Gaussian Processes for State Space Models and Change Point Detection

Ryan Darby Turner

Department of Engineering

University of Cambridge

A thesis submitted for the degree of

Doctor of Philosophy

July 17, 2011

Acknowledgements

I would like to thank Carl Rasmussen for advice throughout my PhD; Steven Bottone for extensive proof reading; Marc Deisenroth for assisting me in acquiring his LaTeX skills. I acknowledge my examiners Zoubin Ghahramani (internal) and Amos Storkey (external). I would like to thank my co-authors in the publications leading up to my thesis: Yunus Saatçi, Carl Rasmussen, and Marc Deisenroth. I also thank the other proof readers in my group: David Duvenaud, Alex Davies, Ferenc Huszár, Dan Roy, Peter Orbanz, and José Miguel Hernández Lobato. I would like to thank Rockwell Collins, formerly DataPath, Inc., which funded my studentship. I also acknowledge my college, Magdalene College, and my laboratory, the Computational and Biological Learning (CBL) laboratory.

Abstract

This thesis details several applications of Gaussian processes (GPs) for enhanced time series modeling. We first cover different approaches for using Gaussian processes in time series problems. These are extended to the state space approach to time series in two different problems. We also combine Gaussian processes and Bayesian online change point detection (BOCPD) to increase the generality of the Gaussian process time series methods. These methodologies are evaluated on predictive performance on six real world data sets, which include three environmental data sets, one financial, one biological, and one from industrial well drilling.

Gaussian processes are capable of generalizing standard linear time series models. We cover two approaches: the Gaussian process time series model (GPTS) and the autoregressive Gaussian process (ARGP). We cover a variety of methods that greatly reduce the computational and memory complexity of Gaussian process approaches, which are generally cubic in computational complexity.

Two different improvements to state space based approaches are covered. First, Gaussian process inference and learning (GPIL) generalizes linear dynamical systems (LDS), for which the Kalman filter is based, to general nonlinear systems for nonparametric system identification. Second, we address pathologies in the unscented Kalman filter (UKF). We use Gaussian process optimization (GPO) to learn UKF settings that minimize the potential for sigma point collapse.

We show how to embed mentioned Gaussian process approaches to time series into a change point framework. Old data, from an old regime, that hinders predictive performance is automatically and elegantly phased out. The computational improvements for Gaussian process time series approaches are of even greater use in the change point framework. We also present a supervised framework learning a change point model when change point labels are available in training.

These mentioned methodologies significantly improve predictive performance on the diverse set of data sets selected.

Preface

I assume the reader has knowledge of the manipulations of probability theory and calculus. Gaussian processes (GPs) are introduced in the text, but form a compressed introduction. Therefore, I recommend the reader has knowledge of the content in Rasmussen and Williams [2006]. Likewise, it is helpful they have knowledge of graphical models, conditional independence, and message passing contained in Bishop [2007, Ch. 8].

This thesis aggregates and extends content published throughout the course of my PhD. These publications are Turner [2010]; Turner et al. [2009b] (Section 5.1.1), Turner et al. [2009a, 2010] (Section 4.3), Turner and Rasmussen [2010] (Section 4.2), and Saatçi et al. [2010] (Section 5.3) as well as my first year report Turner [2008].

Given my presence as second author in Saatçi et al. [2010] I must elaborate on which portions of the paper are my contributions. The code and original idea for the paper were my original work except for the work with HMC covered in Section 5.3.2. All work on the extensions and improvements since the paper are a result of my own work.

Declaration This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated above through the relevant publications and their corresponding sections. Since some these publications were done in collaboration the term "we" will be used instead of "I."

Supervisor: Carl Edward Rasmussen
Advisor and internal examiner: Zoubin Ghahramani
Industrial supervisor: Steven Bottone
External examiner: Amos Storkey

Keywords Bayesian, Change point detection, filtering, Gaussian processes, machine learning, nonparametric, smoothing, state space, statistics, time series, Toeplitz, UKF.

Citation We have provided the `bibtex` for this thesis in order to make the citation process easier:

```
@PHDTHESIS{Turner2011,
  author = {Ryan Darby Turner},
  title = {Gaussian Processes for State Space Models and Change
    Point Detection},
  school = {University of Cambridge},
  year = {2011},
  address = {Cambridge, UK},
  month = {July}
}
```

Contribution Sections marked with ($\star$) are a core contribution of this thesis.

Brian Fantana: “They’ve done studies, you know. They say 60% of the time, it works every time.”
Ron Burgundy: “That doesn’t make any sense.”
$\sim$ Will Ferrell on conditional probability in *Anchorman: The Legend of Ron Burgundy* (2004).

Contents

List of Algorithms

List of Figures

List of Tables

Notation

We use different typeface for different objects. We write a scalar as x, a vector as $\mathbf{x}$, a matrix as $\mathbf{X}$, and a set as $\mathcal{X}$. The ith element of a vector is in the typeface of a scalar x_i. When changing typeface represents a notation collision we use the parenthetical form. For instance, the ith row and jth column of $\mathbf{X}$ as $\mathbf{X}(i, j)$. We represent an inclusive range between a and b as $a{:}b$. We will also use this to subsample a matrix: $\mathbf{X}(a{:}b, c{:}d)$. In the time series context, we use the shorthand of $(N) := (t-N){:}(t-1)$ for the last N elements before a time index t.

Sets	
$\mathbb{C}$	The complex numbers
$\mathbb{R}$	The real numbers
$\mathbb{Q}$	The rational numbers
$\mathbb{Z}$	The integers
$\mathbb{N}$	The natural numbers starting at 1
$\mathbb{N}_0$	The natural numbers starting at 0
$\mathbb{S}^D$	All positive definite matrices of size $D \times D$
$\mathbb{U}^D$	All upper triangular matrices of size $D \times D$
$\mathbb{L}^D$	All lower triangular matrices of size $D \times D$
$\mathbb{P}(x)$	The set of all polynomials in x
$\mathbf{M}(\mathcal{X})$	The set of all probability measures on a set $\mathcal{X}$
$\Pi(\mathcal{X})$	The set of all permutations on a set $\mathcal{X}$
$\mathcal{P}(\mathcal{X})$	The power set of a set $\mathcal{X}$
Linear algebra	
$\|\mathbf{x}\|_p$	The L_p norm of a vector $\mathbf{x}$, by default it is the L_2 norm
$\operatorname{diag}(\mathbf{X})$	Column vector containing the diagonal elements of sq. matrix $\mathbf{X}$
$\operatorname{tr}(\mathbf{X})$	The trace of a matrix $\mathbf{X}$
$\operatorname{chol}(\mathbf{X})$	The upper triangular Cholesky factorization of $\mathbf{X}$
$\mathbf{X}\backslash\mathbf{y}$	Use back substitution to perform $\mathbf{X}^{-1}\mathbf{y}$
$\mathbf{X} \odot \mathbf{Y}$	The Hadamard (element-wise) product of $\mathbf{X}$ and $\mathbf{Y}$
$\mathbf{X} \oslash \mathbf{Y}$	The Hadamard (element-wise) division of $\mathbf{X}/\mathbf{Y}$
$\mathbf{X} \otimes \mathbf{Y}$	The Kronecker product of $\mathbf{X}$ and $\mathbf{Y}$
$\mathbf{1}$	A vector of ones
$\mathbf{0}$	A vector of zeros
$\mathbf{I}$	The identity matrix
$\operatorname{vec}\mathbf{X}$	The vectorization of a matrix $\mathbf{X}$
$\mathbf{Ex}$	The exchange matrix reverses the order of a vector $\mathbf{x}$
$\mathbf{Dx}$	The differencing matrix differences a vector $\mathbf{x}$

$\mathbf{D}^{-1}\mathbf{x}$	The cum. sum matrix finds the cumulative sum of a vector $\mathbf{x}$
Toeplitz($\mathbf{v}$)	Defines matrix $\mathbf{T}$ such that $\mathbf{T}_{ij} = \mathbf{v}(\lvert i-j\rvert+1)$
	Information theory
$\mathrm{H}[x]$	The entropy of a random variable x
$\mathrm{KL}(p\lVert q)$	The Kullback-Leibler (KL) divergence between distn. p and q
$\mathrm{I}(x;y)$	The mutual (Shannon) information between variables x and y
$x \perp\!\!\!\perp y\vert z$	Random variable x is conditionally independent of y given z
	Probability distributions
$\mathbb{P}(\mathcal{D})$	The empirical distribution of a data set $\mathcal{D}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A Gaussian distn. with specified mean $\boldsymbol{\mu}$ and (co-)variance $\boldsymbol{\Sigma}$
$\mathcal{W}_n(\mathbf{V})$	A Wishart distn. with scale matrix $\mathbf{V}$ and n degrees of freedom
$\mathcal{GP}(\mu, k)$	A Gaussian process (GP) with mean function $\mu(\cdot)$ and kernel $k(\cdot,\cdot)$
$\mathrm{DP}(H, \alpha)$	A Dirichlet process (DP) with base measure H and concentration α
Laplace(μ, σ)	A Laplace distribution with mean μ and scale σ
DL(μ, γ)	A discrete Laplace distribution with mean μ and scale γ
$\mathrm{St}_\nu(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A m.v. Student's t distn. with mean $\boldsymbol{\mu}$, cov. $\boldsymbol{\Sigma}$, and ν degrees of freedom.
Gamma(α, β)	A gamma distribution with shape α and inverse scale β
$\mathcal{U}[a, b]$	A uniform distribution between a and b
$\log\mathcal{N}(\mu, \sigma^2)$	A log normal distn. that exponentiates samples from $\mathcal{N}(\mu, \sigma^2)$
	Miscellaneous
χ	A message in a message passing scheme
$\mathcal{S}$	A set of sufficient statistics
$x := y$	x defined as y
$x =: y$	y defined as x
$x \sim p$	Variable x is sampled from distribution p
$p(\cdot)$	A probability density on a continuous space, such as $\mathbb{R}$
$P(\cdot)$	A probability distribution on a discrete space, such as $\mathbb{Z}$
$\mathcal{O}(\cdot)$	The big-O asymptotic complexity of an algorithm
$\leftarrow$	An assignment operation in an algorithm
$\mathcal{F}\{f\}$	The Fourier transform of a function f
$f * g$	The convolution of functions f and g
$\mathbb{I}\{\cdot\}$	The indicator function, one if the statement in the braces is true and zero otherwise
SSE	Sum of (standardized) predictive square errors

Acronyms

Many acronyms are used throughout this thesis. We provide a list of acronyms used widely throughout the literature as well as those defined by the author.

Acronyms in common usage

ACF	autocorrelation function
AI	artificial intelligence
AIC	Akaike information criterion
AR	autoregressive (model)
ARD	automatic relevance determination
ARIMA	autoregressive integrated moving average
ARMA	autoregressive moving average
ARMAX	autoregressive moving average model with exogenous inputs
BIC	Bayesian information criterion
BMC	Bayesian Monte Carlo
CDF	cumulative distribution function
DP	Dirichlet process
DPM	Dirichlet process mixture
EKF	extended Kalman filter
EM	expectation maximization
FFT	fast Fourier transform
FIC	fully independent conditional
FITC	fully independent training conditional
GP	Gaussian process
GP-LVM	Gaussian process latent variable model
GPS	global positioning system
HMC	Hamiltonian Monte Carlo
KL	Kullback-Leibler (divergence)
KS	Kolmogorov-Smirnov (test)
LDS	linear dynamical system
MA	moving average
MAE	mean absolute error
MAP	maximum a posteriori
MCMC	Markov chain Monte Carlo
MDP	Markov decision process
MLE	maximum likelihood estimate
MSE	mean square error
OCR	optical character recognizer
ODE	ordinary differential equation

PAC	probably approximately correct
PACF	partial ACF
RBF	radial basis function
RJ-MCMC	reversible jump MCMC
RKHS	reproducing kernel Hilbert space
RMSE	root mean square error
RQ	rational quadratic (kernel)
RTS	Rauch-Tung-Striebel (smoother)
SDE	stochastic differential equation
SE	squared exponential (kernel)
SEM	stochastic expectation maximization
SMC	sequential Monte Carlo
SR	square root (filter)
UCB	upper confidence bound
UKF	unscented Kalman filter
UT	unscented transform
VAR	vector autoregression
VC	Vapnik-Chervonenkis
Novel acronyms	
ARGP	autoregressive Gaussian process
BLR	Bayesian linear regression
BOCPD	Bayesian online change point detection
CPD	change point detection
DIM	data independent model
DL	discrete Laplacian
GP-ADF	Gaussian process assumed density filter
GPIL	Gaussian process inference and learning
GPK	Gaussian process as Kalman (filter)
GPO	Gaussian process optimization
GPTS	Gaussian process time series (model)
GPYW	Gaussian process Yule-Walker (setup)
IFM	independent factor model
NLDS	nonlinear dynamical system
NLL	negative log likelihood
POE	probability of everything
SGP	sparse Gaussian process
TIM	time independent model
UPM	underlying predictive model

Chapter 1

Introduction

Machine learning is the study of algorithms whose performance improves with increased exposure to data. Most computer programs do not become more intelligent no matter how much data they process; their entire functionality is specified in advance by the programmer. A machine learning algorithm, by contrast, will become more intelligent as more data is processed. In machine learning, software must learn to match its output to training examples, where the correct output is provided to the algorithm. The classic example is optical character recognizers (OCR): Images of characters, say zero to nine, are provided and the software must output which digit is in the image. As opposed to trying to specify rules about what makes a two a two, example images are provided along with labels (this is known as the *training set*). A good machine learning algorithm will predict the correct character in a *test set* when novel images are provided to the algorithm. An OCR is an example of an *iid data set*, the images are independent of one another and their attributes do not change over time. The canonical examples of machine learning are iid, but we focus on *time series data*. Examples of time series data include air temperature or stock market returns.

Early on, machine learning researchers realized they had closer ties to statistics than other parts of computer science [Friedman, 1997]; functionality was being specified by real data not abstract specifications on paper, the original approach in artificial intelligence (AI), as is the case with data structures like stacks and queues. In contrast to an OCR, it is easy to specify on paper the requirements such that a sorting algorithm is correct. However, machine learning still maintains

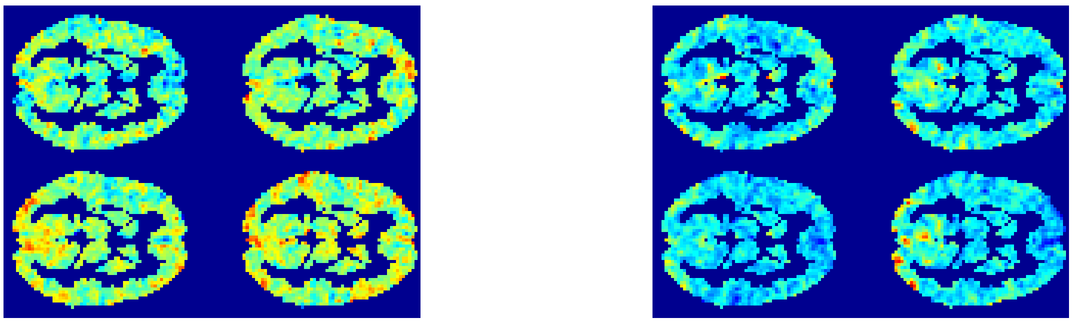

(a) Celery (b) Airplane

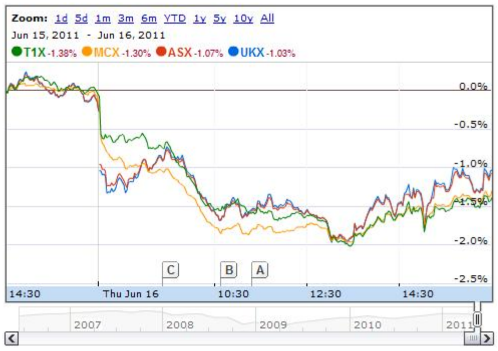
Figure 1.1: **Classification example**: fMRI images from subject one in Mitchell et al. [2008]. On the left we show four fMRI cross-sections when the subject was exposed to the word celery. On the right we show four cross-sections for airplane. In the context of classification, we would like to predict the word presented to the subject based on the fMRI image.

some independence from statistics. Machine learning is more concerned about new efficient algorithms for complex tasks while statistics is more concerned about "understanding" a data set. Central to machine learning is test set performance.

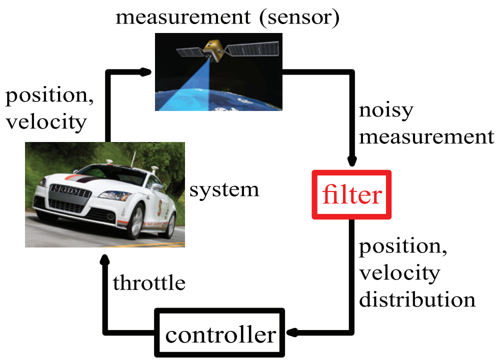
Most machine learning algorithms do not exist in a vacuum. They are usually tied to a given application area, which utilizes specialized domain knowledge. Some of these areas include speech applications, computer vision, bioinformatics, and finance [Mitchell, 2006]. We use an example classifying functional magnetic resonance (fMRI) images based on a stimulus instead of the more cliche example of numerical digits. This example uses *computational neuroscience* as the domain area; each domain comes equipped with its own prior knowledge.

In this example, we summarize the results of Mitchell et al. [2008]; fMRI data was taken from subjects while being presented with a variety of different nouns. In Figure 1.1, we show four cross-sections from one subject while either being presented with celery or airplane. After training on only 50 word-image pairs per subject, Mitchell et al. [2008] were able to use machine learning methods to predict whether a subject was presented with celery or airplane on novel new fMRI test set images with 70% accuracy on the average subject.

When a programmer is assigned the task of writing such a predictive program, certain tasks can be accomplished with standard computer science. The



(a) Time series prediction



(b) State estimation for control

Figure 1.2: Two examples of time series data. On the right we show a control system for an autonomous car [Montemerlo et al., 2008] using GPS as a sensor and throttle as a control signal. The filter combines measurements over time to reduce sensor noise. On the left we show data from the financial markets where the problem is more time series prediction than control.

programmer knows how to take the results and store them in a database, possibly make a web interface to the database, interface to the fMRI machine, and so on. The programmer does not know how to write one bothersome function. How do we take this array of N^3 voxels, the fMRI image, and output the word used as the stimulus? A programmer may try to elicit explicit rules from a neuroscientist about how to classify the images, but such approaches are limited without the use of real training data.

The task of identifying a word, for example, celery or airplane, from an image is called classification. In this case, it is binary classification. If we include a third word, e.g. corn, the task would become multiclass classification. If the task is to identify certain continuous valued statistics on the objects, such as the duration of a stimulus, then the task would be a regression problem. Chapter 3 focuses on a start-of-the-art nonlinear regression method known as Gaussian processes (GPs) [Rasmussen and Williams, 2006].

Although the iid examples like the fMRI classifier are the "bread-and-butter" of machine learning, this thesis focuses on time series examples. Figure 1.2(a) shows four indices of the Financial Times Stock Exchange (FTSE), a clear example for time series prediction; better modeling of this time series could lead to better portfolio management. Returns in financial markets are known to have volatility clusters [Lux and Marchesi, 2000], which makes it necessary to have models that respond to changes in variance. Change point models, discussed in Chapter 5, are useful in situations where there is a sudden increase in volatility possibly due to a new earnings report or political event.

Figure 1.2(b) illustrates a classic engineering problem: feedback control systems. The figure illustrates a feedback control system for an autonomous vehicle. The vehicle infers its position and velocity from a global positioning system (GPS) and odometer, and controls those variables via motor throttle. Industrially this is known as odometer assisted GPS [Geier, 1996]. The sensor measurements are noisy and therefore the vehicle must use multiple measurements over time to reduce the error in its state estimation. Simple averaging does not work since the vehicle is constantly moving, more clever methods must be used. When a feedback system is governed by stationary linear differential equations (also known as an LTI system) the Kalman filter [Kalman, 1960] is the optimal method. Methods involved in the nonlinear and nonstationary cases have been an active research area over the last few decades. Chapter 4 reviews common existing methods and introduces a few new approaches.

Example model A basic predictive model would model a scalar ($y_i \in \mathbb{R}$) by placing a normal-inverse-gamma prior on iid Gaussian observations:

$$y_i \sim \mathcal{N}(\mu, \sigma^2)\,, \tag{1.1}$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma^2/\kappa)\,, \quad \sigma^{-2} \sim \text{Gamma}(\alpha, \beta)\,. \tag{1.2}$$

In this example the parameters are $\theta = \{\mu, \sigma^2\}$ and the model hyper-parameters are $\{\mu_0, \kappa, \alpha, \beta\}$. This gives a Student's t predictive distribution $p(y_{N+1}|y_{1:N})$. In later chapters this model is referred to as the time independent model (TIM) since all the data points are independent conditional on the parameters: $y_i \perp\!\!\!\perp y_j|\theta$.

TIM is a density estimation (unsupervised or output only) model. However, we may have input variables as well; for instance, in the fMRI example the image is the input $x \in \mathcal{X}$, where $\mathcal{X}$ is the set of all N^3 voxel images. In the unsupervised case $\mathcal{X} = \emptyset$. When we incorporate input variables we are in the supervised regime.

The natural extension of TIM to the supervised case is linear regression:

$$y_i \sim \mathcal{N}(ax_i + b, \sigma^2)\,. \tag{1.3}$$

The parameters are now $\theta = \{a, b, \sigma^2\}$.

1.1 Contributions

In this thesis we develop advanced methods for temporal data. We compare our methodology to the standard linear approaches such as autoregressive (AR), moving average (MA), and Kalman filter methods as well some nonlinear extensions. TIM is a naive model that will serve as a reference point or "baseline" for performance. The bulk of the thesis' material is found in three chapters: Gaussian processes, state space models, and change point detection. We summarize how each of these elements — Gaussian processes, state space, and change points — combine to form this thesis' contribution in the Venn diagram of Figure 1.3.

