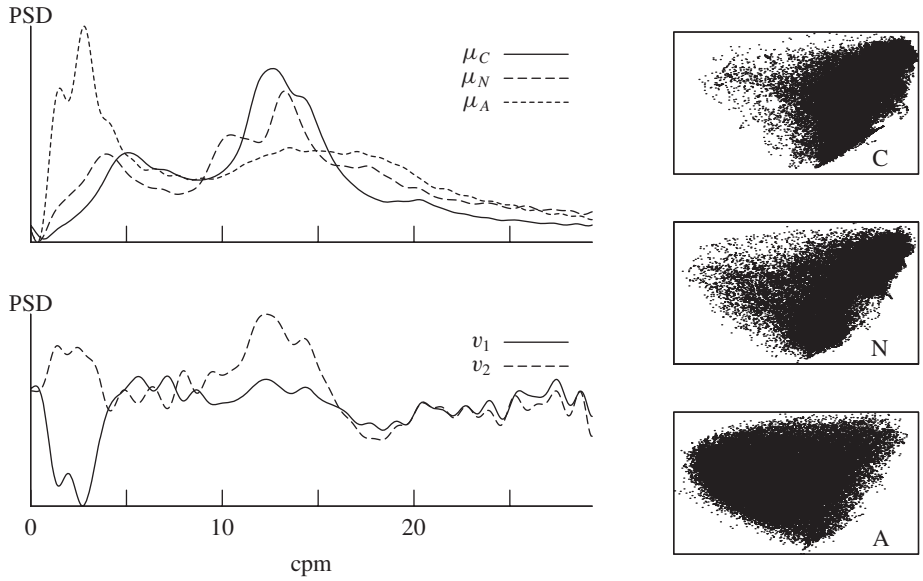


Figure 6.7. *Linear discriminant analysis of phase jitter periodograms. The plot in the upper left shows the following mean periodograms: μ_C, the mean for the c records; μ_N, the mean of the minutes that the expert classified as normal in the a records; and μ_A, the mean of the minutes that the expert classified as apnea in the a records. The lower left plot shows the basis vectors that result from the linear discriminant analysis. Scatter plots of the three classes projected on the basis (v_1, v_2) appear on the right.*

6.4.3 Classifying Records

My classification algorithm analyzes each record in two passes. On the first pass I assign an entire record to one of the following groups: *High* (H), *Medium* (M), or *Low* (L). The goal is to have the a records assigned to group H and the c records assigned to group L. The M group is a buffer to avoid the error of assigning either an a record to L or a c record to H.

Initially, I hoped to use a likelihood ratio test for the first pass. After some testing of different model classes, I selected the following two models:

mod_A2: A two state HMM with AR-4 models for the low pass heart rate signal and simple Gaussian models for the respiration vectors.

mod_C1: A simple model with a single AR-4 model for the low pass heart rate signal and a single Gaussian model for the respiration vectors.

I trained mod_A2 on all of the a records and trained mod_C1 on all of the c records. When I applied the resulting classifier to the training data, there were a few errors. To improve the performance, I considered other signal characteristics. Since the errors were on records for which the low pass heart rate oscillations were small, I looked at characterizations of

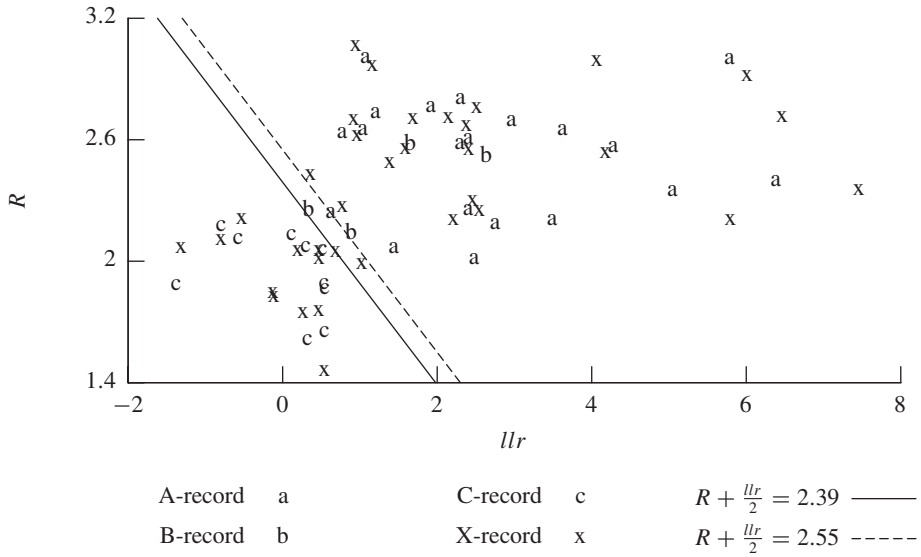


Figure 6.8. *The first pass classifier. I have plotted the location of each record using the log-likelihood ratio llr and the ratio statistic R. Records to the left of the line $2.39 - \frac{llr}{2}$ are in the L group. Records to the right of the line $2.55 - \frac{llr}{2}$ are in the H group. And those in between are in the M group.*

the amplitude of that signal. I came up with the following statistic:

$$R = \frac{S_{74}}{L_1},$$

where S_{74} is the *size* of the smallest waveform peak that is larger than 74% of the peaks in the record and L_1 is the average of the absolute values of all the peaks in the record.

Using both the likelihood ratio and R, I was still unable to correctly classify all of the a and c training records with a straight line. I could let record $a06$ fall in the L group or let record $c08$ fall in the H group. So I retrained the models using augmented training data. In particular, I simply used the records that were near the decision boundary more than once. A procedure called *boosting* [33, 19] augments training sets based on classification errors in a principled way. My augmentation was ad hoc by comparison. The result (see Fig. 6.8) is a pair of statistics with which I could classify all of the a records and c records correctly using a line.

Notice that in Fig. 6.8 I have plotted *all* of the records, the training data, and the test data. Perhaps an honest man would not have looked at the test data until he had built a classifier based on the training data alone. In preparing the models and code for this chapter I did look at the training data, but I did not look at the expert classifications. Although I was trying to recreate the restrictions of the contest, those restrictions seem a little weak in view of the fact that James demonstrated that he could classify the test data using the ECG alone. I looked at each of the 11 x records in the L group, and none of them seemed to have evidence of apnea.