In Chapter 3, we give a brief introduction to Gaussian processes (GPs) before moving on to advanced aspects. Gaussian processes are a nonlinear regression method, extending (1.3), that do not assume the function underlying the relationship between x and y can be represented by an analytic form. They belong to a class of methods known as *Bayesian nonparametrics*. We review two ways of accounting for extra sources of uncertainty in GPs: uncertain inputs and output scale uncertainty. In uncertain inputs, the input variable x is known only up to a Gaussian distribution. We must "hedge our bets" when making predictions on the outputs y in that case. In output scale uncertainty we treat the scale, or units, of the outputs y as unknown and they must be inferred from data. This is sometimes known as a "t-process." We make contributions by offering a simplified derivation of the uncertain input equations and presenting updates to do inference in the t-process for the same computational cost as a standard Gaussian process.

There are two approaches to applying Gaussian process methods to time series: Gaussian process time series (GPTS) and the autoregressive Gaussian process (ARGP). We compare these two approaches: The GPTS generalizes many of the

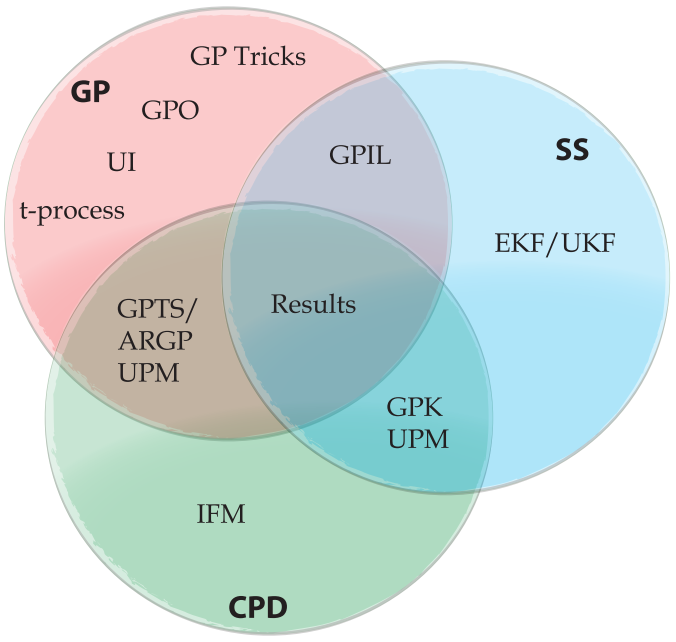


Figure 1.3: A Venn diagram representing the contributions and topics in this thesis. We present the areas as change point detection (CPD) (Chapter 5), Gaussian processes (GP) (Chapter 3), and state space models (SS) (Chapter 4). Gaussian process optimization (GPO) (Section 4.2.1), GP uncertain inputs (UI) (Section 3.2), and uncertain output scale (the "t-process") (Section 3.1) are in the GP circle. Gaussian process inference and learning (GPIL) (Section 4.3) is both Gaussian processes and state space modeling. Gaussian process change point models (Section 5.3) cover Gaussian processes and change point detection. Whereas, the independent factor model (IFM) is only in change point detection. Improvements to the unscented Kalman filter (UKF) (Section 4.2), and the related EKF are in state space modeling. Finally, the results section (Section 5.5) evaluates methodologies in all circles.

standard linear time series methods, while the ARGP is arguably more general still than the GPTS. However, some elegant properties of the GPTS are lost when moving to an ARGP. The ARGP is also more computationally difficult than the GPTS. We review and extend methods to bring down the computational penalty of GPTS using Toeplitz methods or approaches that convert a GP to a Kalman filter (GPK). Rank-1 matrix updates bring down the cost of ARGP. We extend Rank-1 updates with a novel method called the *sub-evidence* that allows us to cheaply make predictions at any point in a time series for any starting point of the training data.

Chapter 4 emphasizes state space models. We review existing work in state space models such as the Kalman filter. Approximate extensions designed for systems governed by nonlinear differential equations such as the extended Kalman filter (EKF) and unscented Kalman filter (UKF) are presented in a unifying framework for approximate filters. We cover some of the failure modes of these approximations, i.e. poor predictive performance, and introduce model based methods to make such failure less likely. We utilize GPs through the use of Gaussian process optimization (GPO) for learning in this setup. GPs are also used in a state-of-the-art nonlinear filter known as the GP assumed density filter (GP-ADF), which utilizes the uncertain input methodology covered in Chapter 3. We extend the GP-ADF by learning the system dynamics, in effect the governing differential equations, of time series with the Gaussian process inference and learning (GPIL) algorithm. Like a GP in a regression context, the GPIL does not assume the system can be represented by an analytic form. We empirically benchmark these approaches.

Finally, in Chapter 5 we extend the change point detection framework first presented by Adams and MacKay [2007] and Fearnhead and Liu [2007]. In a change point detection framework, we phase out old training data that is detrimental to the performance of an underlying model. In the absence of change points, we would expect more data to only increase the performance of an algorithm. We first build upon this approach for *hyper-parameter* learning. We learn from training data quantities that were previously set by hand: What is the *hazard function*, how often do change points occur? What is the prior of the underlying model, i.e. what kinds of changes typically happen at change point? In extension of the previous approaches that assumed TIM-like or simpler linear models as the underlying model, we "plug-in" the Gaussian process based methods, GPTS and ARGP, as the underlying model. The computational complexity can be greatly reduced using the methods of Chapter 3: Toeplitz methods, GPK, and the sub-evidence. We can do this in both a streaming way, as the data is coming in, and honing our inferences in retrospect by extending the methods of Fearnhead [2006]. Our final contribution is a method for supervised change point detection that uses labeled known change points in training. We also incorporate label noise on the change point labels. Finally, we evaluate all of our methods and the standard approaches (over ten methods in all) on six real world data sets. The data sets, summarized in Chapter 2, are environmental applications (Whistler snowfall, Nile record, fish killer), biology (bee waggle dance), geological engineering (well log), and finance (industry portfolios).

Measuring performance Throughout the machine learning literature, and this thesis, methods are motivated by performance increases. Therefore, we must precisely define how we measure performance. For iid problems, we randomly take N data points from our data set and place them in the training set, while the remaining N' are placed in the test set. The parameters of a method are set using the training set, while the predictive accuracy of a method is measured on the test set according to a loss function. For time series methods, we generally train on the first T time steps and test on the remaining T' time steps; it is more realistic to test a model's ability to predict the future based on the past than vice-versa.

Consistent with rational decision making we consider the average loss over each data point, where the loss function $L \in \mathcal{Y} \times \mathcal{A} \to \mathbb{R}$ measures the compatibility between the actual data point y and an action $a \in \mathcal{A}$. The average loss over each test point is known as the empirical loss:

$$\widehat{L} := \frac{1}{N'} \sum_{i=1}^{N'} L(y_i, a_i)\,. \tag{1.4}$$

The hypothetical limiting case is known as the generalization error: $\mathcal{L} := \lim_{N' \to \infty} \widehat{L}$. We obtain actions a from predictive probability distribution $p(y_i|x_i)$ by minimizing the expected loss,

$$a_i = \operatorname*{argmin}_a \mathbb{E}\left[L(y_i, a)\right] = \operatorname*{argmin}_a \int L(y_i, a) p(y_i|x_i)\, dy_i\,. \tag{1.5}$$

This procedure is known as Bayes' decision rule. The black box functionality of a machine learning method is shown in Figure 1.4.

For continuous variables $y \in \mathbb{R}$ we consider the mean-square-error (MSE),

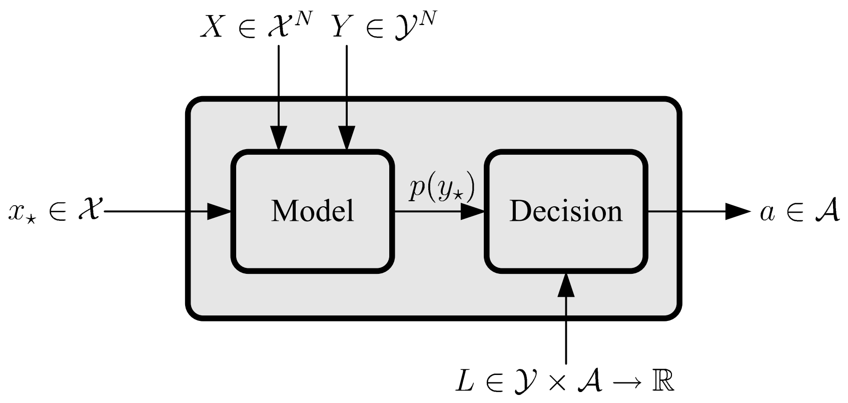


Figure 1.4: Black box model of a machine learning algorithm. The algorithm is "configured" by a set of training inputs X and outputs Y. Then for any test input $x_\star$ it outputs an action a. Inside the black box we have drawn two smaller black boxes which exist if a Bayesian decision process is being used. The first inner black box provides a predictive probability distribution $p(y_\star)$ according to a model. The second inner black box implements (1.5).

$L(y_i, a) = (y_i - a)^2$, and the mean-absolute-error (MAE), $L(y_i, a) = |y_i - a|$. The optimal actions for MSE and MAE are to assign a the mean and median of p, respectively. We will also use the log loss, also called negative log likelihood (NLL), $L(y_i, p) = -\log p(y_i|x_i)$, which has special properties for evaluating the entire predictive distribution p. More complex loss functions are possible for more complex tasks such as ranking, predictive intervals, resource allocation, and so on. A more thorough background and motivational aspects of performance evaluation is presented in Section B.1.

Bayesian machine learning Researchers in machine learning often borrow ideas from Bayesian statistics. A Bayesian approach allows us to apply the rules of probability in a unified way to the learning, prediction, and inference tasks. The probability of an event is treated as betting odds rather than a hypothesized limiting frequency in infinite independent trials (the frequentist paradigm). If we take probabilities to be betting odds, de Finetti [1931b] showed that the laws of probability are the unique rules that allow a bookie to avoid being placed in Dutch books: scenarios where a bookie takes a collection of bets where he will lose money in every possible outcome.[1] See Pollard [2002, Sec. 1.5] for details.

In statistics, a common alternative to the Bayesian paradigm is hypothesis testing. Consider a celery detector in the fMRI example, for a Bayesian celery detector, we say the probability that the stimulus is a celery, *given the image we observed*, is $x\%$. The analog of this statement in hypothesis testing (frequentist) terms would advertise that the probability of classifying the image as a celery stimulus *given it is really from an airplane* is less than α, usually 5% (this is known as a false positive rate). A practitioner would most likely use hypothesis testing for a model selector: The probability of observing some statistic of the data set, or more extreme in some sense, given it was actually generated from some proposed model where airplanes and celery give identical observations is less than 5%. We can illustrate the difference using an extreme case reasoning: A celery detector, or model selector, that completely ignored the data and randomly alerted 5% of the time would satisfy the frequentist statement.[2]

In machine learning, the generalization error is of primary importance. The frequentist optimality criterion here is the *minimax* risk: the "true" probability distribution of the data has been adversarially picked by nature to trick the algorithm. There is often no clear way to obtain the minimax optimal algorithm and bounds to the generalization error are often used as a proxy, for instance in Vapnik-Chervonenkis (VC) analysis [Vapnik and Chervonenkis, 1971]. This is in contrast to hypothesis testing; the statistical learning theory approach (such as VC analysis) emphasizes generalization error bounds. For any arbitrary sampling distribution, we could generate many independent data sets and only in less than 5% of them would the misclassification of airplanes and celery on the test set be more than ϵ. The bound ϵ can be calculated from the training set. A Bayesian machine learning algorithm has optimal Bayes' risk: We get the lowest expected loss in expectation under the prior. A more in-depth coverage of these issues is in Section B.2.

[1] Websites such as `www.betfair.com` and `www.intrade.com` are modern manifestations of the thought experiment in de Finetti [1931b]. If the odds available to those making simultaneous bets were not *coherent*, i.e. consistent with the rules of probability, it would be possible to make a risk free profit through arbitrage.

[2] The concept of statistical power was developed to exclude such tests. However, power will be the function of a true latent parameter. Only in simple situations will there be a test that is uniformly most powerful: more powerful than other tests for all values of a latent parameter.

1.2 Nonparametric Methods

Nonparametric methods allow the number of parameters to grow with the sample to better fit the data.[1] In Bayesian methods, we must specify a prior on the parameters before we observe the data, or even the sample size, which precludes us from changing the number of parameters as the sample size grows. Ferguson [1973] observed that the solution to this dilemma is for nonparametric Bayesian methods to place priors directly on infinite dimensional objects. We must make the "parameters" θ infinite dimensional.[2] For instance, they can be functions ($\mathbb{R}^D \to \mathbb{R}$) as in Gaussian processes, over probability distributions $\mathbf{M}(\mathbb{R}^D)$ as in Dirichlet processes (DPs), or over infinite binary matrices ($\mathbb{N} \times \mathbb{N} \to \{0, 1\}$) as in the Indian buffet process. A corollary of the parameters θ being infinite dimensional is that the data cannot be summarized by any bounded finite number of sufficient statistics as the data set grows. In fact, Hipp [1974] showed that only finite dimensional exponential family models summarize the data with sufficient statistics of bounded dimension. In other words, when making predictions we need to keep all the training data in memory; we cannot summarize the data by its sample mean and variance as in the parametric Gaussian case, i.e. TIM. We illustrate the more complex distributions that can be expressed by a nonparametric approach in Figure 1.5.

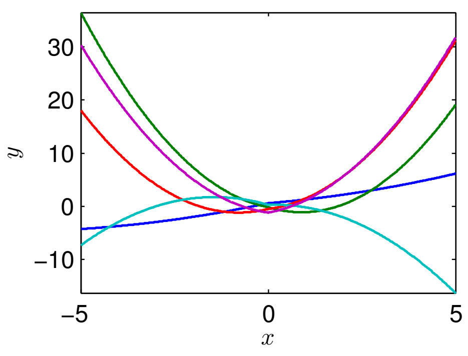
Nonparametric methods are often motivated by making some set of parameters in a parametric model infinite. For instance, Gaussian processes can be motivated as Bayesian linear regression with an infinite number of basis functions. Likewise, Dirichlet process mixtures (DPMs) are motivated as mixture models where the number of mixture components M become infinite. Paradoxically, this often makes the algorithms more efficient. This is often because the computational complexity $\mathcal{O}(\cdot)$ of a parametric (finite) model contains the number of components M. Using nonparametric methods forces the modeler to use an efficient representation that is invariant to M since doing otherwise would

[1] In "All of Nonparametric Statistics" Wasserman [2006, Ch. 1] infamously does not define nonparametric statistics as to avoid inviting controversy. We take a more bold approach.

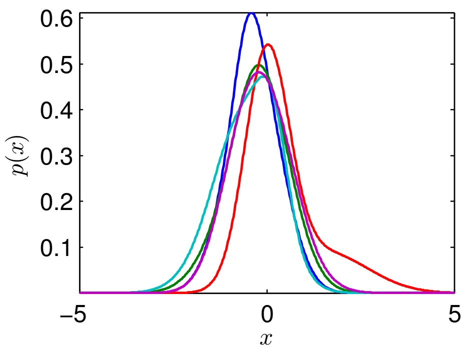
[2] The term nonparametric causes quite a deal of confusion as it often leads people to believe we are working with models with zero parameters when in fact we are referring to models with an infinite number of parameters!

cause the computational complexity to become infinite when working with the nonparametric version.

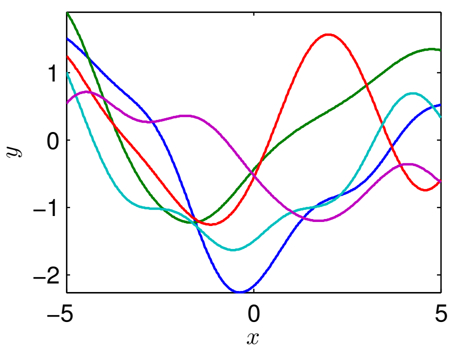
A common question to ask is whether nonparametric models offer any advantage over their parametric counterparts when the number of components M is relatively large. In the nonparametric model the number of parameters (infinite) is always much larger than the number of data points; in the parametric case, there is always *some* data set N where the parameters no longer outnumber the data points. Realistically, we may never reach that regime. However, recall Ferguson [1973] noted that in a perfectly coherent Bayesian setting the number of data points N should not affect our model choice. Often, it is merely more elegant to let the number of parameters become infinite than pick some large, but still arbitrary, number.



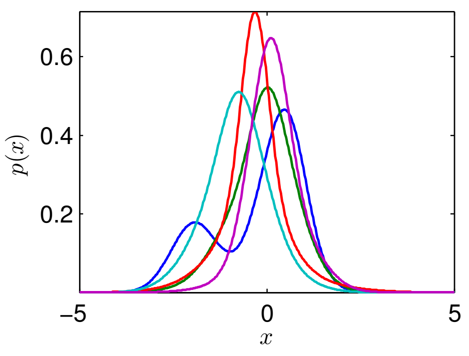
(a) Three basis function linear regression



(b) Three component mixture of Gaussians



(c) Gaussian process



(d) Dirichlet process mixture

Figure 1.5: Sample draws illustrating the difference between nonparametric and parametric distributions. The top row shows parametric models while the bottom row shows their nonparametric extension. The left column is for regression while the right column is density estimation.

1.3 Overview of Time Series Data

In this section we describe the time series data paradigm, challenges in modeling real world time series, and the two main modeling approaches. All probabilistic time series models are based on either the autoregressive or the state space approach (known as data driven and parameter driven models, respectively, in econometrics [Francis et al., 2011]). Accounting for many aspects of real world time series, such as periodicity and missing data, is an often ignored part of the literature, but is important in application. We build on this section's introduction to the autoregressive approach in Chapter 3. Likewise, we build on the state space approach in Chapter 4.

Time series are quantities that vary through time. They often consist of continuous measurements y_t that can be sampled at any point in time, arbitrarily. Examples include circuit voltage, atmospheric temperature, and stock market returns. In more precise terms, time series data maps from a time index $t \in \mathcal{T}$ to a measurement $y_t \in \mathbb{R}$. Time series can be in discrete time ($\mathcal{T} = \mathbb{Z}$) or continuous time ($\mathcal{T} = \mathbb{R}$), meaning discrete time is a special case of continuous time. For example, if we measure the voltage exactly once per hour we model it as discrete time where the time index is how many samples have been taken so far. We call this case *uniform sampling*. However, if voltages are measured at arbitrary times or with changing sampling rates it is more convenient to model it as a continuous time process. To simplify the analysis we often, but not always, focus on discrete time processes.

If we are looking at multivariate time series, for example if we are considering a circuit's voltage and current in the same model, then it is easier if we have *synchronized sampling*. The sampling is synchronized if the different quantities, also referred to as *channels*, are sampled at the same time. In the multivariate case, $\mathbf{y}_t \in \mathbb{R}^D$, we treat a time series as discrete if the sampling is uniform and synchronized. Otherwise, it makes more sense to use continuous time models. Note that discrete time still allows for missing data. If the data is sampled at one minute intervals but occasionally there are two or three minute gaps between data points then we model it as discrete time with missing data.

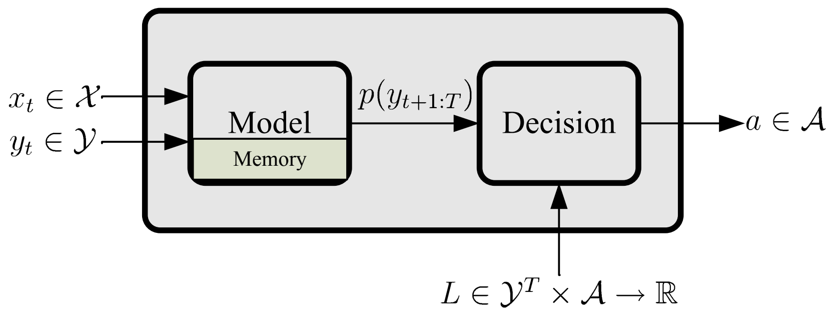
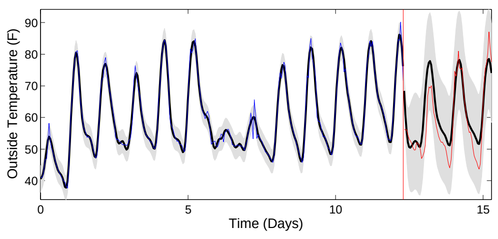


Figure 1.6: Black box model of a time series method. This is the temporal analog to the iid black box in Figure 1.4. The model is the same as in the earlier iid black box except we now have a memory unit drawn on the bottom of the model. This signifies that information from recent data points is stored for predictions. The time series comes in one step at a time in y_t along with its *external inputs*, also known as *covariates* or *control inputs*, x_t.

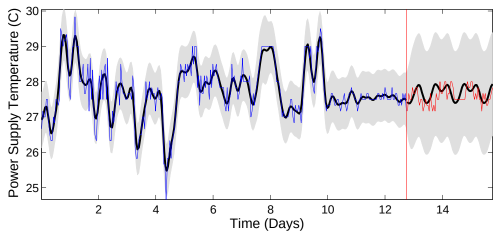
1.3.1 Challenges in Time Series Data

Throughout this thesis we discuss approaches to handling features often found in real world time series. In this section we give an overview of the challenges often present. These challenges are often ignored since they are not theoretically interesting, and are therefore under emphasized. However, we motivate aspects of the models constructed in later chapters based on these ideas. Furthermore, we take these challenges into consideration during the evaluation of the methods. The challenges in this section are illustrated on a real example in Figure 1.7.

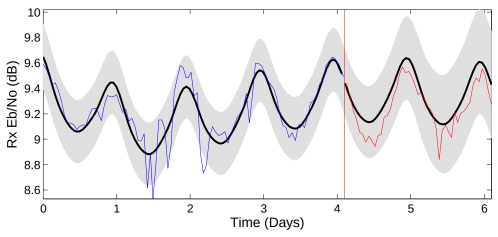
Trends The most common time series feature to account for is trends. For instance, if there is a clear linear trend in the time series, it is inadvisable to use a model that aggressively reverts to the mean when extrapolating. However, simple linear models are even worse for extrapolating as they assume a time series will keep increasing or decreasing forever. Trends can also be removed through differencing. Exponential trends can sometimes be elegantly handled by moving



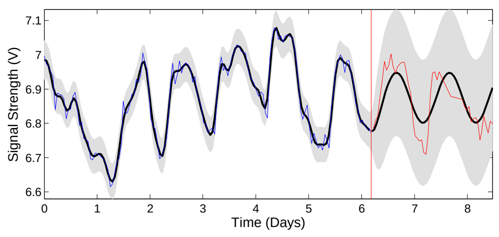
(a) Outdoor Temperature



(b) Power Supply Temperature



(c) Receiver Energy



(d) Signal Strength

Figure 1.7: We illustrate some of the challenges of real time series data on four examples from an electronic communication system. In each of the figures, the blue line represents the time series in the past while the red line represents it in the future. The black line and gray region represent the mean and 95% error bars in an example time series analysis on each time series; The region after the vertical red line is a forecast based on the data before the red line. Periodicity, at a daily period, is present to some degree in all the examples. Likewise, all the time series have some degree of a smooth underlying pattern, a short term correlation, not explained by the periodicity. The outside temperature and power supply temperature show obvious change points. The power supply temperature is visibly discretized to 1 C levels.

to the "return" space:

$$r_t := \log\left(\frac{y_t}{y_{t-1}}\right) = \log(y_t) - \log(y_{t-1})\,. \tag{1.6}$$

The term return space is used as this formula converts between prices and returns when working with financial data. Using Gaussian process methods, covered in Chapter 3, the model can automatically and implicitly difference, or move to return space, time series when appropriate. We can strike an appropriate balance between aggressive extrapolation and mean reversion in a model based manner.

Differencing the time series adds noise in some sense. If each y has additive

Gaussian noise then the noise variance in $y_t - y_{t-1}$ is twice that in y_t. However, there is no loss in information as differencing is an invertible operation. A model on the differenced time series merely implies a different model on the original time series. We can work with the differencing level that is most compatible with a given model. Therefore, although differencing increases the “noise level” differencing does not necessarily lower performance.

Periodicity Many real world time series involve periodic patterns, especially on natural time scales, such as days, weeks, or years. A common approach to periodicity is to look at the Fourier transform of the time series and find frequencies with large power. Sinusoids at this frequency are then removed (band-pass filtering) [Baxter and King, 1999]; however, this requires setting arbitrary thresholds. Differencing over the time of a period can also be applied. These ad hoc approaches can introduce artifacts and be counter-productive [Nelson and Kang, 1981]. In Chapter 3 we illustrate how model based approaches to handle periodicity are more elegant and effective. If the periodicity occurs in multiplicative fashion, preprocessing with a log-scale transform can be useful.

Outliers Outliers could be the result of an unusual effect in the actual process being observed. Financial returns are notorious for being “heavy tailed” and many traders have been ruined for making erroneous Gaussian assumptions [Crouhy et al., 2005, Ch. 14]. However, the outliers in question may simply be “junk data.” For instance, missing data may be mistakenly entered as a measurement of zero due to careless programming. Outliers may also be the result of sensor anomalies, where a sensor gets stuck at a certain value. Machine learning methods for sensor failure detection were explored in Osborne et al. [2010].

Standardizing In general, it is advisable to standardize (linearly transform the data $\mathbf{y}_{1:T}$ to have mean zero and unit variance) a time series before applying a method to it. Firstly, it may help if a method is not appropriately scale invariant. Additionally, standardization may alleviate numerical problems. Data whitening [Bishop, 2007, Ch. 12] can be applied to multivariate data to make each channel uncorrelated.

Standardizing allows the data to subtly influence the prior, which is illegal in a strict Bayesian formalism. In other words, standardizing induces a data dependent prior. For instance, placing the mean of the prior of the data at zero and then standardizing the data is the same as placing the prior at the sample mean of the data.

If we are willing to learn the mean by maximum likelihood anyway then standardizing does nothing more than avoid potential numerical problems. If we are trying to do full Bayesian inference, standardizing the data will usually only give the prior a small peek at the data and make little difference to the resulting inferences. However, recent developments suggest this is not always the case in nonparametric models such as Dirichlet process mixtures [Darnieder, 2011].

If the training set is continuously growing, we want our methods to learn the scale of the data. They should be robust enough to work without any standardizing. However, we can standardize the data to some initial estimate of the scale of the data anyway, to limit the potential for numerical difficulties.

Short-term correlations Short-term correlations can be visualized by autocorrelation functions (ACF), partial ACFs (PACF), and scatter plots. The ACF on a time series $\mathbf{y}_{1:T}$ is defined as:

$$\mathrm{ACF}(p) := \mathrm{Corr}\left[y_t, y_{t-p}\right] = \frac{\mathrm{Cov}\left[y_t, y_{t-p}\right]}{\sqrt{\mathrm{Var}\left[y_t\right]\mathrm{Var}\left[y_{t-p}\right]}}\,, \tag{1.7}$$

and the PACF is:

$$\mathrm{PACF}(p) := \mathrm{Corr}\left[y_t, y_{t-p} | \mathbf{y}_{t-p+1:t-1}\right] \tag{1.8}$$

$$= \frac{\mathrm{Cov}\left[y_t, y_{t-p} | \mathbf{y}_{t-p+1:t-1}\right]}{\sqrt{\mathrm{Var}\left[y_t | \mathbf{y}_{t-p+1:t-1}\right]\mathrm{Var}\left[y_{t-p} | \mathbf{y}_{t-p+1:t-1}\right]}}\,. \tag{1.9}$$

The methods of Chapter 3 fit "hand-in-glove" for directly modeling short-term correlations.

Change points Parameters in the generative process may suddenly change. Typically we would expect that the more data is observed the better a model will perform at predicting. However, in time series analysis there are times in practice where using old data is counter-productive. Many ad hoc approaches [Anagnostopoulos et al., 2008] have been developed to throw away old data, most of which involve placing some form of "forgetting factor" in a parameter estimator to implement a *Robbins-Monro* like scheme [Robbins and Monro, 1951]. The methods in Chapter 5 automatically phase old data out as it becomes less accurate in making predictions. Such change points methods work in a fully coherent Bayesian manner without resorting to approximate inference.

Missing data Missing data is the source of many real data statistical headaches. Substituting zero or some other value for missing data is a naive, yet sometimes used method. An advantage of a fully probabilistic approach is that it allows for a principled way to deal with missing data. A missing variable is simply marginalized out instead of conditioned on. A matter of concern is the *missing at random* assumption [Little and Rubin, 1987]. Does observing the data point as missing provide information of its latent value? This issue arose in the NetFlix challenge and was studied by Marlin et al. [2007]. When NetFlix users are rating movies, the fact that they have not rated a movie means they have decided not to watch it. Contrary to the missing at random assumption, this provides information that they may not rate the movie highly. We emphasize the distinction between the *observation* that data is missing and the *assumption* that it is missing at random.

Censored data Sensors often have minimum and maximum values that are sometimes exceeded in the data. For instance, a thermometer might only return data in a range of 0 C to 100 C. Properly modeling the observation using $P(x \geq 100)$ may give much different results than $p(x = 100)$.

Discretization Sensor data is often coarsely discretized to Δx intervals. A sensor might be designed to only report as many significant digits as it is confident in. For instance, a thermometer might only report to the nearest $\Delta x = 1$ C. Numerical and other problems can result when applying a method that expects the data to be continuous to discrete data. For instance, a long stream of identical measurements x, which should not happen in continuous data, may cause problems when attempting to infer the variance.

The most principled approach is to consider the latent value of the continuous time series prior to discretization x_t^{cont}, which is within a certain range of the observed discretized value x_t. Meaning we work with $P(x_t^{\text{cont}} \in [x_t - \Delta x/2, x_t + \Delta x/2])$ rather than $p(x_t^{\text{cont}} = x_t)$. We can also add a minimum noise level in the model according to the discretization level Δx. However, a simple fix that works with any black box method, is to add *salt* to the data x to arrive at salted data s:

$$s_t := x_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{U}[-\Delta x/2, \Delta x/2]\,. \tag{1.10}$$

The more rigorous approach would be to create many salted versions of a training set and average the predictions. However, a singly salted method may give approximately the same answer if the added noise is very small relative to the signal.

We justify salt by the following setup. Suppose we are given a black box that computes $p(\theta|\mathbf{x}_{1:T}^{\text{cont}})$, possibly the posterior on some parameters or a posterior predictive. However, since we only have $\mathbf{x}_{1:T}$ we must use this black box to compute $p(\theta|\mathbf{x}_{1:T})$. We see that

$$p(\theta|\mathbf{x}_{1:T}) = \int p(\theta|\mathbf{x}_{1:T}^{\text{cont}})p(\mathbf{x}_{1:T}^{\text{cont}}|\mathbf{x}_{1:T})d\mathbf{x}_{1:T}^{\text{cont}}\,, \tag{1.11}$$

$$p(\mathbf{x}_{1:T}^{\text{cont}}|\mathbf{x}_{1:T}) \propto P(\mathbf{x}_{1:T}|\mathbf{x}_{1:T}^{\text{cont}})p(\mathbf{x}_{1:T}^{\text{cont}}) \tag{1.12}$$

$$\overset{(1.10)}{=} p(\mathbf{x}_{1:T}^{\text{cont}}) \prod_{t=1}^{T} \mathbb{I}\{x_t^{\text{cont}} \in [x_t - \Delta x/2, x_t + \Delta x/2]\}\,. \tag{1.13}$$

If Δx is small enough such that the prior predictive $p(\mathbf{x}_{1:T}^{\text{cont}})$ is approximately constant within the small region near $\mathbf{x}_{1:T}$ then the posterior will be approximately uniform:

$$p(\theta|\mathbf{x}_{1:T}) \approx \int p(\theta|\mathbf{x}^{\text{cont}}_{1:T}) \prod_{t=1}^{T} \mathcal{U}[x^{\text{cont}}_t| - \Delta x_t/2, \Delta x_t/2] d\mathbf{x}^{\text{cont}}_{1:T} \tag{1.14}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} p(\theta|\mathbf{s}^i_{1:T}) \,, \tag{1.15}$$

where s^i_t is the ith out of N iid draw of s_t from (1.10). We arrive at the salted algorithm by averaging over many salted versions, which effectively reduces the variance from Monte Carlo error in (1.10).

Non-uniform sampling Data analysis is simplest when a sensor is sampled on a periodic interval, e.g. exactly every 600 s. Uniform sampling makes the data naturally suited towards discrete time methods. However, this is not always the case in real world data, which gives an advantage to time series methods that work in the continuous time domain.

Point masses Point probability masses often occur in what might be assumed as a continuous variable. For instance, we could treat daily snowfall as a continuous variable with a smooth density. However, there is a positive probability of seeing exactly 0 cm of snowfall in a day. In this case, we might want to use a classifier to estimate the probability that it snows paired with a regression method to predict the log snowfall conditional that it does snow.

Summary This description of challenges is by no means exhaustive, but should have sufficient coverage. Details of certain challenges given an emphasis in this thesis are: periodicity, short-term correlations, and change points. Discretization is present to some degree in all data sets. Discretization is often, but not always, neglected. In later chapters, we discuss continuous time models that can handle non-uniform sampling without difficulty. These challenges provide a context for other parts of the thesis.

Much of the literature on these challenges in machine learning is augmented by similar literature in signal processing, for instance: censored data [Olofsson, 2005], a review of the long studied connection between the ACF/PACF and the frequency domain [Oppenheim and Schafer, 1989], a review of "robust" signal processing for handling outliers [Kassam and Poor, 1985], extensions to non-uniform sampling [Marvasti, 2001], and handling missing data [Cooke et al., 2001].

1.3.2 Modeling Approaches

Discrete time data is modeled using either *autoregressive* or *state space* approaches. State space models are based on the notion that there is an unobserved true state of the system, or latent state, evolving over time that can only be observed indirectly. For example, the location of a car can only be observed through noisy observations from either an odometer or a GPS. Tracking and control as well as unsupervised tasks like visualization or time dependent dimensionality reduction depend on the notion of a state space. Other applications, such as finance, are more interested in predicting future observations than finding the latent state. There, the latent state is merely a tool, which may or may not be beneficial for prediction.

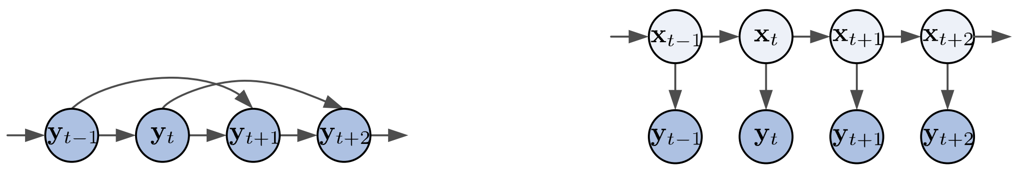
The most basic state space model with continuous valued latent state is the linear dynamical system (LDS), which is the discrete time analog of a linear differential equation. The hidden Markov model (HMM) [Baker, 1975] is the discrete space analog of an LDS. LDS methods treat predictable nonlinear behavior as noise to a large degree. Many approaches have been invented to make predictions in nonlinear dynamical systems (NLDS), Section 4.1.3.

To make the state space/autoregressive dichotomy concrete, suppose we have a time series of length T, $\mathbf{Y} := [\mathbf{y}_1, \ldots, \mathbf{y}_T]$. In an autoregressive approach we directly model the function mapping the last p values[1] of $\mathbf{y}$ to the next value of $\mathbf{y}$. A model of the form:

$$\mathbf{y}_t \sim p(\mathbf{y}_t | \mathbf{Y}_{(p)}, \theta) \tag{1.16}$$

is an autoregressive model of order p with parameters θ. We typically restrict

[1] We use the shorthand of $(N) := (t - N){:}(t - 1)$ for the last N elements before a time index t.



(a) Autoregressive model (b) State space model

Figure 1.8: Graphical models for autoregressive and state space models. The autoregressive model shown here happens to be of order two. These graphical models show the difference in conditional independence assumptions between the two approaches. The dark blue shading signifies an observed node (variable) while the light blue signifies a latent variable.

ourselves to consider the case where:

$$p(\mathbf{y}_t|\mathbf{Y}_{(p)},\theta) = \mathcal{N}(f(\mathbf{Y}_{(p)}),\Sigma)\,, \tag{1.17}$$

for some function $f \in \mathbb{R}^{D\times p} \to \mathbb{R}^D$ and noise covariance $\Sigma \in \mathbb{S}^D$. By contrast, a state space model would be of the form:

$$\mathbf{x}_t \sim p_f(\mathbf{x}_{t-1}|\theta)\,, \quad \mathbf{y}_t \sim p_g(\mathbf{x}_t|\theta)\,, \tag{1.18}$$

where typically only $\mathbf{y}_t$ is directly observable. Similar to the autoregressive case, we typically restrict ourselves to consider the case where:

$$p_f(\mathbf{x}_t|\mathbf{x}_{t-1},\theta) = \mathcal{N}(f(\mathbf{x}_{t-1}),\Sigma_f)\,, \quad p_g(\mathbf{y}_t|\mathbf{x}_t,\theta) = \mathcal{N}(g(\mathbf{x}_t),\Sigma_g)\,, \tag{1.19}$$

where $f \in \mathbb{R}^D \to \mathbb{R}^D$ and $\Sigma_f \in \mathbb{S}^D$ are the *system function* and *system noise* covariance, respectively. Likewise, $g \in \mathbb{R}^M \to \mathbb{R}^D$ and $\Sigma_g \in \mathbb{S}^M$ are the *observation function* and *observation noise* covariance, respectively. If we further restrict ourselves to linear systems we are left with the form:

$$\begin{aligned} \mathbf{x}_t &\sim \mathcal{N}(\mathbf{A}\mathbf{x}_{t-1},\boldsymbol{\Sigma}_f)\,, \\ \mathbf{y}_t &\sim \mathcal{N}(\mathbf{C}\mathbf{x}_t,\boldsymbol{\Sigma}_g)\,. \end{aligned} \tag{1.20}$$

Some draws from systems (1.17) and (1.18) are shown in Figure 1.8.

We summarize the models discussed here using the “probability of everything”

(POE).[1] If we summarize our prior on the model as $p(\theta)$, we summarize the autoregressive form as:

$$p(\mathbf{Y}, \theta) = p(\theta) \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{Y}_{(p)}, \theta) \, . \tag{1.21}$$

Where as in the state space form:

$$p(\mathbf{X}, \mathbf{Y}, \theta) = p(\theta) \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{x}_t, \theta) p(\mathbf{x}_t | \mathbf{x}_{t-1}, \theta) \, . \tag{1.22}$$

In the case of nonparametric models θ might be infinite dimensional and we cannot work with $p(\theta)$ explicitly, as mentioned in 1.2, requiring us to write the POE after integrating out θ. For the autoregressive model this is:

$$p(\mathbf{Y}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{Y}_{1:t-1}) \, , \tag{1.23}$$

which is merely the standard one-step-ahead predictive formulation of the marginal likelihood $p(\mathbf{Y})$, also known as the *evidence*. For the state space model:

$$p(\mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{X}_{1:t}, \mathbf{Y}_{1:t-1}) p(\mathbf{x}_t | \mathbf{X}_{1:t-1}) \, . \tag{1.24}$$

Both of these reductions are merely using the marginalization rules of probability.

We can convert between modeling data $\mathbf{y}_t$ in autoregressive or in state space form. Just as we integrate out θ from the state space form to get (1.24), we can further marginalize out $\mathbf{x}_t$ from a state space model leaving us with a predictor in autoregressive form (1.23), albeit with potentially infinite order. In practice, we may predict with the marginals of bounded order $p(\mathbf{y}_t | \mathbf{Y}_{(p)})$ instead of using all the information $p(\mathbf{y}_t | \mathbf{Y}_{1:t-1})$. We can approximate a state space model via a finite order autoregressive model. If we are using the negative log likelihood (NLL) as the loss measure, we quantify the loss in predictive accuracy from truncating the

[1] This comes from a quote of Steve Gull, “Always write down the probability of everything.”

model order using mutual information:

$$\begin{aligned}
&\mathrm{I}(\mathbf{y}_t; \mathbf{Y}_{1:t-p-1}|\mathbf{Y}_{(p)}) \\
&\overset{(A.43)}{=} \mathrm{H}[\mathbf{y}_t|\mathbf{Y}_{1:t-p-1}] - \mathrm{H}[\mathbf{y}_t|\mathbf{Y}_{1:t-1}] \qquad (1.25) \\
&\overset{(A.45)}{=} \mathbb{E}\left[\text{NLL under order } p \text{ model}\right] - \mathbb{E}\left[\text{NLL under state space model}\right] . \qquad (1.26)
\end{aligned}$$

This quantifies how much information is thrown away by discarding $\mathbf{Y}_{1:t-p-1}$ when making predictions.

Alternatively, we convert an autoregressive model into a state space form by "storing" the previous observations in the latent space. We use the following setup:

$$y_t = x_{t,1} \in \mathbb{R}\,, \quad x_{t,1} \sim p(\mathbf{x}_{t-1,1:p}|\theta)\,, \quad x_{t,i} = x_{t-1,i-1}\,, \; i \in 2{:}p\,, \qquad (1.27)$$

where p is the autoregressive predictor from (1.16) substituting $\mathbf{x}_{t-1,1:p}$ for $\mathbf{y}_{(p)}$. Therefore, a state space model with latent dimension $D = p \times E$ would be more generic than an autoregressive model of order p.

The distinction of state space models and autoregressive models provides a good first level splitting in the taxonomy of time series methods. In physical systems, prior knowledge often tempts the use of state space models. Knowledge of the equations of motion, for example, makes the representation of a state space model much more natural. State space models also have the advantage of not requiring the specification of an arbitrary order parameter. However, when inference becomes difficult the direct approach of autoregressive models can be advantageous.

1.3.3 Online Algorithms

Many applications demand *online* algorithms, meaning they are able to incorporate a new data point without repeating the entire training procedure. More generally an offline algorithm is known as a *batch* method.

In the state space community, online and retrospective inference tasks are known as filtering and smoothing, respectively. We consider the case of a vehicle tracking system to illustrate the distinction. When we are designing an autonomous navigation system; we would like to perform filtering. The system must decide where to steer the vehicle next based on its current location. The system must use the latest information from its sensors to detect oncoming obstacles. By contrast, a system on a Google street view[1] car is capable of smoothing. Street view must tag GPS locations to the street view photographs before being used. The street view system uses GPS and odometer measurements taken before and after the photo was taken. Street view has the luxury of using past and future GPS information in a retrospective context.

A subset of online methods is stream processing methods; these methods not only update to new points but do not have to keep the previously observed data in memory. These methods are memory efficient since they only need to keep *sufficient statistics* of the data.

Summary We have discussed the two different approaches to time series models: autoregressive and state space. In each framework we can use either linear or nonlinear models. Most of the classical methods are linear, but may struggle to fully account for the common features in time series discussed. Methodology in Chapter 3 covers nonlinear versions of autoregressive models and Chapter 4 covers nonlinear versions of state space models. Both chapters embrace a nonparametric Bayesian approach. These extensions help us deal with the frequent time series challenges, periodicity, change points, non-uniform sampling, etc., in a principled manner.

[1]http://maps.google.com/

Chapter 2

Data Sets

In this chapter we provide a preview of the data sets used in the experimental results, Section 1.3.1. This provides a context in which to view the methodological developments in the following chapters. We use six real data sets: Nile, well log, snowfall, bee dance, industry portfolios, and "fish killer." These data sets vary greatly in size and dimensionality. They each contain different aspects of time series mentioned in Section 5.5. We aim to best visualize each of the data sets to assist in noticing each of these time series features. We summarize the data sets used throughout this thesis in Table 2.1. Additionally, we provide the source of each data set in Table 2.2.