6.4.4 Model Topology and Training Data

After choosing the statistic $R + \frac{llr}{2}$ for classifying records in the first pass, I had to select model structures and training strategies for the classifiers that would implement the second pass. In a somewhat haphazard fashion,[40] I selected HMMs with the topologies drawn in Fig. 6.9. In the figure, each state label has the form $C_{Ti[j]}$ with the following interpretation:

C The *class* of the state: Normal N or Apnea A

T The *type* of state: Home H, isle I, or peak P. I assigned special observation classes to the second state in each peak group (P_{*2} in Fig. 6.9) and to the times that typical peaks occur in the data. Those assignments forced the training algorithm to fit the observation models of the P states to the characteristics of typical peaks in the data.

i An index to separate multiple instances of the same type

j An index used only for P states that identifies the subtype

Starting from random parameters, I trained each of the three models on subsets of the available training data. For the *low* model, Mod_L, I used records with classification statistics below 2.39, that is, all of the c records and record b04. For the *medium* model, Mod_M, I used records with classification statistics between 2.23 and 3.0, that is, records c07, c08, b04, a06, b01, and a11. And for the *high* model, Mod_H, I used records with classification statistics above 2.55, that is, all of the a records and records b01, b02, and b03. I chose the training sets after looking at Fig. 6.8. I wanted records in the training set of each of the three models that would be like the test records to which the model would be applied.

6.4.5 Tunable Parameters

After training the three models, I combined them with the first pass statistic and classified each minute of the training data with the following results:

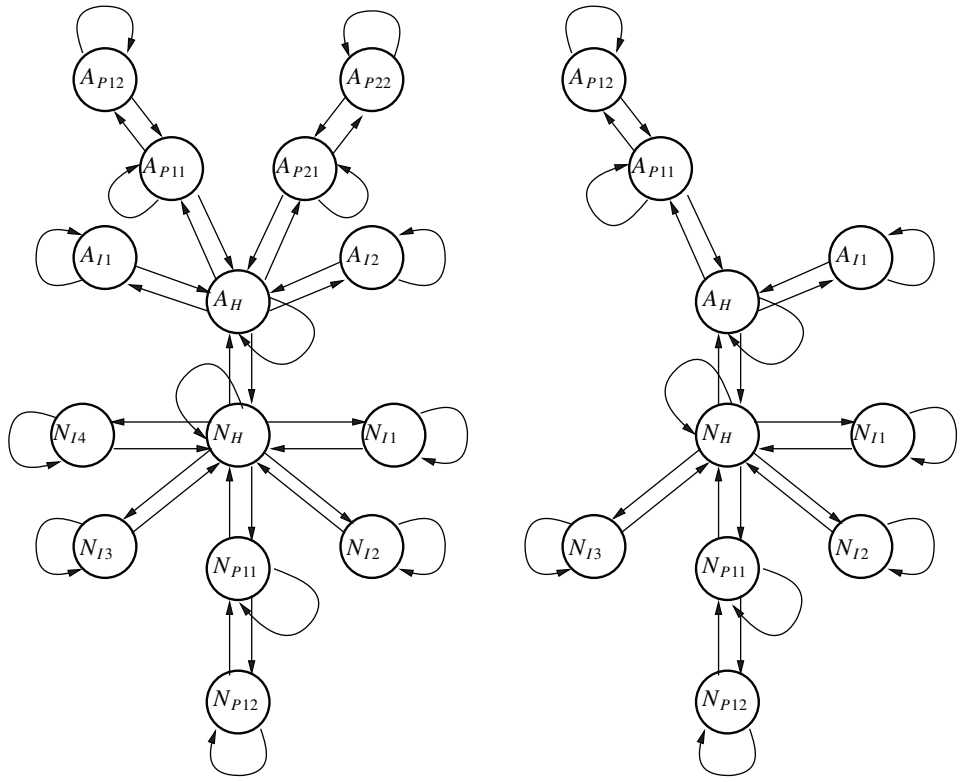
Percent of training minutes correctly classified: 78.87%.

Number of normal minutes misclassified: Of the 10,136 minutes labeled normal by the expert, my code classified 3,162 or 31.2% as apnea.

Number of apnea minutes misclassified: Of the 6,452 minutes labeled apnea by the expert, my code classified 343 or 5.3% as normal.

Looking at the numbers and wishing for a score above 90%, I came up with two more parameters to adjust. The first was a factor that acts like a threshold adjustment. The idea was to favor normal classifications just a little bit more in the hope that I could reduce the number of misclassified normal minutes at the cost of a smaller increase in misclassified apnea minutes. I implemented the adjustment by multiplying all observation probabilities in *normal* states by a factor slightly more than one. The adjustment has the effect of increasing

[40] I selected the topology to optimize the performance of flawed software. Following a disk failure, I rebuilt the software and fixed the known flaws, but I did not reexamine the model topology.



Mod_H Mod_M and Mod_L

Figure 6.9. *Structure of HMMs for minute by minute classification in the second pass of my procedure. I used the structure on the left for those records classified as H on the first pass and the structure on the right for those records classified as M or L on the first pass.*

the number of minutes classified as normal. If the factor is less than one, it has the opposite effect. I called the factor *Fudge*.

The second factor I decided to adjust modifies the relative weighting of the two components of the observations, i.e., the low pass heart rate y_{hr} and the respiration vector y_{resp}. In Section 6.4.4, for the conditional probability of both components given the state, I used the product of the conditional probabilities,

$$P(y|s) = P(y_{\text{hr}}|s)P(y_{\text{resp}}|s),$$

where $y \equiv \left(y_{\text{hr}}, y_{\text{resp}}\right)$ denotes the composite observation. To give more weight to the low pass heart rate component of the model, I could raise that probability to a power higher than one, i.e.,

$$P(y|s) = \left(P(y_{\text{hr}}|s)\right)^{\text{Pow}} P(y_{\text{resp}}|s).$$

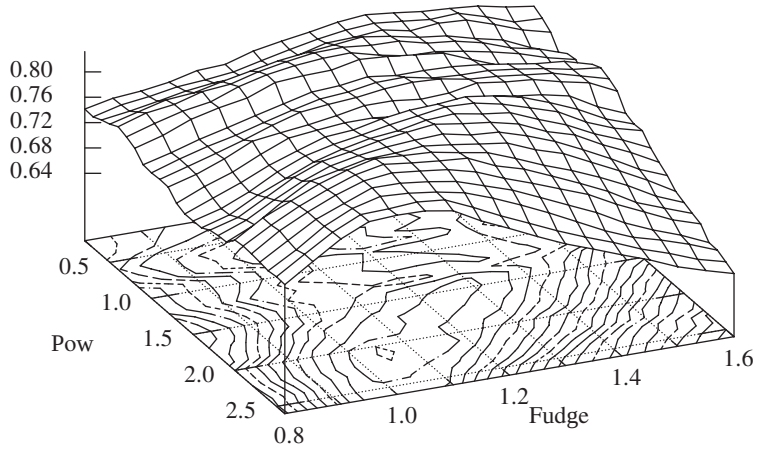


Figure 6.10. *The response of classification performance to changes in* Pow *and* Fudge*. I have plotted the performance of Mod_H trained and evaluated on the H group of records. As described in the text,* Pow *governs the relative weighting of the low pass heart rate signal to the respiration characteristics and* Fudge *is a bias for choosing the* normal *classification. The Z-axis is the fraction of minutes classified correctly.*

I have plotted a survey of the dependence of classification performance on the adjustable parameters *Pow* and *Fudge* in Fig. 6.10. On the basis of such surveys, I selected the following parameters:

	Mod_L	Mod_M	Mod_H
Fudge	2.2	1.15	1.05
Pow	0.9	2.5	2.1

6.4.6 Results

With tuned parameters, the model classified 88.24% of the minutes in the training data correctly (See Table 6.1 for more detail.). Applying the same model to the test data yielded a score of 86.37% with 11,292 minutes classified as normal and 5,976 minutes classified as apnea. Thus my code classified 34.61% of the minutes in the test data as apnea, while the expert classified 38.96% of the minutes in the training data as apnea.

In my final attempt to improve the score, I adjusted the Fudge factor used by Mod_H so that the total fraction of minutes in the test data classified as apnea matched the fraction of minutes in the training set classified as apnea by the expert. Changing the Fudge factor to 0.988 yielded a score of 86.95% with 10,541 minutes classified as normal and 6,727 or 38.96% minutes classified as apnea. While I was disappointed not to get a score with HMMs that would have won in 2000, a look at Table 6.2 verifies that the HMM performance is among those *top scores*.

Table 6.1. *Performance with tuned values of* Fudge *and* Pow *on training records. I have sorted the list in order of how well the code classified the minutes in each record. For each record, the number in the column labeled $N \rightarrow A$ is the number of minutes labeled as* normal *by the expert that the code labeled as* apnea*. The interpretations of the other columns are similar.*

Record	$N \rightarrow N$	$N \rightarrow A$	$A \rightarrow N$	$A \rightarrow A$	% Right
a11	198	46	138	84	0.6052
b02	255	169	14	79	0.6460
a06	276	27	140	66	0.6719
a08	197	114	24	165	0.7240
b01	362	105	2	17	0.7798
a07	96	93	12	309	0.7941
a18	37	14	82	356	0.8037
b03	296	71	13	60	0.8091
a03	175	98	0	246	0.8112
a20	184	10	78	237	0.8271
a15	91	50	36	332	0.8310
a05	147	30	44	232	0.8366
a16	140	21	51	269	0.8503
a13	213	38	28	215	0.8664
a09	90	24	39	342	0.8727
a10	404	13	49	50	0.8798
a14	69	57	2	381	0.8841
a17	302	24	32	126	0.8843
a02	72	36	19	401	0.8958
a19	289	8	30	174	0.9242
a12	14	29	3	530	0.9444
b04	418	0	10	0	0.9766
a01	11	8	0	470	0.9836
a04	35	4	2	451	0.9878
c07	424	0	4	0	0.9907
c05	462	0	3	0	0.9935
c09	465	0	2	0	0.9957
c10	429	0	1	0	0.9977
c03	452	1	0	0	0.9978
c06	466	0	1	0	0.9979
c02	500	0	1	0	0.9980
c01	483	0	0	0	1.0000
c04	481	0	0	0	1.0000
c08	513	0	0	0	1.0000
Sum	9046	1090	860	5592	0.8824

Table 6.2. *Here are the scores described in this chapter interspersed with the top scores from the CINC*2000 *Web site (*`http://www.physionet.org/challenge/2000/top-scores.shtml`*).*

Score	Entrant	Entries
92.62	J. McNames, A. Fraser, and A. Rechtsteiner: Portland State University, Portland, OR, USA	4
92.30	B. Raymond, R. Cayton, R. Bates, and M. Chappell: Birmingham Heartlands Hospital, Birmingham, UK	8
89.36	P. de Chazal, C. Henehan, E. Sheridan, R. Reilly, P. Nolan, and M. O'Malley: University College - Dublin, Ireland	15
87.56	M. Schrader, C. Zywietz, V. von Einem, B. Widiger, G. Joseph: Medical School Hannover, Hannover, Germany	9
87.30	M.R. Jarvis and P.P. Mitra: Caltech, Pasadena, CA, USA	3
86.95	Second entry in this chapter. Adjust *Fudge* to get fraction of apnea minutes in the test records to match the fraction of apnea minutes in the training records.	
86.24	First entry in this chapter. Models and parameters tuned to the training records.	
85.63	Z. Shinar, A. Baharav, and S. Akselrod: Tel-Aviv University, Ramat-Aviv, Israel	1
85.54	C. Maier, M. Bauch, and H. Dickhaus: University of Heidelberg, Heilbronn, Germany	5
84.49	J.E. Mietus, C.-K. Peng, and A.L. Goldberger: Beth Israel Deaconess Medical Center, Boston, MA, USA (unofficial entry)	

6.5 Classification Versus Estimation

In the early chapters of this book, I have emphasized choosing model parameters to maximize the likelihood of the data, while the tweaks and fudges I have used in this chapter are concerned with improving classification performance rather than improving likelihood. While it is true that if one had access to accurate probabilistic characterizations of observations conditional on class membership, then the best classifier would be a likelihood ratio classifier, it is not true that without such characterizations the best approach to classification is to estimate them. This point is made forcefully by Vapnik [16], who says, "one should solve the [classification] problem directly and never solve a more general problem as an intermediate step."