Name	T	domain	units	SF	US	SS	labels
Nile	663	$\mathbb{R}^+$	mm	1 year	Yes	N/A	No
Well Log	4,050	$\mathbb{R}^+$	-	-	Yes	N/A	No
Snowfall	13,880	$\mathbb{R}^+$	cm	1 day	Yes	N/A	No
Bee Dance	4,960	$\mathbb{R}^2 \times [0, 2\pi]$	px px rad	1/15 s	Yes	Yes	Yes
Portfolios	11,455	$(\mathbb{R}^+)^{30}$	USD	1 day	No	Yes	No
Fish Killer	45,175	$\mathbb{R}^+$	m	15 min	Yes	N/A	Yes

Table 2.1: Summary of data sets used. Sampling frequency is abbreviated as SF, while uniform sampling is abbreviated as US, and synchronized sampling as SS. The US dollar is abbreviated as USD. The bee data is made up of six sequences of length: 1,057, 1,124, 602, 756, 813, and 608. This gives a total of 4,960 frames. The x and y coordinates are measured in pixels (px) while the angle is radians (rad).

Name	Source
Nile	http://lib.stat.cmu.edu/S/beran http://mldata.org/repository/data/viewslug/nile-water-level/
Well Log	Schlumberger http://mldata.org/repository/data/viewslug/well-log/
Snowfall	http://www.climate.weatheroffice.ec.gc.ca/ Whistler Roundhouse station, identifier 1108906 http://mldata.org/repository/data/viewslug/whistler-daily-snowfall/
Bee Dance	http://www.cc.gatech.edu/~borg/ijcv_psslds/ http://mldata.org/repository/data/viewslug/bee/
Portfolios	http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/Data_Library/det_30_ind_port.html http://mldata.org/repository/data/viewslug/industry-portfolio/
Fish Killer	Aquatic Informatics http://mldata.org/repository/data/viewslug/fish_killer/

Table 2.2: Sources of data sets used. We summarize the original source of each data set used. We also provide the location of the data set on the http://mldata.org/ data set repository.

Nile data We consider the Nile data set [Beran, 1994], which has been used to evaluate many change point methods [Garnett et al., 2009; Whitcher et al., 1998]. The data set, shown in Figure 2.1, is a record of the lowest annual water levels on the Nile river during 622–1284 measured at the island of Roda, near Cairo, Egypt. Domain knowledge in geophysics suggests a change point in year 715 due to an upgrade in ancient sensor technology to the nilometer. Eltahir and Wang [1999] provides evidence that anomalies in the Nile record can be used to infer years of El Niño. The installation of the nilometer is the most visually noticeable change in the time series.

Bee waggle dance data Honey bees perform what is known as a waggle dance on honeycombs. The three stage dance is used to communicate with other honey bees about the location of pollen and water. Ethologists are interested in identifying the change point from one stage to another to further decode the signals bees send to one another. The bee data set contains six videos of sequences of bee waggle dances [Oh et al., 2008]. The video files have been preprocessed to extract the bee's position and head-angle at each frame, which is shown in Figure 2.2. While many in the literature have looked at the cosine and sine of the angle, we

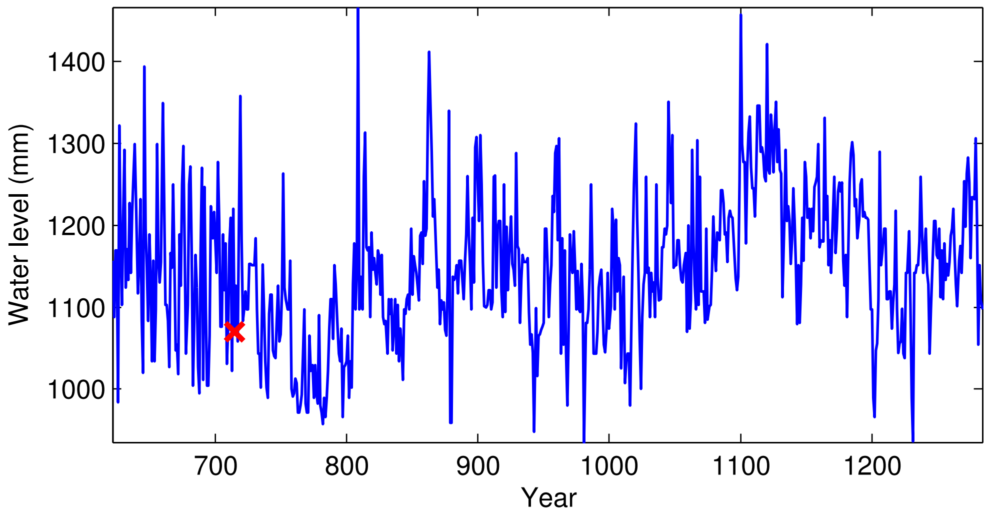


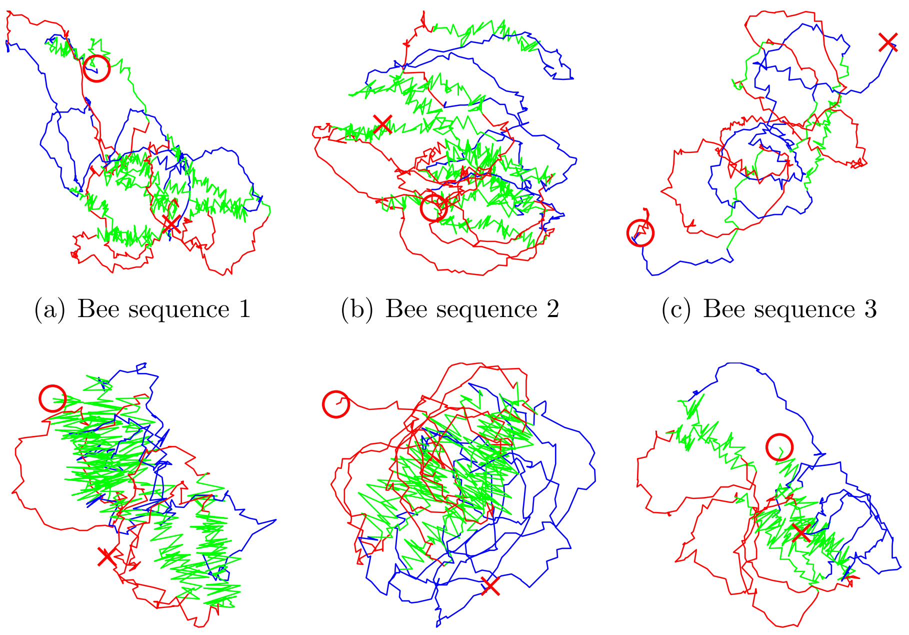
Figure 2.1: We show the Nile data (solid blue). The y-axis is the water level in mm, while the x-axis is the year. The red cross marks the installation of the new nilometer in 715.

chose to analyze angle *differences*.

Snowfall data We also used historical daily snowfall data in Whistler, BC, Canada over the period July 1 1972 to December 31 2009, shown in Figure 2.3. A probabilistic model of the next day snowfall is of great interest to local skiers. In this data set, being able to adapt to different noise levels is key: There may be highly volatile snowfall during a storm and then no snow in between storms.

Industry portfolio data We tried a multivariate data set: the "30 industry portfolios" data set, which was also used in the context of change point detection by Xuan and Murphy [2007]. The data consists of daily returns of 30 different industry specific portfolios from July 1 1963 to December 31 2008. The portfolios consist of NYSE, AMEX, and NASDAQ stocks from industries such as food, oil, telecoms, etc.

Fish killer data We used water level data from a dam in Richmond, BC, Canada [Osborne et al., 2011]. Dams typically include a fish weir mechanism for fish to bypass a dam. However, when the dam is not up to standard the water


(a) Bee sequence 1 (b) Bee sequence 2 (c) Bee sequence 3

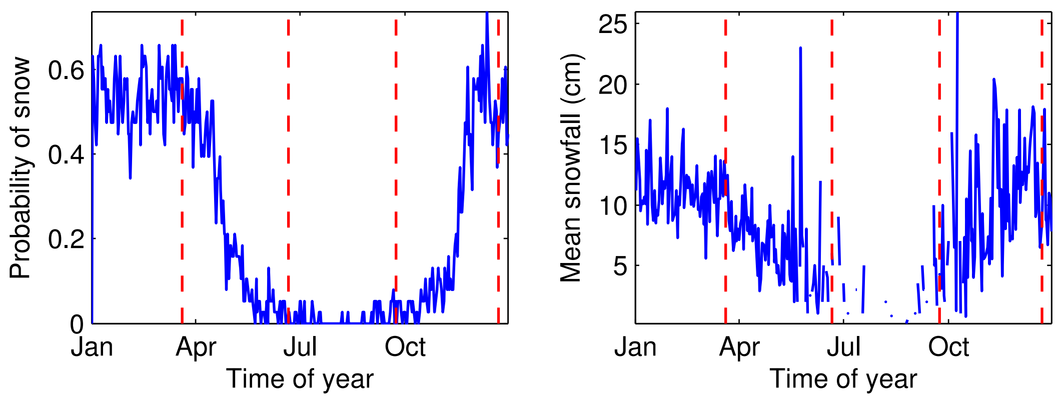
(d) Bee sequence 4 (e) Bee sequence 5 (f) Bee sequence 6

Figure 2.2: We illustrate bee dance sequences one through six. The plots correspond to the bee's trajectory during each video sequence. Left turns are shown in solid red, waggle in solid green, and right turn in solid blue. The bee's starting position is marked by a red circle (◯) while the end position is marked with a red cross ($\times$).

level oscillates quickly in a particular pattern, shown in Figure 2.4. When this happens the fish get stuck behind the dam and die. Water level data is used by environmental authorities to notify a dam owner when there is a risk of a mass loss of fish. An automated alert system would make these operations more efficient.

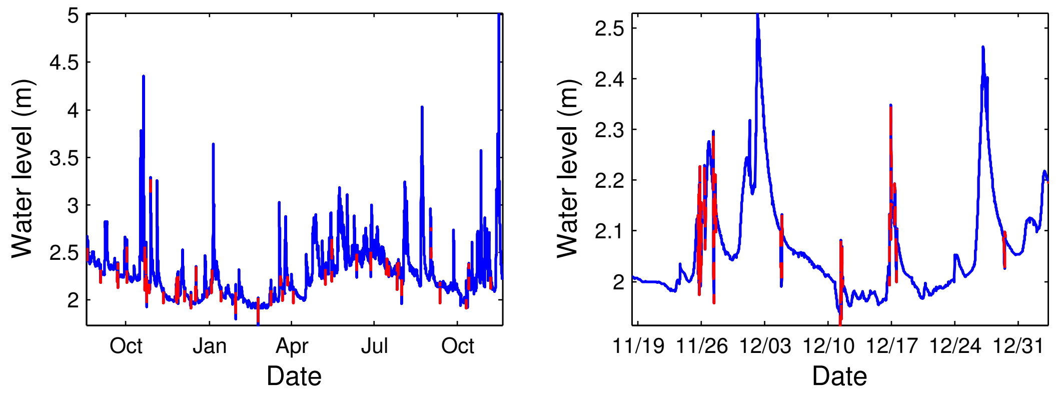
Since water level is positive, we worked with log water level. The beginning and end of every water oscillation (fish kills) are treated as a change point for the purposes of supervision.

Well log data We applied our method to the well log data set for detecting changes in rock stratification, described in Adams and MacKay [2007] and originally used in Ó Ruanaidh et al. [1994]. Shown in Figure 2.5, the time series consists of 4050 nuclear magnetic resonance measurements taken from the drill



(a) Probability of snow (b) Mean snowfall

Figure 2.3: We summarize the Whistler snowfall data set. On the left we show the empirical probability of snowfall, $P(\text{snow} > 0)$, for each day of the year over 1972–2009. Likewise, on the right we show the empirical mean snowfall conditional there is snow at all, $\mathbb{E}\left[\text{snowfall}|\text{snow} > 0\right]$, for each day of the year. We are not plotting the data for February 29 (on leap year). The estimated means are erratic in late spring and early autumn since the number of days with nonzero snowfall is small. Season boundaries are shown by the dashed vertical lines.



(a) Water level (b) Water level closeup

Figure 2.4: We plot the fish killer data. On the left, we show the water level behind the dam (solid blue) in m for the year long period. The areas of abnormal behavior are shown in dashed red. On the right, we show a portion of the time series in a closeup.

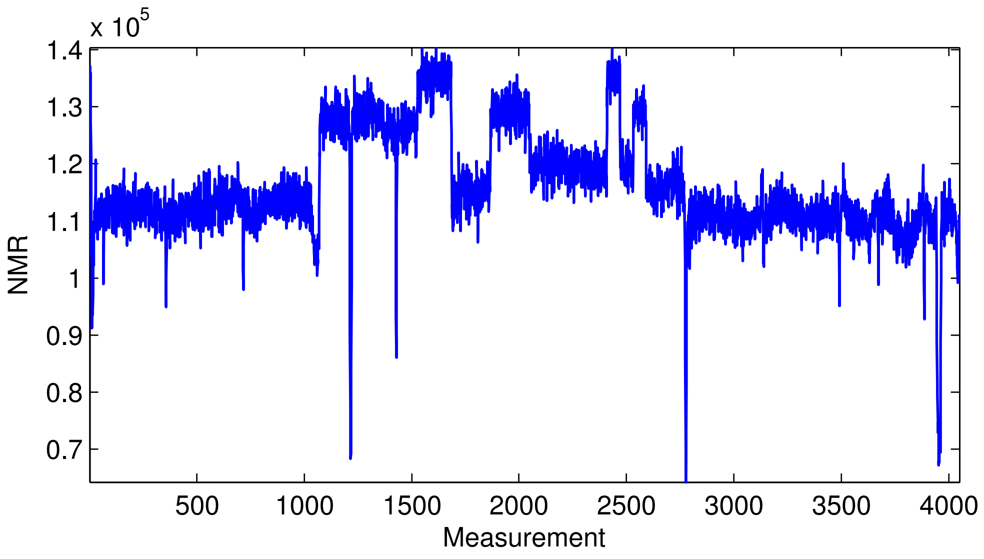


Figure 2.5: We show the well log data (solid blue). The y-axis is the NMR response.

while drilling a well. The method is commonly used to detect changes in the type of rock being drilled. Well log interpretation is an important area of geostatistics [Serra, 1984]. Changes in the mean are from changes in rock stratification; changes in the noise variance are also observed.

Chapter 3

Gaussian Process Overview

We provide a brief summary of Gaussian processes in this chapter. It therefore makes a rather compressed introduction to the topic. A more thorough introduction is available in [Rasmussen and Williams, 2006]. We go into detail in certain derivations that are particularly useful in later chapters, such as the output scale uncertainty in Section 3.1. We explain two different approaches to Gaussian process time series modeling and go into the details of efficient implementation, in Sections 3.4 and 3.6. Although the methods in these sections are useful in their own right, we further generalize them in Chapters 4 and 5. The Gaussian process approaches to time series in this chapter automatically handle some, but not all, of the challenges mentioned in Section 1.3.1. We show how many standard modeling approaches can be derived as special cases of Gaussian process time series models in Section 3.4.3. Finally, we discuss efficient rank-1 based methods to efficiently do GP inference over different windows of the data in Section 3.5.

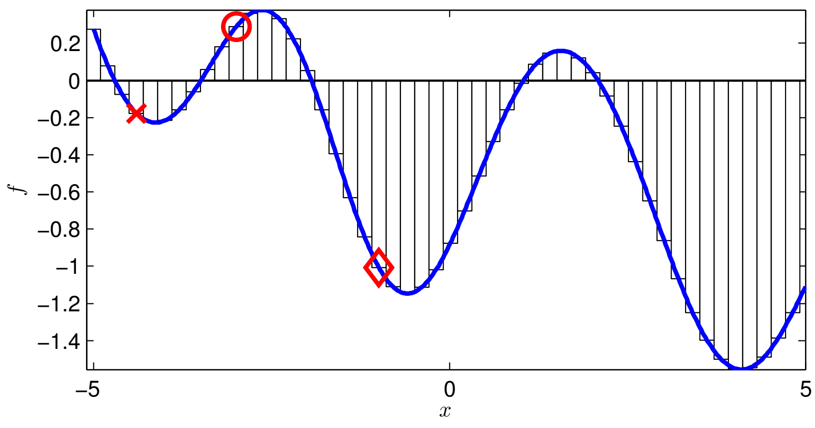
When performing a regression task we assume there exists some optimal prediction function $f \in \mathcal{X} \to \mathcal{Y}$, possibly with a noise distribution. In linear regression we may assume that the outputs y are a linear function of the inputs $\mathbf{x}$, as in (1.3), with some parameters θ, often much smaller than the number of training examples N: $|\theta| \ll N$. However, for many real world data sets a simple parametric form, such as a linear form, is an unrealistic assumption. Therefore, we would like models that can learn general functions f. Since the functions are not summarizable by a small (fixed) number of parameters θ, maximum likelihood estimation of the parameters causes severe overfitting. In fact, in a Gaussian process the effective number of parameters is often infinite. Therefore, in order to do inference and make predictions in a probabilistic framework, we must put a prior probability distribution on functions. We make predictions using our posterior on an underlying predictive function f given a set of training examples in the form of input-output pairs: $(\mathbf{x}, y)$.

A Gaussian process (GP) is a nonparametric method as it places probability distribution over functions. While most probability distributions are over finite dimensional objects, such as scalars $\mathbb{R}$ or vectors $\mathbb{R}^D$, GPs are over infinite dimensional objects f, functions (typically, $\mathbb{R}^D \rightarrow \mathbb{R}$).[1] Therefore, GPs are a fundamental tool in Bayesian nonparametrics along with the Dirichlet process (DPs), which is a distribution on distributions p (typically, $\mathbf{M}(\mathbb{R}^D)$). Although called GPs in machine learning, they were historically called *kriging* in the (geo)statistics community [Matheron, 1973]. GPs are only a distribution on functions f, not on the inputs $\mathbf{x}$. This means GPs are discriminative, knowledge of the distribution on $\mathbf{x}$ does not help us predict y. We cannot learn anything from training examples where y is missing.

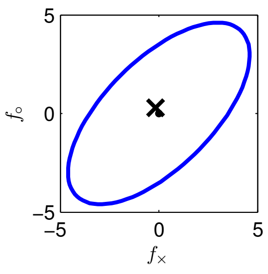
Conceptually, a function $f \in \mathbb{R} \rightarrow \mathbb{R}$ can be thought of as an (uncountably) infinitely long vector;[2] imagine discretizing a function into a lookup table with infinitely fine bins. That lookup table is stored in a vector $\mathbf{f}$. A GP is merely a multivariate normal distribution on the vector $\mathbf{f}$, as illustrated in Figure 3.1. The marginalization property of Gaussians (A.32) implies that we only need to consider the covariance between the points in the function we have observations of $\mathcal{D} = \{(\mathbf{x}_i \in \mathbb{R}^D, y_i \in \mathbb{R})\}_{i=1}^N$, where $y_i = f(\mathbf{x}_i)$ with possibly some Gaussian observation noise. Therefore, we only explicitly need to work with a probability distribution on finite dimensional object $\mathbf{y} := [y_1, \ldots, y_N]$ while a GP implicitly places a distribution on an infinite dimensional object f. While practitioners accept this trick as "kosher," mathematicians require more rigorous analysis to prove that GPs "exist" [Orbanz, 2009].

[1] GPs are applicable to much more general inputs $\mathbf{x}$ than $\mathbb{R}^D$, such as graphs and strings, but we assume $\mathbf{x} \in \mathbb{R}^D$ for simplicity.

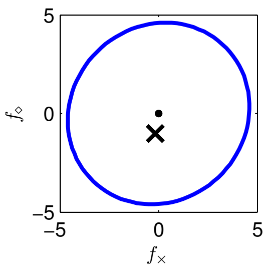
[2] If the function is smooth, as it often is in GP setups, the function can be discretized into a countably infinite vector. All smooth functions are summarized by their values at the rationals, meaning $f \in \mathbb{Q} \rightarrow \mathbb{R}$.



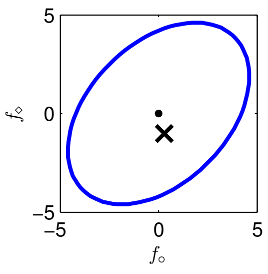
(a) A function f discretized into vector $\mathbf{f}$



(b) Joint on $(\times, \bigcirc)$



(c) Joint on $(\times, \diamond)$



(d) Joint on $(\bigcirc, \diamond)$

Figure 3.1: On the top row we show a synthetic function $f \in \mathbb{R} \to \mathbb{R}$ (solid blue) sampled from GP that has been discretized to form a vector $\mathbf{f}$. The entries in $\mathbf{f} \in \mathbb{R}^N$ correspond to the height of the bins. We mark three of the vector entries with $\times$, $\bigcirc$, and $\diamond$. On the bottom row we show the joint distribution between every combination of the value of $\mathbf{f}$ at these three points. The solid blue ellipse represents the equiprobability lines representing 90% of the distributions mass. The black dot shows the mean of the distribution while the black crosses show the value of the two respective sampled elements of $\mathbf{f}$. Note that the distributions are all Gaussian. For any combination of entries in $\mathbf{f}$, the joint will be Gaussian.

Prior specification A GP is specified by a mean function $\mu(\cdot) \in \mathbb{R}^D \to \mathbb{R}$ and a (positive-definite) covariance function, also called a *kernel*, $k_\xi(\cdot,\cdot) \in \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ with hyper-parameters ξ. The covariance function hyper-parameters ξ describe general properties of the functions generated from the GP, such as smoothness, input scale and output scale. A GP implies the following distribution on the data $\mathbf{y}$:

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}), \quad \mu_i := \mu(\mathbf{x}_i), \quad \mathbf{K}(i,j) := k_\xi(\mathbf{x}_i, \mathbf{x}_j) \tag{3.1}$$

$$\Leftrightarrow \mathbf{y} \sim \mathcal{GP}(\mathbf{X}|\mu, k), \quad \mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_N]. \tag{3.2}$$

The matrix $\mathbf{K} \in \mathbb{S}^N$ is known as the covariance, or Gram, matrix, whose entries $\mathbf{K}(i,j)$ are often thought of as the "similarity" between inputs $\mathbf{x}_i$ and $\mathbf{x}_j$. We use the notation $\mathcal{GP}$ to denote when a set of outputs $\mathbf{y}$ is drawn from a GP at inputs $\mathbf{x}$; we also use the notation to say a function f is drawn from a GP:

$$f \sim \mathcal{GP}(\mu, k). \tag{3.3}$$

The prior mean function is typically set to zero: $\mu(\mathbf{x}) = 0$. We summarize this setup with the graphical model in Figure 3.2.