Appendix A

Formulas for Matrices and Gaussians

Here I review some material necessary for deriving (4.31)–(4.35). Similar material appears in Appendix A of Kailath, Sayed, and Hassibi [7].

Block Matrix Inverse

If G is an $n \times n$ invertible matrix, K is an $m \times m$ invertible matrix, and H and J are $n \times m$ and $m \times n$, respectively, then direct matrix multiplication verifies that

$$\begin{bmatrix} (G - HK^{-1}J)^{-1} & -(G - HK^{-1}J)^{-1}HK^{-1} \\ -(K - JG^{-1}H)^{-1}JG^{-1} & (K - JG^{-1}H)^{-1} \end{bmatrix} \begin{bmatrix} G & H \\ J & K \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I} \end{bmatrix}, \quad \text{(A.1a)}$$

$$\begin{bmatrix} G & H \\ J & K \end{bmatrix} \begin{bmatrix} (G - HK^{-1}J)^{-1} & -G^{-1}H(K - JG^{-1}H)^{-1} \\ -K^{-1}J(G - HK^{-1}J)^{-1} & (K - JG^{-1}H)^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I} \end{bmatrix}, \quad \text{(A.1b)}$$

assuming that $(G - HK^{-1}J)^{-1}$ and $(K - JG^{-1}H)^{-1}$ exist. One can derive other expressions for the inverse by using the Sherman–Morrison–Woodbury formula (see (A.3)) to expand terms in (A.1).

By noting that

$$\begin{bmatrix} (G - HK^{-1}J)^{-1} & -G^{-1}H(K - JG^{-1}H)^{-1} \\ 0 & (K - JG^{-1}H)^{-1} \end{bmatrix} \begin{bmatrix} G & H \\ J & K \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ (K - JG^{-1}H)^{-1}J & (K - JG^{-1}H)^{-1}K \end{bmatrix}$$

and taking the determinant of both sides

$$\left|(K - JG^{-1}H)^{-1}\right| |K| = \left|(G - HK^{-1}J)^{-1}\right| \left|(K - JG^{-1}H)^{-1}\right| \left|\begin{bmatrix} G & H \\ J & K \end{bmatrix}\right|$$

one finds the following formula for determinants:

$$\left|\begin{bmatrix} G & H \\ J & K \end{bmatrix}\right| = |K| \left|(G - HK^{-1}J)\right|. \quad \text{(A.2)}$$

Sherman–Morrison–Woodbury Formula

If G and J are invertible matrices, H and K have dimensions so that $(G + HJK)$ makes sense, and $\left(KG^{-1}H + J^{-1}\right)^{-1}$ exists, then

$$(G + HJK)^{-1} = G^{-1} - G^{-1}H\left(KG^{-1}H + J^{-1}\right)^{-1}KG^{-1}. \tag{A.3}$$

Multiplying both sides by $(G + HJK)$ verifies the formula. The special case

$$\left(L^{-1} + H^{\top}J^{-1}H\right)^{-1} = L - LH^{\top}\left(HLH^{\top} + J\right)^{-1}HL \tag{A.4}$$

is efficient when the dimension, m, of J is smaller than the dimension n of L because an $n \times n$ matrix inversion on the left is replaced by an $m \times m$ matrix inversion on the right.

Marginal and Conditional Distributions of a Gaussian

Suppose that $W = \begin{bmatrix} U \\ V \end{bmatrix}$ is a Gaussian random variable with an n-dimensional component U and an m-dimensional component V. I write its distribution

$$W \sim \mathcal{N}(\mu_W, \Sigma_W) \quad \text{or, equivalently,} \quad P(w) = \mathcal{N}(\mu_W, \Sigma_W)|_w$$

with

$$\mu_W = \begin{bmatrix} \mu_U \\ \mu_V \end{bmatrix} \quad \text{and} \quad \Sigma_W = \begin{bmatrix} \Sigma_{UU} & \Sigma_{UV} \\ \Sigma_{VU} & \Sigma_{VV} \end{bmatrix} \equiv \begin{bmatrix} A & C \\ C^{\top} & B \end{bmatrix},$$

where I have introduced $A \equiv \Sigma_{UU}$, $B \equiv \Sigma_{VV}$, and $C \equiv \Sigma_{UV}$ to shorten the notation. If I denote

$$\Sigma_W^{-1} = \begin{bmatrix} D & F \\ F^{\top} & E \end{bmatrix},$$

then from (A.1) and (A.3)

$$D = (A - CB^{-1}C^{\top})^{-1} \qquad = A^{-1} + A^{-1}CEC^{\top}A^{-1}, \tag{A.5a}$$

$$E = (B - C^{\top}A^{-1}C)^{-1} \qquad = B^{-1} + B^{-1}C^{\top}DCB^{-1}, \tag{A.5b}$$

$$F = -A^{-1}CE \qquad = -DCB^{-1}. \tag{A.5c}$$

In this notation, the marginal distributions are

$$P(u) = \int P(u, v)\, dv \tag{A.6a}$$

$$= \mathcal{N}(\mu_U, A)|_u, \tag{A.6b}$$

$$P(v) = \int P(u, v)\, du \tag{A.6c}$$

$$= \mathcal{N}(\mu_V, B)|_v, \tag{A.6d}$$

and the conditional distributions are

$$P(u|v) = \frac{P(u, v)}{P(v)} \tag{A.7a}$$

$$= \mathcal{N}\left(\mu_U + CB^{-1}(v - \mu_v), D^{-1}\right)\Big|_u, \tag{A.7b}$$

$$P(v|u) = \frac{P(v, u)}{P(u)} \tag{A.7c}$$

$$= \mathcal{N}\left(\mu_V + C^\top A^{-1}(u - \mu_U), E^{-1}\right)\Big|_v. \tag{A.7d}$$

Notice that the covariance of the *marginal* distribution of U is given by the UU block of Σ_W but that the inverse covariance of the *conditional* distribution of U is given by the UU block of Σ_W^{-1}.

As a check of these formulas, I examine $P(u|v)\,P(v)$ and find that

$$\begin{aligned}
P(u|v)\,P(v) &= \frac{\sqrt{|D|}}{\sqrt{(2\pi)^n}} e^{-\frac{1}{2}\left(u-\mu_U-CB^{-1}(v-\mu_V)\right)^\top D\left(u-\mu_U-CB^{-1}(v-\mu_V)\right)} \\
&\quad \times \frac{1}{\sqrt{(2\pi)^m\,|B|}} e^{-\frac{1}{2}(v-\mu_V)^\top B^{-1}(v-\mu_V)} \\
&= \frac{1}{\sqrt{(2\pi)^{n+m}\,|\Sigma_W|}} \exp\left(-\frac{1}{2}\Big[\right. \\
&\quad \left(u - \mu_U - CB^{-1}(v-\mu_V)\right)^\top D\left(u - \mu_U - CB^{-1}(v-\mu_V)\right) \\
&\quad \left. + (v-\mu_V)^\top B^{-1}(v-\mu_V)\Big]\right) \\
&= \frac{1}{\sqrt{(2\pi)^{n+m}\,|\Sigma_W|}} \\
&\quad \times e^{-\frac{1}{2}\left((u-\mu_U)^\top D(u-\mu_U)+2(v-\mu_V)^\top F^\top(u-\mu_U)+(v-\mu_V)^\top E(v-\mu_V)\right)} \\
&= P(u, v);
\end{aligned}$$

all is right with the world. In the above, (A.2) implies that $\frac{\sqrt{|D|}}{\sqrt{|B|}} = \frac{1}{\sqrt{|\Sigma_W|}}$.

Completing the Square

Some of the derivations in Section 4.3 rely on a procedure called *completing the square*, which I illustrate with the following example. Suppose that the function $f(u)$ is the product of two n-dimensional Gaussians, $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$, i.e.,

$$f(u) = \frac{1}{\sqrt{(2\pi)^n\,|\Sigma_1|}} e^{-\frac{1}{2}(u-\mu_1)^\top\Sigma_1^{-1}(u-\mu_1)} \frac{1}{\sqrt{(2\pi)^n\,|\Sigma_2|}} e^{-\frac{1}{2}(u-\mu_2)^\top\Sigma_2^{-1}(u-\mu_2)} \tag{A.8}$$

$$= \frac{1}{\sqrt{(2\pi)^{2n}\,|\Sigma_1|\,|\Sigma_2|}} e^{-\frac{1}{2}\left[(u-\mu_1)^\top\Sigma_1^{-1}(u-\mu_1)+(u-\mu_2)^\top\Sigma_2^{-1}(u-\mu_2)\right]} \tag{A.9}$$

$$\equiv \frac{1}{\sqrt{(2\pi)^{2n}\,|\Sigma_1|\,|\Sigma_2|}} e^{-\frac{1}{2}\left[Q(u)\right]}. \tag{A.10}$$

By expanding the function $Q(u)$ in the exponent, I find that

$$Q(u) = u^\top \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right) u - 2u^\top \left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right) + \mu_1^\top \Sigma_1^{-1}\mu_1 + \mu_2^\top \Sigma_2^{-1}\mu_2 \quad \text{(A.11)}$$

$$= u^\top q u - 2u^\top l + s, \quad \text{(A.12)}$$

where the quadratic, linear, and scalar terms are

$$q = \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right),$$
$$l = \left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right),$$
$$s = \mu_1^\top \Sigma_1^{-1}\mu_1 + \mu_2^\top \Sigma_2^{-1}\mu_2,$$

respectively.

Completing the square means finding values μ, Σ, and R for which (A.12) takes the form

$$Q(u) = (u-\mu)^\top \Sigma^{-1}(u-\mu) + R, \quad \text{(A.13)}$$

where R is not a function of u. One can verify by substitution that the solution is

$$\Sigma^{-1} = q,$$
$$\mu = \Sigma l,$$
$$R = s - \mu^\top \Sigma^{-1}\mu.$$

For the product of Gaussians example (A.8),

$$\begin{aligned}
\Sigma^{-1} &= \Sigma_1^{-1} + \Sigma_2^{-1}, \\
\mu &= \Sigma\left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right) \\
&= \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)^{-1}\left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right), \\
R &= \mu_1^\top \Sigma_1^{-1}\mu_1 + \mu_2^\top \Sigma_2^{-1}\mu_2 - \mu^\top \Sigma^{-1}\mu \\
&= \mu_1^\top \Sigma_1^{-1}\mu_1 + \mu_2^\top \Sigma_2^{-1}\mu_2 - \left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right)^\top \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)^{-1}\left(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2\right).
\end{aligned}$$

In words, the product of two Gaussian density functions is an unnormalized Gaussian density function in which the inverse covariance is the sum of the inverse covariances of the factors, and the mean is the average of the factor means weighted by the inverse covariances.

Appendix B
Notes on Software

A software package that *makes* this book is available at `www.siam.org/books/ot107`. The package consists of LaTeX source files, scripts, code, and data required for making the figures and ultimately the book. On a GNU/Linux system, after unpacking the files and installing the prerequisites, typing "make" in the top level directory will create a copy of the book after some hours of computation.

I reorganized the software before the book was distributed to make the HMM tools a separate package that has few prerequisites. The remainder of the software, the *book package*, requires most of the dependencies listed in the next section to run.

Dependencies

The original code relied on many free software packages. Although I developed the code on Debian GNU/Linux systems, it will run on any of the many other platforms for which the dependencies are available. In addition to the standard GNU development tools, the book package requires the following and their dependencies:

gnuplot Old reliable plotting package driven with either a command line interface or a script

libgsl0-dev GNU Scientific Library

python An interpreted, interactive, object-oriented, extensible programming language

python-dev Header files necessary for compiling code wrapped by SWIG

python-matplotlib A GUI driven plotting package that I use to plot a spectrogram

python-scipy Scientific methods for Python. SciPy was developed as I wrote the book. Hoping for both stability and access to a rich set of tools, I switched between using the SciPy, numeric, and numarray packages too often. Now SciPy is reasonably stable and complete.

python-tk Python interface to the Tk GUI toolkit

rubber A tool to automate LaTeX compilations. Rubber figures out how many LaTeX passes are needed and also runs metapost/bibtex/dvips/, etc., as necessary. Although I recommend rubber, the code will run without it.