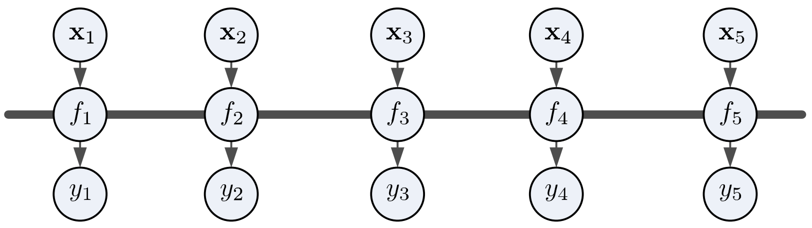


Figure 3.2: Graphical model for a Gaussian process. The function values f are fully connected, as indicated by the bold bar, and are dependent on $\mathbf{x}$. The observations y are f plus some observation noise. We caution the reader that the bold bar is a non-standard notation in the graphical models community.

The most common covariance function is the squared-exponential (SE),[1]

$$k_\xi(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)\right) \tag{3.4}$$

$$= \sigma_0^2 \exp\left(-\frac{1}{2}\|\mathbf{L}^\top(\mathbf{x}_i - \mathbf{x}_j)\|_2^2\right), \quad \mathbf{L} := \mathrm{chol}(\mathbf{M})^\top \in \mathbb{L}^D, \tag{3.5}$$

which generates smooth functions that are infinitely differentiable. Typically, we make $\mathbf{M}$ axis aligned for *automatic relevance determination* (ARD) [Neal, 1998]:

$$\mathbf{M} := \mathrm{diag}(\boldsymbol{\ell}^{-2}), \quad \xi := \{\sigma_0^2 \in \mathbb{R}^+, \boldsymbol{\ell} \in (\mathbb{R}^+)^D\}. \tag{3.6}$$

The SE-ARD covariance has $|\xi| = D + 1$ hyper-parameters. The ARD kernels can control the degree of spatial correlation differently in each input dimension. Therefore, when learning the hyper-parameters the kernel can automatically reduce the influence of irrelevant inputs. Similarly, there is the Laplace covariance:

$$k_\xi(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 \exp\left(-\|\mathbf{L}^\top(\mathbf{x}_i - \mathbf{x}_j)\|_1\right), \tag{3.7}$$

which generates rough functions that are not differentiable. The $\mathbf{M}$ matrix in the SE case is analogous to $\mathbf{L}\mathbf{L}^\top$ case in the Laplace covariance. Due to the spherical symmetries of the L_2 norm we can use any matrix square root of $\mathbf{M}$ to get $\mathbf{L}$ for the SE covariance [Ng, 2004]. Since those symmetries do not exist in the case of the L_1 norm we define the Laplace covariance in terms of $\mathbf{L}$ rather than $\mathbf{M}$. In an ARD Laplace:

$$\mathbf{L} := \mathrm{diag}(\boldsymbol{\ell}^{-1}), \quad \xi := \{\sigma_0^2, \boldsymbol{\ell}\}. \tag{3.8}$$

Gaussian observation noise of variance $\sigma_n^2 \in \mathbb{R}^+$ is accounted for by adding σ_n^2 along the diagonal of $\mathbf{K}$.

The SE and Laplace covariance functions are said to be *stationary* because they only depend on the difference between points $\mathbf{x}_i - \mathbf{x}_j$. A covariance is *isotropic*, invariant to direction, if it only depends on the magnitude of the dif-

[1] Although in wide use in the community, the term squared-exponential is a misnomer. It could more accurately be called an exponentiated-quadratic.

ference $\|\mathbf{x}_i - \mathbf{x}_j\|_2$. Consequently, isotropic covariance functions $\subset$ stationary covariance functions $\subset$ positive definite (all) covariance functions. A survey of several covariance functions is done in Rasmussen and Williams [2006, Ch. 4].

Many mathematical operations preserve positive definiteness and can therefore be used for combining "old kernels" to make "new kernels." The two most common of these operations are the summation and product: The sum of any two valid (positive-definite) covariance functions is also a valid covariance function; the product of any valid two covariance functions is a valid covariance. Sampling a function from a GP with the sum of two kernels is equivalent to sampling functions from each kernel independently and adding them. For brevity we now drop the hyper-parameters ξ from the kernel function $k_\xi(\cdot,\cdot)$ and write $k(\cdot,\cdot)$.

Example The constant covariance $k'(\mathbf{x}_i, \mathbf{x}_j) = c \in \mathbb{R}^+$ can be added in place of a constant mean function. The hyper-parameter c acts as a prior variance on a constant mean function $\mu \in \mathbb{R}$. We also place a hyper-mean on the prior on the mean function $\mu_0 \in \mathbb{R}$. To be precise,

$$\mathbf{y} \sim \mathcal{GP}(\mathbf{X}|\mu_0, k + k') \quad \Leftrightarrow \quad \mathbf{y} \sim \mathcal{GP}(\mathbf{X}|\mu, k)\,, \quad \mu \sim \mathcal{N}(\mu_0, c)\,. \tag{3.9}$$

This gives us increased flexibility. Since the GP often mean reverts when extrapolating, the zero mean assumption is often consequential. The constant covariance allows us to include uncertainty in the mean function without necessitating separate computations for the posterior mean function.

Example Another meta-covariance function is composition: We can project the input $\mathbf{x}$ to any feature space $\phi(\mathbf{x})$ and apply the kernel in that space and get a valid kernel. In concrete terms,

$$k'(\mathbf{x}, \mathbf{x}') = k(\phi(\mathbf{x}), \phi(\mathbf{x}'))\,. \tag{3.10}$$

MacKay [1998] constructed the periodic covariance function by projecting the input to the unit circle, $\phi(\mathbf{x}) = \begin{bmatrix}\cos(\mathbf{x}) & \sin(\mathbf{x})\end{bmatrix}^\top$, and then applying the squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \exp\left(-\frac{(\cos(\mathbf{x}) - \cos(\mathbf{x}'))^2 + (\sin(\mathbf{x}) - \sin(\mathbf{x}'))^2}{2\boldsymbol{\ell}^2}\right) \tag{3.11}$$

$$= \sigma_0^2 \exp\left(-\frac{2\sin^2(\frac{\mathbf{x}-\mathbf{x}'}{2})}{\boldsymbol{\ell}^2}\right) . \tag{3.12}$$

Synthetic functions f sampled from the periodic kernel will be exactly periodic with period 2π, the functions will *not* be sinusoids with probability one. In practice, we often add a square exponential term, which means the sampled function f will be the sum of a periodic function and an SE function. We can also do a covariance product with a squared exponential, which allows the periodicity to decay and the periodic component to change slowly. When using the periodic covariance function, there are pitfalls in learning the hyper-parameters $\xi = \{\sigma_0, \boldsymbol{\ell}\}$. If the periodic component is only present for a portion of the time series, then the type-II MLE estimate of the amplitude of the periodic component σ_0 will be near zero. Using the product of the periodic and squared exponential alleviates this problem.

We graphically demonstrate the differences between different kernels in Figure 3.3. We give examples of the priors and posteriors, for a set of points, for three kernels. In addition to the point-wise mean and error bars we plot samples from the distribution, as the mean function is unrepresentative of the whole distribution. We also plot the CDF on how large x must be before the function passes a particular threshold. We do this to illustrate that GPs are flexible enough to compute various quantities of interest about a function, not merely a predictive mean and variance.

Posterior calculation We now explain how to calculate posterior distributions for pointwise prediction. Using the standard conditioning rules for a Gaussian distribution, (A.32), and (3.1) we get the predictive distribution on a new observation $y_\star$ at test input $\mathbf{x}_\star$:

$$\begin{bmatrix} \mathbf{y} \\ y_\star \end{bmatrix} \overset{(3.1)}{=} \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_\star \\ \mathbf{K}_\star^\top & \mathbf{K}_{\star\star} \end{bmatrix}\right) , \tag{3.13}$$

implying

$$
\begin{aligned}
p(y_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu_\star, \sigma_\star^2)\,, && (3.14)\\
\mu_\star \overset{(A.34)}{=} \mathbf{K}_\star^\top \mathbf{K}^{-1}\mathbf{y} \in \mathbb{R}\,, && (3.15)\\
\sigma_\star^2 \overset{(A.34)}{=} \mathbf{K}_{\star\star} - \mathbf{K}_\star^\top \mathbf{K}^{-1}\mathbf{K}_\star \in \mathbb{R}^+\,. && (3.16)
\end{aligned}
$$

Here $\mathbf{K}_\star := k(\mathbf{X}, \mathbf{x}_\star) \in \mathbb{R}^{N\times 1}$ is the cross-covariance between the test input $\mathbf{x}_\star$ and the training inputs $\mathbf{X}$. Likewise, $\mathbf{K}_{\star\star} = k(\mathbf{x}_\star, \mathbf{x}_\star) \in \mathbb{R}^+$ is the prior variance of the test point. For numerical reasons, we always avoid explicitly working with the inverse $\mathbf{K}^{-1}$ and use matrix back substitution methods instead.[1] If the test inputs are not known at training time, the following form is often used:

$$
\begin{aligned}
\mathbf{L} := \mathrm{chol}(\mathbf{K})^\top \in \mathbb{L}^N\,, \quad \boldsymbol{\beta} := \mathbf{L}^\top \backslash (\mathbf{L}\backslash \mathbf{y}) \in \mathbb{R}^N\,, && (3.17)\\
\mu_\star \overset{(3.15)}{=} \mathbf{K}_\star^\top \boldsymbol{\beta}\,, && (3.18)\\
\sigma_\star^2 \overset{(3.16)}{=} \mathbf{K}_{\star\star} - (\mathbf{L}\backslash \mathbf{K}_\star)^\top (\mathbf{L}\backslash \mathbf{K}_\star)\,. && (3.19)
\end{aligned}
$$

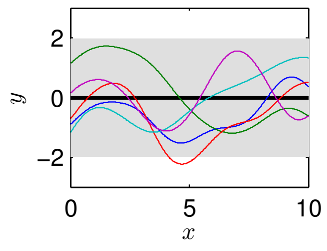
This, like the preceding form, requires $\mathcal{O}(N^3)$ computation in training, but only $\mathcal{O}(N)$ for the predictive means in test and $\mathcal{O}(N^2)$ for the predictive variances. The computational savings do not come from the superiority of back substitution over matrix inversion but rather the precomputation of $\boldsymbol{\beta}$ during training.

3.1 Uncertain Output Scale

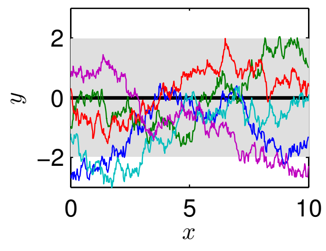
We illustrate the relevant computations for output scale uncertainty in this section. This same setup is referred to as the "t-process" in Rasmussen and Williams [2006, Sec. 9.9]. We review the derivation of the t-process and then introduce an improvement that allows it to be used in a streaming manner. We also use an example to argue that it is preferable over a standard GP in many cases.

GP hyper-parameters ξ are tractably learnable by maximizing the marginal

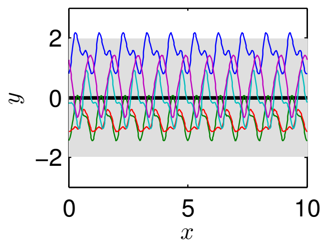
[1] We use the notation $\mathbf{A}\backslash\mathbf{y}$ equivalently to $\mathbf{A}^{-1}\mathbf{y}$ except that we expect the computation to be done using matrix back substitution rather than explicit calculation of the inverse and then multiplying. We use the notation $\mathbf{A} = \mathrm{chol}(\mathbf{B})$ to state that $\mathbf{A}$ is the (upper triangular) Cholesky factorization of $\mathbf{B}$ such that $\mathbf{B} = \mathbf{A}^\top\mathbf{A}$.



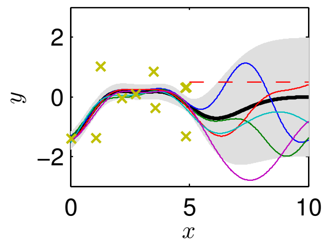
(a) SE prior



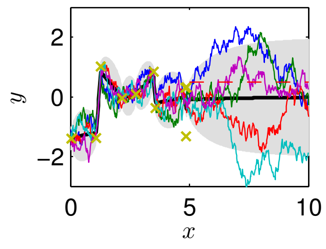
(b) Laplace prior



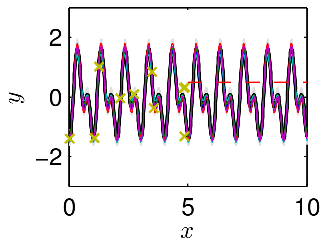
(c) Periodic prior



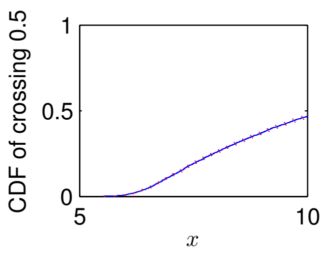
(d) SE posterior



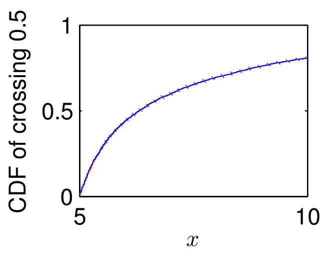
(e) Laplace posterior



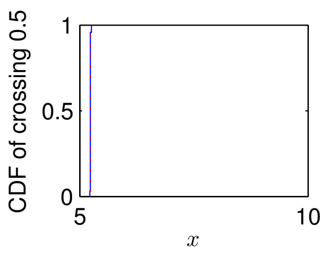
(f) Periodic posterior



(g) SE posterior CDF



(h) Laplace posterior CDF



(i) Periodic posterior CDF

Figure 3.3: The top row shows sample functions f from GP priors using either the SE (left), Laplace (middle), or periodic (right) covariance functions. The gray area represents the 95% error bars and the black line is the mean function of the GP. The samples are the other (colored) lines. On the second row we sample from the respective posteriors after observing f at $N = 10$ points shown with blue crosses ($\times$). On the bottom row we show the CDF (solid blue) for the time until the function f crosses 0.5 for the first time after the input $x = 5$ (the dashed red line in the second row). Since the CDF is estimated by Monte Carlo samples the error bars on the CDF are shown in dashed red. Since the periodic covariance function has essentially "nailed down" the function, it is highly confident where the threshold is crossed as seen in the CDF plot.

likelihood $p(\mathbf{y}|\mathbf{X}, \xi)$. This is sometimes called type-II maximum likelihood. In a full Bayesian treatment, we should integrate out the hyper-parameters. Unfortunately, this cannot be done analytically in general, e.g. for the input scale. Monte Carlo methods are usually used to estimate these integrals [Titsias et al., 2010].

However, the output scale, with a gamma[1] prior, can be integrated out analytically and adds some increased generality for a small computational penalty. Consider the following setup using shape parameter α_0 and inverse scale β_0 for the gamma prior on τ,

$$\mathbf{y} \sim \mathcal{GP}(\mathbf{X}|\mu_0, k')\,, \quad k'(\mathbf{x}_i, \mathbf{x}_j) := k(\mathbf{x}_i, \mathbf{x}_j)/\tau\,, \quad \tau \sim \text{Gamma}(\alpha_0, \beta_0)\,. \tag{3.20}$$

Here we apply an unknown scaling factor $\tau \in \mathbb{R}^+$ to the covariance function k to get a new covariance function k'. Note that without loss of generality we can set $\beta_0 = 1$ if we can control the scale of k by the covariance hyper-parameters ξ. This corresponds to placing a gamma prior on the scale of variation in $\mathbf{y}$. The posterior on τ after N observations of $\mathbf{y}$ is:

$$\begin{aligned} p(\tau|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\tau, \mathbf{X})p(\tau) &= \mathcal{N}(\mathbf{y}|0, \mathbf{K}/\tau)\text{Gamma}(\tau|\alpha_0, \beta_0) && (3.21)\\ &\propto |\mathbf{K}/\tau|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathbf{y}^\top(\mathbf{K}/\tau)^{-1}\mathbf{y}\right) \tau^{\alpha_0-1}\exp(-\beta_0\tau) && (3.22)\\ &\propto \tau^{\alpha_0+\frac{N}{2}-1} \exp\left(-\beta_0\tau - \frac{1}{2}\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}\tau\right)\,, && (3.23) \end{aligned}$$

which is the standard form of gamma,

$$\begin{aligned} p(\tau|\mathbf{X}, \mathbf{y}) &= \text{Gamma}(\alpha_N, \beta_N)\,, && (3.24)\\ \alpha_N = \alpha_0 + \frac{N}{2}\,, &\quad \beta_N = \beta_0 + \frac{1}{2}\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}\,. && (3.25) \end{aligned}$$

This allows us to find the posterior on the output scale analytically.

[1] Roughly five different parameterizations of the gamma distribution exist. We use the shape, $\alpha \in \mathbb{R}^+$, and inverse scale, $\beta \in \mathbb{R}^+$, parameterization. Meaning that if $\tau \sim \text{Gamma}(\alpha, \beta)$ then $p(\tau) = (\beta^\alpha/\Gamma(\alpha))\tau^{\alpha-1}\exp(-\beta\tau)$ and $\mathbb{E}[\tau] = \alpha/\beta$. This parameterization is the most convenient for Bayesian updating.

Streaming implementation We improve upon (3.25) by computing the posterior in a streaming manner. The term $\mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y}$ is referred to as the standardized-square-error (SSE) of the GP because it is equal to the sum of the (standardized) square error from the one-step-ahead predictives:

$$\text{SSE} := \sum_{i=1}^{N} \frac{(y_i - \overbrace{\mathbb{E}\left[y_i|\mathbf{y}_{1:i-1}, \mathbf{x}_{1:i}\right]}^{=:\widehat{y}_i})^2}{\underbrace{\text{Var}\left[y_i|\mathbf{y}_{1:i-1}, \mathbf{x}_{1:i}\right]}_{=:\sigma_i^2}} \tag{3.26}$$

$$= \left(\frac{\mathbf{y} - \widehat{\mathbf{y}}}{\boldsymbol{\sigma}}\right)^\top \left(\frac{\mathbf{y} - \widehat{\mathbf{y}}}{\boldsymbol{\sigma}}\right) \tag{3.27}$$

$$= (\mathbf{L}\backslash\mathbf{y})^\top (\mathbf{L}\backslash\mathbf{y}) = \mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y}\,. \tag{3.28}$$

Showing that $\mathbf{L}\backslash\mathbf{y}$ gives the standardized one-step-ahead errors requires rank-1 update equations, which is covered in Section 3.6. Like the likelihood, this relation holds for any permutation of the data points. The SSE formulation allows us to update the posterior on τ online with $\mathcal{O}(1)$ additional computation. We note that there is an active research area in efficient implementations of online Gaussian processes [Csató and Opper, 2002].

We now also efficiently compute the evidence and posterior predictive. The marginal likelihood of the data is found using Bayes' rule,

$$p(\mathbf{y}|\mathbf{X}) = \frac{p(\mathbf{y}|\tau, \mathbf{X})p(\tau)}{p(\tau|\mathbf{X}, \mathbf{y})} = \frac{\mathcal{N}(\mathbf{y}|0, \mathbf{K}/\tau)\text{Gamma}(\tau|\alpha_0, \beta_0)}{\text{Gamma}(\tau|\alpha_N, \beta_N)}\,. \tag{3.29}$$

The non-normalization part of the gamma distribution in the denominator cancel out with the numerator yielding a Student's t, denoted $\text{St}(\cdot)$, marginal distribution:

$$p(\mathbf{y}|\mathbf{X}) \propto \Gamma(\alpha_N)\beta_N^{-\alpha_N} \propto \left(\beta_0 + \frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y}\right)^{-\left(\alpha_0 + \frac{N}{2}\right)} \tag{3.30}$$

$$\propto \left(1 + \frac{1}{2\beta_0}\mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y}\right)^{-\left(\alpha_0 + \frac{N}{2}\right)}, \tag{3.31}$$

which we can write as

$$\left(1 + \frac{1}{2\alpha_0}\mathbf{y}^\top \left(\frac{\beta_0}{\alpha_0}\mathbf{K}\right)^{-1} \mathbf{y}\right)^{-\frac{(2\alpha_0+N)}{2}} \tag{3.32}$$

$$\propto \mathrm{St}_{2\alpha_0}\left(\mathbf{0}, \frac{\beta_0}{\alpha_0}\mathbf{K}\right) . \tag{3.33}$$

The proportionality relations are with respect to $\mathbf{y}$ when computing $p(\mathbf{y}|\mathbf{X})$. Similar to (3.15) we get the following posterior predictive equations:

$$p(y_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \int p(y_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{y}, \tau)p(\tau|\mathbf{X}, \mathbf{y})d\tau \tag{3.34}$$

$$= \int \mathcal{N}(y_\star|\mu_\star, \sigma_\star^2/\tau)\mathrm{Gamma}(\tau|\alpha_N, \beta_N)d\tau \,, \tag{3.35}$$

$$\mu_\star \overset{(A.34)}{=} \mathbf{K}_\star^\top \mathbf{K}^{-1}\mathbf{y} \,, \tag{3.36}$$

$$\sigma_\star^2 \overset{(A.34)}{=} \mathbf{K}_{\star\star} - \mathbf{K}_\star^\top \mathbf{K}^{-1}\mathbf{K}_\star \,. \tag{3.37}$$

Equation (3.35) is the same as the marginal likelihood (3.33) if we had scale prior $\mathrm{Gamma}(\tau|\alpha_N, \beta_N)$ and likelihood $\mathcal{N}(y_\star|\mu_\star, \sigma_\star^2/\tau)$. Therefore, by (3.33)

$$p(y_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \mathrm{St}_{2\alpha_N}\left(\mu_\star, \frac{\beta_N}{\alpha_N}\sigma_\star^2\right) . \tag{3.38}$$

We make some remarks about the uncertain output scale setup. Firstly, note that the predictive mean $\mu_\star$ is not directly affected by integrating out the scale. Scale integration only relieves a degree of over-confidence that results when fitting the scale via maximum likelihood. Therefore, integrating out the output scale may affect the length scale found when optimizing the evidence and thereby indirectly affect the predictive mean. Secondly, note that the uncertain output scale affects the entire covariance function k, which includes terms for the noise and constant covariance. Therefore, a larger output scale also implies a larger amount of noise. Output scale uncertainty implies an unknown scale of the underlying function f but also a fixed signal-to-noise ratio. Nonetheless, output scale uncertainty still increases the generality of a GP and merely changing the evidence from a Gaussian to Student's t represents a more reasonable and robust distribution on the data. If outliers are pervasive throughout a data set it is most appropriate to use a Student's t observation noise model, which requires sampling methods [Neal, 1997] or approximate inference [Kuss, 2006; Vanhatalo et al., 2009].

We have provided a method to integrate out one hyper-parameter of the covariance function k, the overall scale, without incurring any significant computational cost. We would often like to integrate out additional hyper-parameters. However, this cannot be done without approximate methods or a significant increase in computational complexity. Additionally, we would often like to include some heavy tail noise on the observations for increased robustness, but this requires approximate methods. Uncertain output scale may provide some level of additional robustness while allowing for exact inference.

3.2 Uncertain Inputs

We may be in a situation where there is uncertainty in location of the test inputs $\mathbf{x}_\star$ to the GP. In this section, we cover Gaussian process prediction when the test input $\mathbf{x}_\star$ has a Gaussian distribution. Due to the complex interaction of the inputs in the kernel function k, we cannot analytically do GP inference when multiple inputs, i.e. the training set, are uncertain. However, when only the test input is uncertain, and has a Gaussian distribution, exact predictive moments from a GP can be calculated analytically. We focus on the case of an SE-ARD kernel k, where inference with uncertain inputs is easiest. The derivations in this section are based on the results in Deisenroth [2009]; Girard et al. [2003]; Kuss [2006]; Quiñonero-Candela et al. [2003]. We review these results but provide a more concise derivation.

Suppose we would like to predict a function value $f(\mathbf{x}_\star)$, $f \in \mathbb{R}^D \to \mathbb{R}$, for an *uncertain* test input $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $f \sim \mathcal{GP}$ with an SE-ARD kernel k.[1] A naive observer might assume that a Gaussian process with a Gaussian input density would result in a Gaussian predictive density. We graphically illustrate why this is not the case in Figure 3.4. If a Gaussian input $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ density

[1] Throughout this section we say certain variables are in $\mathbb{R}^+$, however if kernels other than SE-ARD are used, such as trigonometric kernels, these variables might take on negative values.

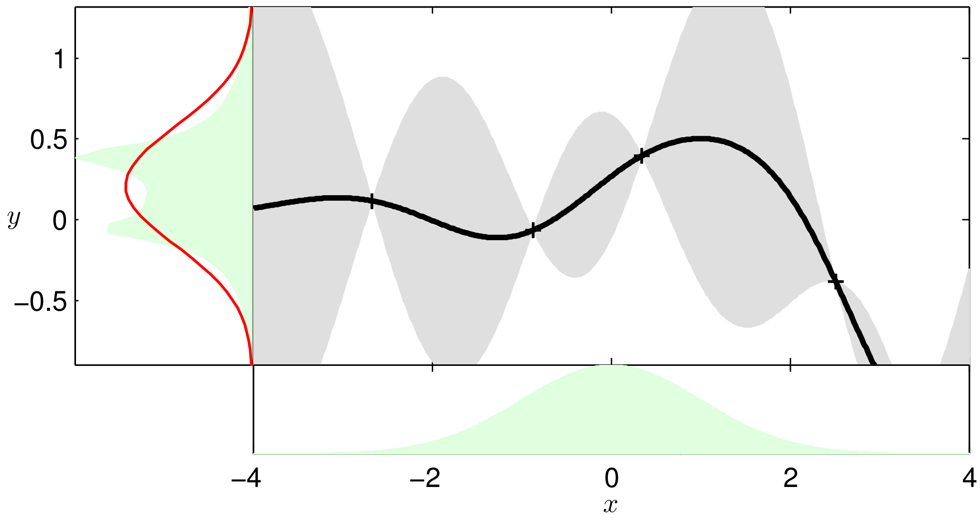


Figure 3.4: GP prediction at an uncertain test input. The input distribution $p(\mathbf{x}_\star)$ is the green Gaussian sitting on the bottom of the right panel. This panel also shows the mean function (black) and the 95% error bars (shaded) based on the training data points (black pluses +). To determine the expected function value, we average over both the input distribution (green, sitting on the bottom of the right panel) and the function distribution (GP model). The shaded distribution represents the exact distribution over function values. The exact predictive distribution (green shaded, left panel) is approximated by a Gaussian (red) that possesses the mean and the covariance of the exact predictive distribution (known as moment matching).

is propagated through a nonlinear function, the predictive distribution

$$p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \xi) = \int p(f(\mathbf{x}_\star)|\mathbf{x}_\star, \xi)p(\mathbf{x}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{x}_\star \tag{3.39}$$

will not be Gaussian, and often not even unimodal. Fortunately, we can approximate the predictive distribution $p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \xi)$ (we now drop the GP hyperparameters ξ for brevity) with a Gaussian (red in left panel of Figure 3.4) that has the exact same mean and variance as the true predictive distribution (moment matching). This is the optimal approximation to the true density in the Kullback-Leibler (KL) sense [Herbrich, 2005],

$$(\boldsymbol{\mu}_*, \sigma_*^2) = \operatorname*{argmin}_{\mathbf{m},\mathbf{s}} \mathrm{KL}(p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(f(\mathbf{x}_\star)|\mathbf{m}, \mathbf{s}))\,. \tag{3.40}$$

In order to determine the moments, $\boldsymbol{\mu}_*$ and σ_*^2, we must average over the GP posterior on the function f and the input density $p(\mathbf{x}_\star)$.

Expected kernel trick $\star$ We now describe what we term the *expected kernel trick*, which shall be useful when finding the moments in a GP with uncertain inputs. When using SE-ARD kernels we can reinterpret an expectation with regard to an uncertain input in a probabilistic manner, which allows us to simplify the operation of convolving the input distribution with the kernel using standard Gaussian manipulations. Note that this trick bears little relation to the standard *kernel trick*. Although in the GP model with uncertain test inputs the training inputs $\mathbf{x} \in \mathbb{R}^D$ are considered deterministic, we treat them as random in the expected kernel trick. This is purely for mathematical convenience since the equations are in the same form as a model where $\mathbf{x}$ is random; and hence it is the denoted expected kernel trick. For instance,

$$k(\mathbf{x}_i, \mathbf{x}_\star) = c\mathcal{N}(\mathbf{x}_i | \mathbf{x}_\star, \boldsymbol{\Lambda}) \tag{3.41}$$

$$\implies \mathbb{E}_{\mathbf{x}_\star}[k(\mathbf{x}_i, \mathbf{x}_\star) | \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int c\mathcal{N}(\mathbf{x}_i | \mathbf{x}_\star, \boldsymbol{\Lambda})\mathcal{N}(\mathbf{x}_\star | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x}_\star \tag{3.42}$$

$$\overset{(A.35)}{=} c \int \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_\star \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma} + \boldsymbol{\Lambda} & \boldsymbol{\Sigma} \\ \boldsymbol{\Sigma} & \boldsymbol{\Sigma} \end{bmatrix} \right) d\mathbf{x}_\star \,. \tag{3.43}$$

We now break the Gaussian into its conditionals and utilize that probability distributions integrate to one:

$$\mathbb{E}_{\mathbf{x}_\star}[k(\mathbf{x}_i, \mathbf{x}_\star) | \boldsymbol{\mu}, \boldsymbol{\Sigma}] = c\tilde{p}(\mathbf{x}_i) \int \tilde{p}(\mathbf{x}_\star | \mathbf{x}_i) d\mathbf{x}_\star \tag{3.44}$$

$$\overset{(A.32)}{=} c\mathcal{N}(\mathbf{x}_i | \mu, \boldsymbol{\Sigma} + \boldsymbol{\Lambda}) \in \mathbb{R}^+ \,. \tag{3.45}$$

Probability manipulations assuming the algebraic convenience model where $\mathbf{x}_i$ and $\mathbf{x}_\star$ are jointly Gaussian and unknown are demarcated with $\tilde{p}(\cdot)$ and $\tilde{\mathbb{E}}[\cdot]$ to distinguish them the GP model we are using elsewhere throughout this section. Since the kernel is an unnormalized Gaussian we must use a normalizer $c := \alpha^2 (2\pi)^{D/2} |\boldsymbol{\Lambda}|^{\frac{1}{2}} \in \mathbb{R}^+$. The covariance and mean of the joint normal is constructed using the mixing property of Gaussians (A.35). We also find the following expression useful

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k(\mathbf{x}_i, \mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\
&= \int \mathbf{x}_\star c\mathcal{N}(\mathbf{x}_i|\mathbf{x}_\star, \boldsymbol{\Lambda})\mathcal{N}(\mathbf{x}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{x}_\star && (3.46)\\
&\overset{(A.35)}{=} c\int \mathbf{x}_\star \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_i\\ \mathbf{x}_\star\end{bmatrix}\middle|\middle|\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}+\boldsymbol{\Lambda} & \boldsymbol{\Sigma}\\ \boldsymbol{\Sigma} & \boldsymbol{\Sigma}\end{bmatrix}\right) d\mathbf{x}_\star\,. && (3.47)
\end{aligned}
$$

We again break the joint into its conditionals and turn the integral into an expectation:

$$
\begin{aligned}
\mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k(\mathbf{x}_i, \mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &= c\tilde{p}(\mathbf{x}_i)\int \mathbf{x}_\star \tilde{p}(\mathbf{x}_\star|\mathbf{x}_i)d\mathbf{x}_\star && (3.48)\\
&\overset{(A.32)}{=} c\mathcal{N}(\mathbf{x}_i|\mu, \boldsymbol{\Sigma}+\boldsymbol{\Lambda})\tilde{\mathbb{E}}[\mathbf{x}_\star|\mathbf{x}_i] && (3.49)\\
&\overset{(A.34)}{=} c\mathcal{N}(\mathbf{x}_i|\mu, \boldsymbol{\Sigma}+\boldsymbol{\Lambda})(\boldsymbol{\mu}+\boldsymbol{\Sigma}(\boldsymbol{\Sigma}+\boldsymbol{\Lambda})^{-1}(\mathbf{x}_i-\boldsymbol{\mu})) && (3.50)\\
&\overset{(3.45)}{=} \mathbb{E}_{\mathbf{x}_\star}[k(\mathbf{x}_i, \mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}](\boldsymbol{\mu}+\boldsymbol{\Sigma}(\boldsymbol{\Sigma}+\boldsymbol{\Lambda})^{-1}(\mathbf{x}_i-\boldsymbol{\mu})) \in \mathbb{R}\,. && (3.51)
\end{aligned}
$$

Finally, we might be in a scenario where the following has to be computed:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}_\star}[k_a(\mathbf{x}_i, \mathbf{x}_\star)k_b(\mathbf{x}_j, \mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] && (3.52)\\
&= c_a c_b \int \mathcal{N}(\mathbf{x}_i|\mathbf{x}_\star, \boldsymbol{\Lambda}_a)\mathcal{N}(\mathbf{x}_j|\mathbf{x}_\star, \boldsymbol{\Lambda}_b)\mathcal{N}(\mathbf{x}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{x}_\star && (3.53)\\
&\overset{(A.35)}{=} c_a c_b \int \mathbf{x}_\star \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_\star\\ \mathbf{x}_i\\ \mathbf{x}_j\end{bmatrix}\middle|\middle|\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma} & \boldsymbol{\Sigma} & \boldsymbol{\Sigma}\\ \boldsymbol{\Sigma} & \boldsymbol{\Sigma}+\boldsymbol{\Lambda}_a & \boldsymbol{\Sigma}\\ \boldsymbol{\Sigma} & \boldsymbol{\Sigma} & \boldsymbol{\Sigma}+\boldsymbol{\Lambda}_b\end{bmatrix}\right) d\mathbf{x}_\star && (3.54)\\
&= c_a c_b \tilde{p}(\mathbf{x}_i, \mathbf{x}_j)\int \tilde{p}(\mathbf{x}_\star|\mathbf{x}_i, \mathbf{x}_j)d\mathbf{x}_\star && (3.55)\\
&\overset{(A.32)}{=} c_a c_b \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_i\\ \mathbf{x}_j\end{bmatrix}\middle|\middle|\begin{bmatrix}\mu\\ \mu\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}+\boldsymbol{\Lambda}_a & \boldsymbol{\Sigma}\\ \boldsymbol{\Sigma} & \boldsymbol{\Sigma}+\boldsymbol{\Lambda}_b\end{bmatrix}\right) \in \mathbb{R}^+\,. && (3.56)
\end{aligned}
$$

The normalizers $c_a \in \mathbb{R}^+$ and $c_b \in \mathbb{R}^+$ are the same as c except they use the hyper-parameters for k_a and k_b, respectively. These same relations were shown in Deisenroth [2009, Ch. 2]; we have used the expected kernel trick to provide a simplified derivation. The expected kernel trick provides a more elegant derivation of these relations. Although we exclusively work with a SE-ARD kernel with respect to uncertain inputs, similar relations are possible for any kernel we can integrate with respect to a Gaussian such as polynomial and trigonometric kernels.

Univariate output We begin by finding the predictive moments (mean and variance) of a GP with uncertain inputs in the univariate case, after which we progress to the multivariate scenario. For a univariate output, we find the predictive mean $\boldsymbol{\mu}_* \in \mathbb{R}$ of $p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ using the law of iterated expectations (A.40),

$$\begin{aligned}\boldsymbol{\mu}_* := \mathbb{E}_{\mathbf{x}_\star, f}[f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &\overset{(A.40)}{=} \mathbb{E}_{\mathbf{x}_\star}[\mathbb{E}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &(3.57)\\ &\overset{(3.18)}{=} \mathbb{E}_{\mathbf{x}_\star}[\boldsymbol{\beta}^\top \mathbf{K}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &(3.58)\\ &= \boldsymbol{\beta}^\top \mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &(3.59)\\ &\overset{(3.18)}{=} \boldsymbol{\beta}^\top \mathbf{q}\,, &(3.60)\end{aligned}$$

where $\mathbf{q} := [q_1, \dots, q_N]^\top \in (\mathbb{R}^+)^N$. Using (3.45) we see that

$$\begin{aligned}q_i &:= \mathbb{E}_{\mathbf{x}_\star}[k(\mathbf{x}_i, \mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &(3.61)\\ &\overset{(3.45)}{=} \alpha^2|\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^\top(\boldsymbol{\Sigma} + \boldsymbol{\Lambda})^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\right)\,. &(3.62)\end{aligned}$$

Similarly we find the predictive variance $\sigma_*^2 \in \mathbb{R}^+$ using the law of total variance (A.41):

$$\begin{aligned}\sigma_*^2 &:= \mathrm{Var}_{\mathbf{x}_\star, f}[f(\mathbf{x}_\star)] &(3.63)\\ &\overset{(A.41)}{=} \mathbb{E}_{\mathbf{x}_\star}[\mathrm{Var}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \mathrm{Var}_{\mathbf{x}_\star}[\mathbb{E}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\,. &(3.64)\end{aligned}$$

We can reformulate the first term

$$\begin{aligned}\mathbb{E}_{\mathbf{x}_\star}[\mathrm{Var}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &\overset{(3.16)}{=} \mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_{\star\star}|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_\star^\top \mathbf{K}^{-1}\mathbf{K}_\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &(3.65)\\ &\overset{(A.26)}{=} \alpha^2 - \mathrm{tr}\left(\mathbf{K}^{-1}\mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_\star\mathbf{K}_\star^\top|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\right)\,, &(3.66)\end{aligned}$$

where we have made use of the *cyclic property* (A.26) of the trace. Unlike others in the literature we include the measurement noise on the training set on the diagonal of $\mathbf{K}$. Meaning, where we use $\mathbf{K}$ we would write $(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}$ in the other convention. We can deal with the second term:

$$
\begin{aligned}
\mathrm{Var}_{\mathbf{x}_\star}[\mathbb{E}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &= \mathbb{E}_{\mathbf{x}_\star}[\mathbb{E}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]^2|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\mathbf{x}_\star}[\mathbb{E}_f[f(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]^2 \quad (3.67)\\
&= \mathbb{E}_{\mathbf{x}_\star}[(\boldsymbol{\beta}^\top \mathbf{K}_\star)(\mathbf{K}_\star^\top \boldsymbol{\beta})|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - (\boldsymbol{\beta}^\top \mathbf{q})^2 \quad (3.68)\\
&= \boldsymbol{\beta}^\top \mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_\star \mathbf{K}_\star^\top|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\boldsymbol{\beta} - (\boldsymbol{\beta}^\top \mathbf{q})^2 \,. \quad (3.69)
\end{aligned}
$$

Together (3.66) and (3.69) simplify to

$$
\begin{aligned}
\sigma_*^2 &= \alpha^2 - \operatorname{tr}\left(\mathbf{K}^{-1}\mathbf{Q}\right) + \boldsymbol{\beta}^\top \mathbf{Q} \boldsymbol{\beta} - {\boldsymbol{\mu}_*}^2 \,, \quad (3.70)\\
\mathbf{Q} &:= \mathbb{E}_{\mathbf{x}_\star}[\mathbf{K}_\star \mathbf{K}_\star^\top|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \,. \quad (3.71)
\end{aligned}
$$

We find the entries of the symmetric matrix $\mathbf{Q} \in (\mathbb{R}^+)^{N\times N}$ using (3.56)

$$
\begin{aligned}
\tilde{Q}_{ij} &= \mathbb{E}_{\mathbf{x}_\star}[k(\mathbf{x}_i, \mathbf{x}_\star)k(\mathbf{x}_\star, \mathbf{x}_j)|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\\
&= \frac{k(\mathbf{x}_i, \boldsymbol{\mu})k(\mathbf{x}_j, \boldsymbol{\mu})}{|2\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{\frac{1}{2}}} \exp\left((\tilde{\mathbf{z}}_{ij} - \boldsymbol{\mu})^\top (\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{\Lambda})^{-1} \boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} (\tilde{\mathbf{z}}_{ij} - \boldsymbol{\mu})\right) \,, \quad (3.72)
\end{aligned}
$$

where we have used $\tilde{\mathbf{z}}_{ij} := \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j) \in \mathbb{R}^D$ for brevity.

Multivariate outputs In the multivariate scenario we will consider, for simplicity, the case where the predictive mean vector $\boldsymbol{\mu}_* \in \mathbb{R}^E$ of $p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is merely the concatenation of E independent predictive means. Meaning,

$$
f_a(\mathbf{x}_\star) \perp\!\!\!\perp f_b(\mathbf{x}_\star)|\mathbf{x}_\star \,, \quad (3.73)
$$

for all $a, b \in \{1, \dots, E\}$ and $a \neq b$. We then obtain the predictive mean vector

$$
\boldsymbol{\mu}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\beta}_1^\top \mathbf{q}_1 & \dots & \boldsymbol{\beta}_E^\top \mathbf{q}_E \end{bmatrix}^\top = \operatorname{diag}(\boldsymbol{\beta}_{1:E}^\top \mathbf{q}_{1:E}) \,, \quad (3.74)
$$

where $\mathbf{q}_i$ is computed using (3.62) with length-scales, the input-scales, for the respective output dimension $\boldsymbol{\Lambda}_i$. The diagonal entries of the predictive covariance $\boldsymbol{\Sigma}_* \in \mathbb{S}^E$ are the marginal predictive variances of each target dimension individually. For notational brevity we abbreviate $f_a(\mathbf{x}_\star)$ by $f_a^\star$ for $a \in \{1, \dots, E\}$. Despite the independence of the outputs when the inputs are known (3.73), uncertain inputs result in target dimensions that covary. In other words, the predictive covariance contains non-diagonal terms:

$$\boldsymbol{\Sigma}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma} = \begin{bmatrix} \mathrm{Var}\,[f_1^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \dots & \mathrm{Cov}\,[f_1^\star, f_E^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}\,[f_E^\star, f_1^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \dots & \mathrm{Var}\,[f_E^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \end{bmatrix} . \tag{3.75}$$

The non-diagonal terms of $\boldsymbol{\Sigma}_*$, the cross-covariances, are given by

$$\mathrm{Cov}\,[f_a^\star, f_b^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{f,\mathbf{x}_\star}[f_a^\star f_b^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{f,\mathbf{x}_\star}[f_a^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\mathbb{E}_{f,\mathbf{x}_\star}[f_b^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] . \tag{3.76}$$

We separate the expectations using the conditional independence in the deterministic case, $f_a^\star \perp\!\!\!\perp f_b^\star|\mathbf{x}_\star$, to obtain

$$\mathbb{E}_{f,\mathbf{x}_\star}[f_a^\star f_b^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \overset{(3.73)}{=} \mathbb{E}_{\mathbf{x}_\star}\left[\mathbb{E}_{f_a}[f_a^\star|\mathbf{x}_\star]\,\mathbb{E}_{f_b}[f_b^\star|\mathbf{x}_\star]|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right] . \tag{3.77}$$

Plugging in the mean function from (3.18),

$$\mathbb{E}_{f_a}[f_a^\star|\mathbf{x}_\star] = k_a(\mathbf{x}_\star, \mathbf{X})\,\underbrace{\mathbf{K}_a^{-1}\mathbf{y}_a}_{=:\boldsymbol{\beta}_a} , \tag{3.78}$$

we see that

$$\mathbb{E}_{f,\mathbf{x}_\star}[f_a^\star f_b^\star|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \overset{(3.77)}{=} \mathbb{E}_{\mathbf{x}_\star}[\boldsymbol{\beta}_a^\top k_a(\mathbf{x}_\star, \mathbf{X})^\top k_b(\mathbf{x}_\star, \mathbf{X})\boldsymbol{\beta}_b] \tag{3.79}$$

$$= \boldsymbol{\beta}_a^\top \underbrace{\mathbb{E}_{\mathbf{x}_\star}[k_a(\mathbf{x}_\star, \mathbf{X})^\top k_b(\mathbf{x}_\star, \mathbf{X})]}_{=:\mathbf{Q}^{ab}} \boldsymbol{\beta}_b , \tag{3.80}$$

where the $\boldsymbol{\beta}$ terms have are pulled out of the integral since they are independent of the test input $\mathbf{x}_\star$. We find the entries $\mathbf{Q}^{ab} \in (\mathbb{R}^+)^{N\times N}$ using (3.56)

$$
\begin{aligned}
Q_{ij}^{ab} &= \mathbb{E}_{\mathbf{x}_\star}[k_a(\mathbf{x}_i, \mathbf{x}_\star)k_b(\mathbf{x}_\star, \mathbf{x}_j)] && (3.81)\\
&= \alpha_a^2\alpha_b^2|(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})\boldsymbol{\Sigma} + \mathbf{I}|^{-\frac{1}{2}} \\
&\times \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}(\mathbf{x}_i - \mathbf{x}_j)\right) \\
&\times \exp\left(-\frac{1}{2}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})^\top((\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})^{-1} + \boldsymbol{\Sigma})^{-1}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})\right), && (3.82)\\
\hat{\mathbf{z}}_{ij} &:= \boldsymbol{\Lambda}_b(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_i + \boldsymbol{\Lambda}_a(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_j \in \mathbb{R}^D . && (3.83)
\end{aligned}
$$

Note that there is a different $\mathbf{Q}^{ab}$ for each combination of output dimensions a and b, and $\mathbf{Q}^{ab}$ equals $\mathbf{Q}$ in (3.72) for identical target dimensions $a = b$.