SWIG The *Simplified Wrapper and Interface Generator* wraps code written in C or C++ for any of a number of higher level programming languages. I used it to make fast C implementations of the basic HMM algorithms callable from Python.

tetex or texlive TeX, LaTeX, and much supporting software

transfig For translating xfig drawings to eps

Data

I used the following sets of data for examples. The book package includes extracts that are sufficient to build the book.

Tang's laser data. Carl Otto Weiss mailed me a CD full of data from various experiments that he and Tang did in the 1980s and 1990s. Although I analyzed many of the files, I finally used only a file called *LP5.DAT* in the book (see Section 1.1).

H. L. Mencken's *A Book of Prefaces*. I used Mencken's book for the parts of speech example in Section 1.3.2. Although you can get Mencken's book from *Project Gutenberg* [57], my book package includes a parsed version of text.

CINC2000 ECG data. I used Dr. Thomas Penzel's ECG measurements throughout Chapter 6. You can get the raw data from

`www.physionet.org/physiobank/database/apnea-ecg.`

In the book package, I include the much smaller files that contain estimates of the timing of heart beats.

Clarity and Efficiency

Perhaps the clearest way to describe and code the algorithms of Chapter 2 is in terms of vector and matrix operations. For example, see Kevin Murphy's *Hidden Markov Model (HMM) Toolbox* at

`www.cs.ubc.ca/ murphyk/Bayes/hmm.html`.

In preliminary work on the code for this book, Ralf Juengling used the Python *SciPy* package to similar effect. Here is his implementation of the entire forward algorithm:

```
def forward(hmm, y):
    P_S0, P_ScS, P_YcS = hmm.parameters()
    N = len(hmm.states)
    T = len(y)
```

```
    # create data structures alpha and gamma
    alpha = N.zeros((T, N), N.float64)
    # alpha[t,s] = P(s|y(0..t))
    gamma = N.zeros((T, 1), N.float64)
    # gamma[t] = P(y(t)|y(0..t-1))

    # fill up alpha and gamma
    alpha[0] = P_YcS[y[0]]*P_S0
    gamma[0] = N.sum(alpha[0])
    alpha[0] /= gamma[0]
    for t in xrange(1, T):
        P_ScY_1_prev = N.dot(P_ScS, alpha[t-1])     # (2.5)
        P_SYcY_1_prev = P_YcS[y[t]]*P_ScY_1_prev    # (2.6)
        gamma[t] = N.sum(P_SYcY_1_prev)             # (2.7)
        alpha[t] = P_SYcY_1_prev/gamma[t]           # (2.4)

    return alpha, gamma
```

The comments refer to the formulas in Chapter 2 implemented by each line.

As I developed code for the examples in the book, I abandoned the simplicity of Juengling's code in the pursuit of flexibility and speed. To have code that could use special observation models such as those in Chapters 3 and 6, I replaced *P_YcS[yt]* with a function call. To make code faster, I wrote routines in C and called them from Python using SWIG. And to have code that could operate on the large models required for Fig. 5.6, I used hash tables (Python *dictionaries*) rather than lists or arrays. The result is a collection of code that is flexible and fast enough to build the book in a few hours but not as easy to read as I would have liked.

After many modifications of observation models as I worked on Chapter 6, I found that caching the conditional probabilities of observations given the states works well. Originally I used a class method to calculate $P(y(t)|s)$ for particular values of s and t wherever the code required such a probability. To use different observation models, I wrote subclasses that redefined the method. The technique worked, but there were drawbacks. By profiling the Python code, I found that most of the time was spent calling the methods that implemented $P(y(t)|s)$. For the simplest observation models (discrete outputs which can be implemented as arrays), Karl Hegbloom examined the details and reported:

> Actually, it was not the actual method calls that took so much time. The problem was that Python had to traverse its object-representation data structures in order to find the methods. The result was returned by code about three levels deep. The interpreter dutifully performed that look-up chaining over and over again, once for each run-time method call.
>
> Python is a dynamic language where variables are not constrained to hold values of a particular type. When you declare a variable, you just name it, without saying anything to Python about the data type of the object that will be kept at the location it stands for. This language characteristic, known as "duck typing," makes it easy to implement *interface polymorphism*. "If it walks like a duck

and quacks like a duck, it must be a duck." To make that work, method look-up must happen at run-time. Since there is no syntax for declaring the variable to hold only a certain data-type, the Python byte-compiler has no way of resolving those at compile-time.

Reorganizing the code so that before any call to the forward algorithm, i.e., each iteration of the Baum–Welch algorithm, and before any call to the Viterbi algorithm a single call to an observation method calculates and stores $P(y(t)|s)$ for every value of s and t resolved the drawbacks at the expense of using more memory.[41]

[41] For big problems the modification requires about 30% more memory to run the Baum–Welch algorithm.

Bibliography

Books and Collections

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag, New York, 2006.

[2] T. Cover and J. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[3] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, Wiley-Interscience, New York, 2nd ed., 2001.

[4] J. Ferguson, ed., *Symposium on the Application of Hidden Markov Models to Text and Speech*, Institute for Defense Analyses, Communications Research Division, Princeton, NJ, October 1980.

[5] J. Gibbs, *Elementary Principles in Statistical Mechanics Developed with Especial Reference to the Rational Foundation of Thermodynamics*, Yale University Press, New Haven, CT, 1902. Reprinted, Dover, New York, 1960.

[6] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 3rd ed., 1996.

[7] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*, Prentice–Hall, Upper Saddle River, NJ, 2000.

[8] A. Katok and B. Hasselblatt, *Introduction to the Modern Theory of Dynamical Systems*, Cambridge University Press, New York, 1995.

[9] R. Mane, *Ergodic Theory and Differentiable Dynamics*, Springer-Verlag, Berlin, 1987. Translated from the Portuguese by Silvio Levy.

[10] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, Academic Press, New York, London, 1979.

[11] ——, *Stochastic Models, Estimation, and Control,* Vol. 2, Academic Press, London, 1982.

[12] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, Wiley, New York, 1996.

[13] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA, 2nd ed., 1991.

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Veterling, *Numerical Recipes in C: The art of scientific computing*, Cambridge University Press, Cambridge, UK, 2nd ed., 1992.

[15] J. Schafer, *Analysis of Incomplete Multivariate Data*, Monog. Stat. Appl. Probab. 72, Chapman and Hall, London, 1997.

[16] V. Vapnik, *Statistical Learning Theory*, Wiley-Interscience, New York, 1998.

[17] M. Watanabe and K. Yamaguchi, eds., *The EM Algorithm and Related Statistical Models*, Marcel Dekker, New York, 2004.

Review Articles

[18] J. Ferguson, *Hidden Markov analysis: An introduction*, in Proceedings of the Symposium on the Applications of Hidden Markov Models to Text and Speech, Princeton, NJ, 1980, IDA-CRD, pp. 8–15.

[19] J. Friedman, T. Hastie, and R. Tibshirani, *Additive logistic regression: A statistical view of boosting*, Ann. Statist., 28 (2000), pp. 337–407.

[20] T. Penzel, J. McNames, P. de Chazal, B. Raymond, A. Murray, and G. Moody, *Systematic comparison of different algorithms for apnea detection based on electrocardiogram recordings*, Med. Biol. Eng. Comput., 40 (2002), pp. 402–407.

[21] A. Poritz, *Hidden Markov models: A guided tour*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE, Piscataway, NJ, 1988, pp. 7–13.

[22] L. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proc. IEEE, 77 (1989), pp. 257–286.

[23] T. Sauer, J. Yorke, and M. Casdagli, *Embedology*, J. Statist. Phys., 65 (1991), pp. 579–616.

[24] I. Witten, R. Neal, and J. Cleary, *Arithmetic coding for data compression*, Comm. ACM, 30 (1987), pp. 520–540.

[25] L. Young, *Ergodic theory of differentiable dynamical systems*, in Real and Complex Dynamical Systems, B. Branner and P. Hjorth, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995, pp. 293–336.
`http://citeseer.ist.psu.edu/young95ergodic.html`.

Dissertations and Theses

[26] M. J. Beal, *Variational Algorithms for Approximate Bayesian Inference*, Ph.D. thesis, Gatsby Computational Neuroscience Unit, University College London, London, UK, 2003. Chapter 3, entitled "Variational Bayesian Hidden Markov Models," reviews HMMs in general and describes a Bayesian approach to estimation. The full text of the thesis is available at
`http://www.cse.buffalo.edu/faculty/mbeal/papers.html`.

[27] K. R. Vixie, *Signals and Hidden Information*, Ph.D. thesis, Portland State University, Portland, OR, 2002. Available as Los Alamos Lab report LA-13881-T.

Research Articles

[28] L. Baum and J. Eagon, *An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology*, Bull. Amer. Math. Soc., 73 (1967), pp. 360–363.

[29] L. Baum, T. Petrie, G. Soules, and N. Weiss, *A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains*, Ann. Math. Statist., 41 (1970), pp. 164–171.

[30] G. Benettin, L. Galgani, A. Giorgilli, and J.-M. Sterlcyn, *Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems; a method for computing all of them*, Meccanica, 15 (1980), pp. 9–20.

[31] A. Dempster, N. Laird, and D. Rubin, *Maximum likelihood from incomplete data via the EM algorithm*, J. Roy. Statist. Soc. Ser. B, 39 (1977), pp. 1–38.

[32] A. M. Fraser and H. L. Swinney, *Independent coordinates for strange attractors from mutual information*, Phys. Rev. A, 33 (1986), pp. 1134–1140.

[33] Y. Freund and R. E. Schapire, *Experiments with a new boosting algorithm*, in Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 1996, pp. 148–156.
`http://citeseer.ist.psu.edu/freund96experiments.html`.

[34] J. Gauvain and C. Lee, *Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains.*, IEEE Trans. Speech Audio Process., 2 (1994), pp. 291–298.

[35] N. Gershenfeld, B. Schoner, and E. Metois, *Cluster-weighted modeling for time series analysis*, Nature, 379 (1999), pp. 329–332.

[36] N. Gordon, D. Salmond, and A. Smith, *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*, IEE Proc. F, 140 (1993), pp. 107–113.

[37] H. Haken, *Analogy between higher instabilities in fluids and laser*, Phys. Lett. A, 53 (1975), pp. 77–78.

[38] J. Hughes and P. Guttorp, *A non-homogeneous hidden Markov model for precipitation occurrence*, Appl. Statist, 48 (1999), pp. 15–30.

[39] S. Julier and J. Uhlmann, *A new extension of the Kalman filter to nonlinear systems*, in Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls (Orlando, FL, 1997), Vol. 3068, SPIE, Bellingham, WA, 1997, pp. 182–193.

[40] G. Kitagawa, *Monte Carlo filter and smoother for non-Gaussian nonlinear state space models*, J. Comput. Graph. Statist., 5 (1996), pp. 1–25.

[41] S. Kullback and R. Leibler, *On information and sufficiency*, Ann. Math. Statist., 22 (1951), pp. 79–86.

[42] E. Lorenz, *Deterministic nonperiodic flow*, J. Atmospheric Sci., 20 (1963), pp. 130–141.

[43] D. Ormoneit and V. Tresp, *Improved Gaussian mixture density estimates using Bayesian penalty term and network averaging*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 1996, pp. 542–548.

[44] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. Shaw, *Geometry from a time series*, Phys. Rev. Lett., 45 (1980), pp. 712–716.

[45] Y. B. Pesin, *Characteristic Lyapunov exponents and smooth ergodic theory*, Russian Math. Surveys, 32 (1977), pp. 55–112.

[46] A. Rechtsteiner and A. M. Fraser, *Hidden states for modeling interactions between disparate spatiotemporal scales*, in Proceedings of the International Conference on Complex Systems (ICCS), 2000.