We have now found the exact mean $\boldsymbol{\mu}_*$ and covariance $\boldsymbol{\Sigma}_*$ of the non-Gaussian predictive density $p(f(\mathbf{x}_\star)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with $f \sim \mathcal{GP}$ and $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Algorithm 1 Gaussian process regression with uncertain inputs

```
Require: Use SE-ARD covariance $k$
 1: function GPUR($\xi \in (\mathbb{R}^+)^{D+2\times E}, \mathbf{x} \in \mathbb{R}^{N\times D}, \mathbf{y} \in \mathbb{R}^{N\times E}, \boldsymbol{\mu} \in \mathbb{R}^D, \boldsymbol{\Sigma} \in \mathbb{S}^D$)
 2:     for $a$ in 1 to $E$ do
 3:         $(\boldsymbol{\Lambda}, \alpha_a, \sigma_{\varepsilon_a}^2) \leftarrow \xi_a$
 4:         Calculate covariance matrix $\mathbf{K}_a$ from $\mathbf{x}$ and $\xi_a$
 5:         $\boldsymbol{\beta}_a \leftarrow \mathbf{K}_a^{-1}\mathbf{y}_a$
 6:         Find $\mathbf{q}_a$ using (3.62)
 7:         Find $\mathbf{V}(:, a)$ using (3.90)
 8:     end for
 9:     for $a$ and $b$ in 1 to $E$ do
10:         Find $\mathbf{Q}^{ab}$ using (3.82)
11:         $\mathbf{M}_{ab} \leftarrow \boldsymbol{\beta}_a^\top \mathbf{Q}^{ab}\boldsymbol{\beta}_b$          ▷ Find the second moment using (3.80)
12:     end for
13:     ${\boldsymbol{\mu}_*}^\top \leftarrow \mathbf{1}^\top(\boldsymbol{\beta}_{1:E} \odot \mathbf{q}_{1:E})$          ▷ Using (3.74) and (A.1)
14:     $\boldsymbol{\Sigma}_* \leftarrow \mathbf{M} - \boldsymbol{\mu}_*{\boldsymbol{\mu}_*}^\top + \mathrm{diag}(\sigma_\varepsilon^2)$          ▷ Includes measurement noise
15:     return $\boldsymbol{\mu}_* \in \mathbb{R}^E, \boldsymbol{\Sigma}_* \in \mathbb{S}^E, \mathbf{V} \in \mathbb{R}^{D\times E}$
16: end function
```

When using GPs for filtering we need to compute the input-output covariance. This is the only moment of interest of the joint distribution on ($\mathbf{x}_\star$,$\mathbf{y}_\star$) we have not yet computed. Fortunately, the input-output covariance can be computed analytically as well:

$$\begin{aligned}\mathrm{Cov}_{\mathbf{x}_\star,f_a}[\mathbf{x}_\star, f_a(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma}] &= \mathbb{E}_{\mathbf{x}_\star,f_a}[\mathbf{x}_\star f_a(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma}] \\ &\quad - \mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star|\boldsymbol{\mu},\boldsymbol{\Sigma}]\mathbb{E}_{\mathbf{x}_\star,f_a}[f_a(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma}] \in (\mathbb{R}^+)^{D\times E}\,. \end{aligned} \tag{3.84}$$

We first note that:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}_\star,f_a}[\mathbf{x}_\star f_a(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma}] &= \mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star\mathbb{E}_{f_a}[f_a(\mathbf{x}_\star)|\mathbf{x}_\star]|\boldsymbol{\mu},\boldsymbol{\Sigma}] && (3.85)\\ &= \mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k_a(\mathbf{x}_\star,\mathbf{X})\boldsymbol{\beta}_a|\boldsymbol{\mu},\boldsymbol{\Sigma}] && (3.86)\\ &= \mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k_a(\mathbf{x}_\star,\mathbf{X})|\boldsymbol{\mu},\boldsymbol{\Sigma}]\boldsymbol{\beta}_a && (3.87)\\ &= \sum_{i=1}^{N}\boldsymbol{\beta}_{ai}\mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k_a(\mathbf{x}_\star,\mathbf{x}_i)] \in (\mathbb{R}^+)^D\,. && (3.88)\end{aligned}$$

Substituting back into (3.84) and applying (3.51) we compute each column individually with

$$\begin{aligned}\mathrm{Cov}_{\mathbf{x}_\star,f_a}[\mathbf{x}_\star, f_a(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma}] &= \sum_{i=1}^{N}\boldsymbol{\beta}_{ai}\mathbb{E}_{\mathbf{x}_\star}[\mathbf{x}_\star k_a(\mathbf{x}_\star,\mathbf{x}_i)] - \boldsymbol{\mu}\mathbb{E}_{\mathbf{x}_\star}[k_a(\mathbf{x}_\star,\mathbf{x}_i)] && (3.89)\\ &= \sum_{i=1}^{N}\boldsymbol{\beta}_{ai}q_{ai}\boldsymbol{\Sigma}(\boldsymbol{\Sigma}+\Lambda_a)^{-1}(\mathbf{x}_i-\boldsymbol{\mu}) \in (\mathbb{R}^+)^D\,. && (3.90)\end{aligned}$$

We have now computed all the moments, up to second order, of $p(\mathbf{x}_\star, f(\mathbf{x}_\star)|\boldsymbol{\mu},\boldsymbol{\Sigma})$. We summarize all the calculations necessary to do so in Algorithm 1.

We have built upon Ghahramani and Roweis [1999] which utilized the tractability of propagating normally distributed uncertain inputs through radial basis function (RBF) regressors. We have exploited the RBF structure of GP predictive mean and variances to find exact predictive moments for uncertain test inputs in a GP. The derivations are relatively technical. However, GP prediction with uncertain inputs is abstracted into a black box when called upon in later chapters.

3.3 Multiple Views of Gaussian Processes

The normal distribution on an infinitely long vector view, known as the function space view, is perhaps the most direct way to describe a GP. However, there are many ways to describe and derive a GP, which is a sign of their generality. GPs have links to many well-known methods including Bayesian linear regression, artificial neural networks, and splines. Understanding different views of GPs adds to intuition and helps build a unifying framework. Additionally, different descriptions of a GP are more convenient, either theoretically or computationally, depending on the circumstance.

Weight space Another view, the weight space view, is to think of a GP as Bayesian linear regression with an infinite number of basis functions with Gaussian priors on the weights. If we use the eigenfunctions of the covariance k as the basis functions we get equivalent predictions to the GP described by the function space view. The setup is:

$$\mathbf{y} = \mathbf{\Phi} w \,, \quad \mathbf{\Phi}(i,j) := \phi_j(\mathbf{x}_i) \,, \quad w_j \sim \mathcal{N}(0, \lambda_j) \,, \quad \text{for} \quad j \in \mathbb{N} \,, \tag{3.91}$$

where $\phi_j \in \mathbb{R}^D \to \mathbb{R}$ and $\lambda_j \in \mathbb{R}^+$ are the jth eigenfunction and eigenvalue of the covariance k, respectively. We reconstruct the covariance matrix by

$$\mathbf{K}(i,j) = \sum_{k=1}^{\infty} \phi_k(\mathbf{x}_i) \lambda_k \phi_k(\mathbf{x}_j) \,. \tag{3.92}$$

Here we moved from working with an infinite number of basis functions (3.91) to an $\mathbb{S}^N$ covariance, or kernel, matrix (3.92); this is an instance of a general method known as the *kernel trick* [Schölkopf and Smola, 2001, Ch. 1]. The weight space view is generalized to the mathematically "deeper" reproducing kernel Hilbert space (RKHS) [Schölkopf and Smola, 2001, Ch. 2] view, which is beyond the scope of this thesis.

Neural networks A bridge between neural networks [Rumelhart et al., 1986] and GPs was created by Neal [1996] who showed that a Bayesian neural network with an infinite number of hidden units is equivalent to a GP with a particular covariance function. Representations of new sparse GP approximations as neural networks by Lázaro-Gredilla [2010, Ch. 3] shows the field has come full circle in some respects.

Cholesky space $\star$ We can also link GPs to moving averages, which we call the Cholesky view. Consider the process of sampling synthetic data. We sample from a multivariate normal (and therefore a GP) using its Cholesky factorization:

$$\mathbf{y} = \mathrm{chol}(\mathbf{K})^\top \mathbf{w}\,, \quad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\,. \tag{3.93}$$

This is interpreted as the data $\mathbf{y}$ being a smoothed version of white noise $\mathbf{w} \in \mathbb{R}^N$ as in a moving average. Since the Cholesky factorization is upper triangular, this has a causal interpretation: The noise w_i only influences data points $\mathbf{y}_{i:N}$. We can interpret the entries in the Cholesky factorization as the representing the relationship between nodes in a directed Gaussian belief network [Weiss and Freeman, 2001]. This view is more naturally suited towards the graphical model representation of Figure 1.8(a); in contrast, the function space view is naturally represented by Figure 3.2.

The Cholesky view naturally gives rise to a modeling check for a GP; we get the white noise back by multiplying by the inverse of the Cholesky factorization:

$$\mathbf{w} \overset{(3.93)}{=} \mathrm{chol}(\mathbf{K})^{-\top} \mathbf{y}\,. \tag{3.94}$$

If $\mathbf{w}$ does not look like white noise then it is a sign there is some misspecification in the covariance function or the noise model.

Radial basis functions We can interpret (3.18) as a radial basis function (RBF) predictor. In an RBF,

$$\mu_\star = \sum_{i=1}^{N} w_i k(\|\mathbf{x}_i - \mathbf{x}_\star\|)\,, \tag{3.95}$$

which is equivalent to a GP if $\mathbf{w} = \boldsymbol{\beta}$, and for isotropic covariance functions $\mathbf{K}_\star$ represents $k(\|\mathbf{x}_i - \mathbf{x}_\star\|)$. In general there is no clearly defined mechanism to determine $\mathbf{w}$ in RBFs, but often $w_i = 1/N$. In a GP the weights are inferred using (3.17).

Each of these methods may provide some additional insight into GPs by their connections to GPs. The list is not exhaustive, connections exist with other methods as well. The generality of the GP framework is evidenced by how many methods are special cases of GPs, or limiting cases corresponding to GPs.

3.4 Gaussian Process Time Series

In this section we cover the first of two approaches for GP time series modeling. We describe efficient computational methods for prediction on a fixed and variable horizon as well as for hyper-parameter learning. In this section we compare different methods for attaining the necessary computational savings. These computational methods are useful for the time series approaches discussed in this section alone, but are even more necessary when we combine GPs and change point methods in Chapter 5. The bulk of this section will be literature review. However, we will provide some novel distinctions, clarifications, and slight extensions.

In a Gaussian process time series (GPTS) the time index t is treated as the input while the time series observation y_t is the output:

$$y_t = f(t) + \epsilon_t\,, \quad f \sim \mathcal{GP}(0, k)\,, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_n^2)\,. \tag{3.96}$$

GPTS generalizes many of the classic time series models such as AR, the autoregressive moving average (ARMA) [Murray-Smith and Girard, 2001], and the Kalman filter.

There are many opportunities to improve upon naive inference in GPTS. When making online predictions, ordinary inference in a GP requires $\mathcal{O}(T^3)$ computational time to invert the covariance matrix every time a new data point is encountered. This results in $\mathcal{O}(T^4)$ total computation time for naive implementation. This can be brought down to $\mathcal{O}(T^3)$ using rank-1 update methods (Section 3.5 and 3.6). However, we can leverage the fact that the GP input is one dimensional in GPTS using two different approaches. If the time series is sampled using uniform sampling, the covariance matrix becomes *Toeplitz*. Therefore, we can do predictions using the Yule-Walker equations (see Golub and Van Loan [1996, Sec. 4.7]) in $\mathcal{O}(T^2)$ time, which we call GP Yule-Walker (GPYW).

However, we lower the computational cost further to $\mathcal{O}(T)$ by converting the GPTS model to a state space representation using Kalman filtering using the methods in Hartikainen and Särkkä [2010]; we refer to this method as GP Kalman (GPK). We can also eliminate the uniform sampling requirement by converting the GPTS to a continuous time Kalman filter representation. This approach works for most, but not all, stationary covariance functions. By contrast, the Yule-Walker approach works with any stationary covariance function. Different covariance functions have different latent dimensionality D in the state space representation. Taking D into account the computational cost is $\mathcal{O}(D^3T)$. Note that GPK does not reduce the generally $\mathcal{O}(T^3)$ problem of Gaussian process regression to $\mathcal{O}(T)$ in general since the trick only works in a one-dimensional input space. Hence, it works in GPTS where the input is time t, a one-dimensional quantity.

3.4.1 Toeplitz Methods

In this section we cover how to use GPYW for (fixed and variable horizon) prediction and hyper-parameter learning. If we are using a stationary covariance function[1] $k(t-t')$ and the time series is uniformly spaced, the covariance matrix of $\mathbf{y}$ is

$$\mathbf{K} = \begin{bmatrix} k(0) & k(1) & \dots & k(T-1) \\ k(1) & k(0) & \dots & k(T-2) \\ \vdots & \vdots & \ddots & \vdots \\ k(T-1) & k(T-2) & \dots & k(0) \end{bmatrix}, \tag{3.97}$$

which exactly matches the form of a symmetric Toeplitz matrix. We use the function Toeplitz($\mathbf{v}$) to generate a Toeplitz matrix from its first column $\mathbf{v}$, which

[1] We abbreviate $k(t,t')$ as $k(t-t')$ for stationary covariance functions.

implies

$$\mathbf{K} = \mathrm{Toeplitz}(\begin{bmatrix} k(0) & k(1) & \dots & k(T-1) \end{bmatrix}^\top) . \tag{3.98}$$

Using the Yule-Walker equations, we solve for an arbitrary vector $\mathbf{z} \in \mathbb{R}^N$ in

$$\mathbf{T}\mathbf{z} = -\mathbf{r} \in \mathbb{R}^N , \tag{3.99}$$

$$\mathbf{T} := \mathrm{Toeplitz}(\begin{bmatrix} \rho & \mathbf{r}_{1:N-1}^\top \end{bmatrix}^\top) \in \mathbb{R}^{N \times N} . \tag{3.100}$$

This requires $\mathcal{O}(N^3)$ computation and $\mathcal{O}(N^2)$ memory in the naive method but only $\mathcal{O}(N^2)$ computation and $\mathcal{O}(N)$ memory using the Yule-Walker equations, shown in Algorithm 2 also known as the Durbin algorithm. We generalize to the case where $\mathbf{T}$ is an arbitrary Toeplitz matrix, unrelated to $\mathbf{r}$, using the Levinson algorithm, which is more expensive than Yule-Walker by a constant factor. Furthermore, it is possible to compute the inverse $\mathbf{T}^{-1}$ using the Trench algorithm [Trench, 1964]. The inverse $\mathbf{T}^{-1}$ is persymmetric but not Toeplitz and therefore requires $\mathcal{O}(N^2)$ storage. Consequently the inverse $\mathbf{T}^{-1}$ should be avoided when possible.

Algorithm 2 Implementation of the Yule-Walker equations. Also known as the Durbin algorithm. Solves for $\mathbf{z}$ in $\mathrm{Toeplitz}(\begin{bmatrix} \rho & \mathbf{r}_{1:N-1} \end{bmatrix})\mathbf{z} = -\mathbf{r}$.

```
1:  function YuleWalker($\rho \in \mathbb{R}^+, \mathbf{r} \in \mathbb{R}^N$)
2:      $\mathbf{r} \leftarrow \mathbf{r}/\rho$                ▷ Rest of algorithm assumes $\mathrm{diag}(\mathbf{T}) = 1$.
3:      $(\mathbf{z}(1), \alpha, \beta) \leftarrow (-\mathbf{r}(1), -\mathbf{r}(1), 1)$
4:      for $k = 1$ to $N-1$ do
5:          $\beta \leftarrow (1 - \alpha^2)\beta$
6:          $\alpha \leftarrow -(\mathbf{r}(k+1) + \mathbf{r}(1{:}k)^\top \mathbf{E}\mathbf{z}(1{:}k))/\beta$   ▷ Use exchange matrix $\mathbf{E}$ (A.5)
7:          $\mathbf{z}(1{:}k) \leftarrow \mathbf{z}(1{:}k) + \alpha \mathbf{E}\mathbf{z}(1{:}k)$
8:          $\mathbf{z}(k+1) \leftarrow \alpha$
9:      end for
10:     return $\mathbf{z}$
11: end function
```

One step prediction If we want to predict the next step into the future using the last N observations, we can use the Yule-Walker setup. Using the standard

GP predictive mean equation (3.15):

$$\mathbb{E}\left[y_t | \mathbf{y}_{(N)}\right] \overset{(3.18)}{=} \mathbf{K}_\star^\top \mathbf{K}^{-1} \mathbf{y}_{(N)} \overset{(A.6)}{=} \underbrace{\mathbf{K}_\star^\top \mathbf{E} \mathbf{K}^{-1}}_{=:\boldsymbol{\alpha}^\top} \mathbf{E}\mathbf{y}_{(N)}\,, \tag{3.101}$$

$$\implies \boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{E}\mathbf{K}_\star \implies \mathbf{K}\boldsymbol{\alpha} = \mathbf{E}\mathbf{K}_\star \implies \mathbf{K}(-\boldsymbol{\alpha}) = -\mathbf{E}\mathbf{K}_\star\,, \tag{3.102}$$

$$\mathbf{K}_\star := k(t-N{:}t-1, t) = \mathbf{E}k(1{:}N)\,, \tag{3.103}$$

$$\mathbf{K} := \text{Toeplitz}(\begin{bmatrix}\mathbf{K}_{\star\star} & \mathbf{E}\mathbf{K}_\star\end{bmatrix})\,. \tag{3.104}$$

Since $\mathbf{K}^{-1}$ is persymmetric, we can apply the exchange matrix to both sides $\mathbf{K}^{-1} = \mathbf{E}\mathbf{K}^{-1}\mathbf{E}$.[1] We must use the exchange matrix to flip $\mathbf{K}_\star$ to put it in the form required for the Yule-Walker equations (3.100):

$$-\boldsymbol{\alpha} = -\mathbf{K}^{-1}\mathbf{E}\mathbf{K}_\star \tag{3.105}$$

$$= -\begin{bmatrix} k(0) & k(1) & \dots & k(N-1) \\ k(1) & k(0) & \dots & k(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ k(N-1) & k(N-2) & \dots & k(0) \end{bmatrix}^{-1} \begin{bmatrix} k(1) \\ k(2) \\ \vdots \\ k(N) \end{bmatrix}, \tag{3.106}$$

Now we can make rolling forecasts using the last N observations in the time series using the inner product of $\boldsymbol{\alpha}$ and $\mathbf{E}\mathbf{y}_{(N)}$. This puts a GPTS in a linear autoregressive formulation. Although it seems we want to use all observations in the past, not only the last N, for forecasting, (3.102) is useful in Chapter 5. Additionally, if $\boldsymbol{\alpha}$ decays to negligibly small values for large N it is computationally cheaper to only predict using the last N observations yet only incurring a small approximation error. If we want to predict using all the observations so far, i.e. $N = t$ and therefore increasing, or the work varying predictive horizon then the Levinson algorithm is more appropriate.