[47] R. A. Redner and H.F. Walker, *Mixture densities, maximum likelihood and the EM algorithm*, SIAM Rev., 26 (1984), pp. 195–239.

[48] R. A. Rosales, M. Fill, and A.L. Escobar, *Calcium regulation of single ryanodine receptor channel gating analyzed using HMM/MCMC statistical methods*, J. Gen. Physiol., 123 (2004), pp. 533–553.

[49] D. Ruelle, *An inequality for the entropy of differentiable maps*, Bol. Soc. Brasil. Mat., 9 (1978), pp. 83–87.

[50] F. Takens, *Detecting strange attractors in turbulence*, in Dynamical Systems and Turbulence (Warwick, 1980), Lecture Notes in Mathematics 898, D. A. Rand and L. Young, eds., Springer, Berlin, 1981, pp. 366–381.

[51] D. Tang, C. Weiss, E. Roldan, and G. de Valcarcel, *Deviation from Lorenz-type dynamics of an NH_3 ring laser*, Opt. Commun., 89 (1992), pp. 47–53.

[52] W. Tucker, *The Lorenz attractor exists*, C. R. Acad. Sci. Paris Sér. I Math., 328 (1999), pp. 1197–1202.
`http://citeseer.ist.psu.edu/tucker99lorenz.html`.

[53] D. Viswanath, *The fractal property of the Lorenz attractor*, Phys. D, 190 (2004), pp. 115–128.

[54] C. J. Wu, *On the convergence properties of the EM algorithm*, Ann. Statist., 11 (1983), pp. 95–103.

Web Sites

[55] Debian, *Debian—The Universal Operating System*. Web site. `http://www.debian.org`.

[56] A. M. Fraser, *Software for Hidden Markov Models*. Web site. `http://www.fraserphysics.com/~andy/hmmdsbook` and `www.siam.org/books/ot107`.

[57] Gutenberg, *Project Gutenberg*. Web site. `http://www.gutenberg.org` (founded by M. Hart).

[58] PhysioNet, *Physiotoolkit—Open Source Software for Biomedical Science and Engineering*. Web site. `http://www.physionet.org/physiotools/index.shtml`.

[59] Python, *The Python Programming Language*. Web site. `http://www.python.org/` (created by Guido van Rossum).

[60] SciPy, *SciPy - Scientific Tools for Python*. Web site. `http://www.scipy.org/`.

[61] Ubuntu, *Ubuntu*. Web site. `http://www.ubuntu.com` (distributed by Canonical Ltd.).

Index

Page numbers set in **bold** type indicate a principal or defining entry.

apnea, *see* obstructive sleep apnea
AR, *see* autoregressive (AR)
autoregressive (AR), 110

backward algorithm, 25
  for continuous states, 61, 69
backward forecast, 61
backward Kalman gain matrix, 70
backward update, 62
Baum, 31
Baum–Welch algorithm, 11, **31**, 34
Bayes' net, 16
Bayes' rule, 8
Bayesian estimation, 51
Benettin's procedure, 87
block matrix determinant, 117
block matrix inverse, 117

Cantor set, 83
chaos, 2
completing the square, 119
Computers in Cardiology 2000 (CINC2000), 97
conditional distribution, Gaussian, 118
cross entropy rate, 81

decoded state sequence, 12
delay vector, 17
determinant, *see* block matrix determinant
discrete hidden Markov model (HMM), 7

EM algorithm, *see* estimate maximize (EM) algorithm
entropy, 79
  conditional, 79
  cross, 80
  cross rate, 75
  differential of a continuous random variable, 79
  gap, **86**, 91
  Kolmogorov–Sinai, *see* Kolmogorov–Sinai entropy
  rate, 81
  relative, 80
  relative, as measure of fidelity, 80
  stretching, 82
ergodic, 37, **80**
estimate maximize (EM) algorithm, 31, **38**
extended Kalman filter, 3, **71**, 73

forecast, 60
forward algorithm, 11, **20**
  for continuous states, 60
forward backward algorithm, *see* Baum–Welch algorithm

Gaussian mixture model, 40, **50**
Gaussian observation, 48
Gibbs inequality, 42

Haken, 2
Hausdorff dimension, 84

information form, 71
initial states, probabilities of, *see* probabilities of initial states
Institute for Defense Analysis, xi
inverse covariance form, 65, 71

Jensen's inequality, 42
Juengling, Ralf, 122

Kalman filter, 64
Kalman gain matrix, 64, 68
Kalman gain matrix, backward, 70
Kolmogorov–Sinai entropy, **82**, 91

laser, 2
Ledrappier and Young's formula, 85
linear Gaussian systems, 62, 69
Lorenz system, 2, 73
Lyapunov exponents, 84

MAP, *see* maximum a posteriori (MAP) estimate
map of unit circle, 82
marginal distribution, Gaussian, 118
Markov assumption, 8
Markov chain, 7
Markov process, 7
matrix
  Kalman gain, 64, 68
  transition, 7
matrix inverse, *see* block matrix inverse
matrix, Kalman gain, backward, 70
maximum a posteriori (MAP) estimate, 29, 51, 66
maximum likelihood estimate (MLE), 4, 52
McNames, James, 97
MLE, *see* maximum likelihood estimate (MLE)
multiple maxima, 38
multiple segments, training on, *see* training on multiple segments

obstructive sleep apnea, 97
Oseledec's theorem, 84

partition, 81
Penzel, Thomas, 98
periodogram, 108
Pesin's formula, 85
PhysioNet, 97
point estimates, 38
predictability, 73
probabilities of initial states, 37
pruning transitions, 37

reestimation formulas, 33
regularization, 47, 54

sequence of maximum a posteriori state estimates, 29
Sherman–Morrison–Woodbury formula, 118
Sinai–Ruelle–Bowen (SRB) measure, 85
smoothing, 65, 71
spectrogram, 97
state space model, 1, 4, 59
stationary, 8, 37, 80
stochastic process, 79
Sylvester's inertia theorem, 45

Tang, D. Y., 2
toy problems, 73
training on multiple segments, 37
transition matrix, 7

unit circle, map of, 82
update, 61

Viterbi algorithm, 11, **27**

Weiss, C. O., 122
Wishart distributions, 54