Perhaps counter-intuitively, the predictive variances only depend on N and w and are invariant to $\mathbf{y}$. Therefore, we precompute the predictive variances before

[1] We use $\mathbf{E}$ to refer to the *exchange matrix* (A.5) such that $\mathbf{E}\mathbf{y}_{1:T} = \mathbf{y}_{T:-1:1}$.

seeing the data:

$$\mathrm{Var}\left[y_t|\mathbf{y}_{(N)}\right] = \sigma_\star^2 = \mathbf{K}_{\star\star} - \mathbf{K}_\star^\top \mathbf{E}\mathbf{K}^{-1}\mathbf{E}\mathbf{K}_\star = \mathbf{K}_{\star\star} - \mathbf{K}_\star^\top \mathbf{E}\boldsymbol{\alpha}\,. \tag{3.107}$$

Variable horizon If we are predicting in a variable horizon $w \in \mathbb{N}$ then it does not make sense to precompute and store $\boldsymbol{\alpha}$ since we have to store a different matrix for each prediction horizon leading to greater memory use. In this case it makes more sense to use the Levinson algorithm formulation:

$$\mathbb{E}\left[y_{t+w-1}|\mathbf{y}_{(N)}\right] = \mathbf{K}_\star^\top \underbrace{\mathbf{K}^{-1}\mathbf{y}_{(N)}}_{=:\boldsymbol{\beta}} \implies \mathbf{K}\boldsymbol{\beta} = \mathbf{y}_{(N)}\,, \tag{3.108}$$

$$\mathbf{K}_\star := k(t-N{:}t-1, t+w-1) \tag{3.109}$$

$$= \mathbf{E}k(w{:}N+w-1)\,, \tag{3.110}$$

which is the form of the Levinson algorithm, Algorithm 3. This is the same form for $\boldsymbol{\beta}$ as in (3.17). Now $\boldsymbol{\beta}$ is independent of $\mathbf{K}_\star$ and therefore we can use the same $\boldsymbol{\beta}$ for multiple prediction horizons. Since the right hand side of the linear system, $\mathbf{y}_{(N)}$, is arbitrary in Levinson, we do not have to bother with reversing the order of various vectors as is the case with Yule-Walker. We still need (3.107) to compute the predictive variances. We can do this using either the Durbin algorithm, or for multi-step predictive variances by using Levinson to solve $\mathbf{K}^{-1}\mathbf{E}\mathbf{K}_\star$. However, the predictive variances are computed before any data is observed, meaning that the online complexity is no greater by computing the predictive variances.

Hyper-parameter learning We also want to learn the covariance hyper-parameters ξ by maximizing the evidence. Although we could add the log likelihood of the one-step-ahead predictives, we do this more directly in the case of a GP:

$$\log p(\mathbf{y}|\xi) = -\frac{1}{2}\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| - \frac{T}{2}\log 2\pi \tag{3.111}$$

$$= -\frac{1}{2}\mathbf{y}^\top\boldsymbol{\beta} - \frac{1}{2}\log|\mathbf{K}| - \frac{T}{2}\log 2\pi\,. \tag{3.112}$$

The optional lines in Algorithm 3 are used to calculate the log determinant used here with negligible additional computational cost. In the case of uncertain out-

Algorithm 3 Implementation of the Levinson algorithm. Solves for $\mathbf{z}$ in $\text{Toeplitz}([\rho \quad \mathbf{r}_{1:N-1}])\mathbf{z} = \mathbf{b}$ and optionally find $\log \left|\text{Toeplitz}([\rho \quad \mathbf{r}_{1:N-1}])\right|$.

```
1: function LEVINSON($\rho \in \mathbb{R}^+, \mathbf{r} \in \mathbb{R}^N, \mathbf{b} \in \mathbb{R}^N$)
2:     $(\mathbf{r}, \mathbf{b}) \leftarrow (\mathbf{r}/\rho, \mathbf{b}/\rho)$          ▷ Rest of algorithm assumes $\text{diag}(\mathbf{T}) = 1$.
3:     $(\mathbf{x}(1), \mathbf{z}(1), \alpha, \beta) \leftarrow (\mathbf{b}(1), -\mathbf{r}(1), -\mathbf{r}(1), 1)$
4:     $D \leftarrow 0$
5:     for $k = 1$ to $N - 1$ do
6:         $\beta \leftarrow (1 - \alpha^2)\beta$
7:         $D \leftarrow D + \log\beta$          ▷ Optional: if we want to find $\log|\mathbf{T}|$
8:         $\mu \leftarrow (\mathbf{b}(k+1) - \mathbf{r}(1{:}k)^\top \mathbf{E}\mathbf{x}(1{:}k))/\beta$
9:         $\mathbf{x}(1{:}k) \leftarrow \mathbf{x}(1{:}k) + \mu\mathbf{E}\mathbf{z}(1{:}k)$
10:        $\mathbf{x}(k+1) \leftarrow \mu$
11:        if $k < N - 1$ then          ▷ Every iteration but the last
12:            $\alpha \leftarrow -(\mathbf{r}(k+1) + \mathbf{r}(1{:}k)^\top \mathbf{E}\mathbf{z}(1{:}k))/\beta$
13:            $\mathbf{z}(1{:}k) \leftarrow \mathbf{z}(1{:}k) + \alpha\mathbf{E}\mathbf{z}(1{:}k)$
14:            $\mathbf{z}(k+1) \leftarrow \alpha$
15:        end if
16:    end for
17:    $D \leftarrow D + N\log\rho$          ▷ Optional
18:    return $\mathbf{z}$ and $D = \log|\mathbf{T}|$
19: end function
```

put scale, we expand upon (3.33) to get:

$$\log p(\mathbf{y}|\xi, \alpha_0, \beta_0) = -\left(\alpha_0 + \frac{T}{2}\right)\log\left(1 + \frac{1}{2\alpha_0}\mathbf{y}^\top\left(\frac{\beta_0}{\alpha_0}\mathbf{K}\right)^{-1}\mathbf{y}\right) - \frac{1}{2}\log\left|\frac{\beta_0}{\alpha_0}\mathbf{K}\right|$$
$$+ \log\Gamma\left(\alpha_0 + \frac{T}{2}\right) - \log\Gamma(\alpha_0) - \frac{T}{2}\log(2\pi\alpha_0)\,, \tag{3.113}$$

which simplifies to

$$\log p(\mathbf{y}|\xi, \alpha_0, \beta_0) = -\left(\alpha_0 + \frac{T}{2}\right)\log\left(1 + \frac{1}{2\beta_0}\mathbf{y}^\top\boldsymbol{\beta}\right) - \frac{1}{2}\log|\mathbf{K}|$$
$$+ \log\Gamma\left(\alpha_0 + \frac{T}{2}\right) - \log\Gamma(\alpha_0) - \frac{T}{2}\log(2\pi\beta_0)\,. \tag{3.114}$$

Therefore, we can compute the evidence in $\mathcal{O}(T^2)$ time.

There exist faster Toeplitz system solvers based on fast Fourier transforms

(FFT) that run in $\mathcal{O}(T \log T)$ time [Cunningham et al., 2008, Sec. 4]. However, they do not give intermediate results, i.e. we cannot simultaneously obtain results for the first $t \leq T$ data points without incurring additional computational cost, as is the case with the Durbin algorithm, Algorithm 2. If we want to predict based on the last $M < N$ variables instead of the last N, we use the state of $\mathbf{z}$ after running the loop M iterations. However, when computing the marginal likelihood we do not care about the intermediate results, rendering the FFT method more appropriate.

Initialization When hyper-parameter learning in GPs is done by type-II MLE we use gradient based optimization to do learning. Since the marginal likelihood is typically non-convex, how we initialize the hyper-parameters is consequential. Local optima in GP hyper-parameter learning are typically not as problematic as they are in other methods such as neural networks. Typically, there are only a few local optima to the marginal likelihood.

The most general strategy is to merely sample the initial hyper-parameters randomly. However, we must pick a distribution from which we sample. Most of the hyper-parameters of a GP are not unitless and therefore rescale with the data. A fixed distribution to sample the initial hyper-parameters would be inappropriate since the effective distribution can be changed arbitrarily with a rescaling of the data. There we must adapt the initial guess distribution to the scale of the data. Hyper-parameters usually have the units of the output of the GP y or a particular input dimension $\mathbf{x}_i$. We estimate the scale of the data using the 95% quantile R to maintain robustness against a few outliers, we also consider the median distance Δ between data points on each of the input dimensions. Throughout this thesis we use the following initialization distribution on the initial parameter setting θ_0:

$$\log \theta_0 \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)\,, \tag{3.115}$$

$$\sigma_\theta^2 = (\log(c_1 R/\Delta)/4)^2\,, \quad \mu_\theta = \frac{1}{2}(\log(c_2 R\Delta))\,. \tag{3.116}$$

We motivate this rule by the intuition that it does not make much sense to initialize a length scale beyond the range of the data or smaller than the median distance between the data points. We cannot determine effects on scales much smaller than the distance between points. Likewise, we cannot determine effects on scales larger than the range of the data. Therefore, we if we want the initial point to be between these extremes 95% of the time we use initializer constants $c_1 \approx 1$ and $c_2 \approx 1$. Likewise for hyper-parameters matching with units on the output scale, R is output range and Δ is the typical distance between targets. After running a few iterations of learning with different initializations we select the solution with the largest marginal likelihood.

A level of arbitrariness is allowed in setting this rule given that it is only used for initialization. We only need to be within the *domain of influence* of the global optima. In LDS [van Overschee and de Moor, 1994] and HMMs [Hsu et al., 2009] there exist analytic initializers that have some guarantees of being "close" to the true parameters in some sense. Analogous methods for GP hyper-parameter initialization is an opportunity for future research.

3.4.2 GP Kalman

We can find the covariance function on the data implied by a linear dynamical system (LDS). A key quantity in doing this is the stationary distribution on the latent state $\mathbf{x}_t$. We combine the linear transformation property of Gaussians (A.36) with the definition of an LDS from (1.20) to propagate the covariance matrix one step into the future using the system matrix $\mathbf{A} \in \mathbb{R}^{D \times D}$ and the system noise $\mathbf{Q} \in \mathbb{S}^D$: $\mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^\top + \mathbf{Q}$. Therefore, the stationary covariance matrix $\boldsymbol{\Sigma}_x \in \mathbb{S}^D$ obeys the fixed point equation

$$\boldsymbol{\Sigma}_x = \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^\top + \mathbf{Q}\,. \tag{3.117}$$

If we iterate this fixed point equation from $\boldsymbol{\Sigma}_x = \mathbf{Q}$ we get

$$\boldsymbol{\Sigma}_x \overset{(3.117)}{=} \sum_{i=0}^{\infty} \mathbf{A}^i\mathbf{Q}(\mathbf{A}^i)^\top\,. \tag{3.118}$$

The parameterization of the latent space in an LDS is unidentifiable, we can transform the latent space such that $\mathbf{Q} = \mathbf{I}$ [Roweis and Ghahramani, 1999]. If the $\mathbf{A}$ in this transformed space is symmetric then the above expression simplifies to:

$$\boldsymbol{\Sigma}_x = \sum_{i=0}^{\infty} \mathbf{A}^i (\mathbf{A}^i)^\top = \sum_{i=0}^{\infty} (\mathbf{A}\mathbf{A}^\top)^i \implies \boldsymbol{\Sigma}_x \overset{(A.25)}{=} (\mathbf{I} - \mathbf{A}\mathbf{A}^\top)^{-1} , \tag{3.119}$$

using that $\boldsymbol{\Sigma}_x$ is a matrix geometric series in $\mathbf{A}\mathbf{A}^\top$.

LDS to GPTS To convert an LDS into a continuous time model like a GPTS, we must put the LDS in a continuous time form, which is a stochastic differential equation (SDE):

$$d\mathbf{x}_t = \mathbf{F}\mathbf{x}_t dt + \mathbf{G} d\mathbf{W}_t , \tag{3.120}$$

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \boldsymbol{\nu}_t , \tag{3.121}$$

where $d\mathbf{W}$ is a vector Weiner process. This implies that when propagating from a time t to a time s,

$$\mathbf{x}_s = \exp(\mathbf{F}|s-t|)\mathbf{x}_t + \mathbf{w}_{t,s} , \tag{3.122}$$

$$\mathbf{w}_{t,s} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}\mathbf{G}^\top |s-t|) . \tag{3.123}$$

Therefore, if we assume the discrete time version is sampled at unit time intervals, $s - t = 1$, then

$$\mathbf{A} \overset{(3.122)}{=} \exp(\mathbf{F}) , \quad \mathbf{Q} \overset{(3.123)}{=} \mathbf{G}\mathbf{G}^\top . \tag{3.124}$$

Given that we have a prior covariance of the stationary distribution on $\mathbf{x}_t$, we get a covariance between $\mathbf{x}_t$ and $\mathbf{x}_s$ of:

$$\boldsymbol{\Sigma}(s,t) := \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_s^\top\right] - \underbrace{\mathbb{E}\left[\mathbf{x}_t\right] \mathbb{E}\left[\mathbf{x}_s\right]^\top}_{=0} \overset{(A.36)}{=} \boldsymbol{\Sigma}_x \exp(\mathbf{F}|s-t|) , \tag{3.125}$$

which we refer to as $\boldsymbol{\Sigma}$ for short. If $\mathbf{x}_t$ is a scalar this reduces to a Laplace covariance function (3.7). We can also find the implied covariance between the noisy observations. We get the following joint distribution on $(\mathbf{x}_t, \mathbf{x}_s)$:

$$\begin{bmatrix} \mathbf{x}_t \\ \mathbf{x}_s \end{bmatrix} \overset{(A.36)}{=} \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_x & \boldsymbol{\Sigma} \\ \boldsymbol{\Sigma}^\top & \boldsymbol{\Sigma}_x \end{bmatrix}\right) . \tag{3.126}$$

Following the SDE equations:

$$\begin{bmatrix} \mathbf{y}_t \\ \mathbf{y}_s \end{bmatrix} \overset{(3.121)}{=} \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{x}_s \end{bmatrix} + \begin{bmatrix} \boldsymbol{\nu}_t \\ \boldsymbol{\nu}_s \end{bmatrix} , \tag{3.127}$$

where $\boldsymbol{\nu}_t$ and $\boldsymbol{\nu}_s$ are the measurement noise at time t and s, respectively. Therefore, the covariance on the observations $\mathbf{y}$ is:

$$\operatorname{Cov} \begin{bmatrix} \mathbf{y}_t \\ \mathbf{y}_s \end{bmatrix} \overset{(A.36)}{=} \begin{bmatrix} \mathbf{C}\boldsymbol{\Sigma}_x\mathbf{C}^\top + \mathbf{R} & \mathbf{C}\boldsymbol{\Sigma}\mathbf{C}^\top \\ \mathbf{C}\boldsymbol{\Sigma}^\top\mathbf{C}^\top & \mathbf{C}\boldsymbol{\Sigma}_x\mathbf{C}^\top + \mathbf{R} \end{bmatrix} . \tag{3.128}$$

GPTS to LDS We can use these equations to solve the reverse, and more computationally useful, problem, find the equivalent LDS from GPTS. If the GPTS observations $\mathbf{y}$ are univariate, we often need a multivariate latent state $\mathbf{x}$ to account for the temporal correlations. We set $\mathbf{R}$ to be the measurement noise of the GPTS and set $\mathbf{C} = \begin{bmatrix} 1 & \mathbf{0} \end{bmatrix}$. What remains is to solve for a matrix $\mathbf{F} \in \mathbb{R}^{D \times D}$ such that,

$$\begin{bmatrix} 1 & \mathbf{0} \end{bmatrix} \boldsymbol{\Sigma}_x \exp(\mathbf{F}\Delta t) \begin{bmatrix} 1 & \mathbf{0} \end{bmatrix}^\top \overset{(3.125)}{=} k(\Delta t) , \tag{3.129}$$

for all $\Delta t > 0$. We can interpret the first element $\mathbf{x}_t(1)$ as $f(t)$ and $\mathbf{x}_t(i)$ as the $(i-1)$th derivative of $\mathbf{x}_t$. How many times differentiable samples from a GP with covariance k are determine D, the size of the matrix $\mathbf{F}$. For a Laplace, $\mathbf{F}$ is scalar; for a SE, $\mathbf{F}$ must be infinite; and for a Matérn, $\mathbf{F}$ is of finite size. In order for $\mathbf{F}$ to be of finite size, the spectral density of covariance function $S(\omega) := |\mathcal{F}\{k\}(\omega)|^2$ must be representable as:

$$S(\omega) \propto \frac{1}{\mathbb{P}(\omega^2)} , \tag{3.130}$$

where $\mathbb{P}(\cdot)$ represents any polynomial.

Hartikainen and Särkkä [2010] showed that we can find an appropriate $\mathbf{F}$ by taking the Fourier transform of the SDE. We summarize those key results in the next two equations. We must be able to put the spectral density $S(\omega)$ of the covariance function k in the form:

$$S(\omega) = \frac{q}{(\omega^2)^D + \sum_{i=0}^{D-1} a_i (\omega^2)^i}\,, \tag{3.131}$$

where $q \in \mathbb{R}^+$ is a constant. We can then put $\mathbf{F}$ in the form

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 & \dots & -a_{D-1} \end{bmatrix}\,. \tag{3.132}$$

We use q as the spectral density of the white noise process (the precision). Noise is present only on the last element $\mathbf{x}_m$.

3.4.3 Equivalent Covariances

All of the typical linear time series models such as autoregressive (AR), moving average (MA), autoregressive moving average (ARMA), and the Kalman filter (as shown in the previous section) are equivalent to a GPTS with the appropriate covariance matrix. By putting all the standard time series models in a GPTS framework we provide a unifying framework from what is typically a “zoo” of different methods. Furthermore, we can visualize the generative *assumptions* of each of these methods through the induced covariance functions, which are much more intuitive, or “interpretable,” than examining the weight vectors or the frequency response. We also place learning the parameters within the GP framework, which often classically are done by many ad hoc methods. In particular, selecting the model order is often done by significance testing, ignoring multiple comparison and other issues. If we use a GPTS covariance function with comparable flexibility to the equivalent GPTS covariance function in the classical linear models we sidestep the issue of order selection. The results of this section generalize those of Murray-Smith and Girard [2001].

AR process Any linear autoregressive predictor can be converted to a GPTS. Using (3.102) we can convert a GPTS to an autoregressive formulation. Likewise, we show the reverse operation of converting an autoregressive process to its effective covariance matrix. Recall from the Cholesky space view of a GP (3.93):

$$\mathbf{y} = \mathbf{L}\boldsymbol{\epsilon}_{1:T}\,, \quad \boldsymbol{\epsilon}_{1:T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\,, \quad \mathbf{L} \in \mathbb{L}^T\,. \tag{3.133}$$

Note the similarity to an AR process. The AR process is defined by an $\mathbf{A} \in \mathbb{L}^T$ matrix such that

$$p(y_t|\mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{A}(t, 1{:}t-1)\mathbf{y}_{1:t-1}, \mathbf{A}(t,t)^2)\,. \tag{3.134}$$

The row $\mathbf{A}(t, 1{:}t-1)$ is equivalent to the $\mathbf{E}\boldsymbol{\alpha}$ vector for predicting y_t from (3.102) and $\mathbf{A}(t,t)$ is $\sqrt{\sigma_\star^2}$ from (3.107). We can convert from $\mathbf{A}$ to $\mathbf{L}$ by

$$\begin{aligned}
y_t &\overset{(3.133)}{=} \mathbf{L}(t, 1{:}t)\boldsymbol{\epsilon}_{1:t} && (3.135)\\
&\overset{(3.134)}{=} \mathbf{A}(t, 1{:}t-1)\mathbf{y}_{1:t-1} + \mathbf{A}(t,t)\boldsymbol{\epsilon}_t && (3.136)\\
&\overset{(3.133)}{=} \mathbf{A}(t, 1{:}t-1)\mathbf{L}(1{:}t-1, 1{:}t-1)\boldsymbol{\epsilon}_{1:t-1} + \mathbf{A}(t,t)\epsilon_t && (3.137)\\
\implies \mathbf{L}(t, 1{:}t) &= \begin{bmatrix}\mathbf{A}(t, 1{:}t-1)\mathbf{L}(1{:}t-1, 1{:}t-1) & \mathbf{A}(t,t)\end{bmatrix}. && (3.138)
\end{aligned}$$

Therefore, there is a recursive relation to find the Cholesky factorization of the covariance matrix: $\mathbf{K} = \mathbf{L}\mathbf{L}^\top$.

MA process Converting between a GP and an MA(q) process is even easier:

$$y_t = \mathbf{M}(t, 1{:}t)\boldsymbol{\epsilon}_{1:t}\,, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\,, \tag{3.139}$$

which is already in the form of a GP in the Cholesky view: $\mathbf{L} = \mathbf{M} \implies \mathbf{K} = \mathbf{M}\mathbf{M}^\top$.

We have introduced the linear AR process in a general form. The usual definition for an AR process is:

$$y_t = \mu + \sigma\epsilon_t + \sum_{i=1}^{p} w_i y_{t-i}\,, \quad \epsilon_t \sim \mathcal{N}(0,1)\,. \tag{3.140}$$

Without loss of generality we assume the drift $\mu = 0$ since it corresponds to a mean function, which can be added to y_t. Equivalently we can make the variance on ϵ unity and put a weight of σ on the noise term. Therefore, in a typical AR(p) process:

$$\mathbf{A}(t, t-p{:}t-1) = \mathbf{E}\mathbf{w}^\top\,, \quad \mathbf{A}(t,t) = \sigma \in \mathbb{R}^+\,, \tag{3.141}$$

for all $t > p$.

Likewise, for an MA(q) process, the weights are typically a constant vector. The usual MA[1] is:

$$y_t = \mu + \sigma\epsilon_t + \sum_{i=1}^{q} w_i \epsilon_{t-i}\,, \quad \epsilon_t \sim \mathcal{N}(0,1)\,. \tag{3.142}$$

Again we assume $\mu = 0$ and reformulate as:

$$y_t = \sum_{i=0}^{q} \tilde{w}_i \epsilon_{t-i}\,, \quad \tilde{\mathbf{w}}^\top := \begin{bmatrix} \sigma & \mathbf{w}^\top \end{bmatrix}\,, \quad \epsilon_t \sim \mathcal{N}(0,1) \tag{3.143}$$

$$\implies y_t = \tilde{\mathbf{w}}^\top \mathbf{E}\boldsymbol{\epsilon}_{t-q:t} \tag{3.144}$$

$$\implies \mathbf{M}(t, t-q{:}t) = \mathbf{E}\tilde{\mathbf{w}}^\top\,, \tag{3.145}$$

for all $t > q$.

Edge effects We have shown how to convert standard AR and MA equations and embed them in the covariance matrix of GPTS. However, the issue of how to initialize AR and MA models is typically ignored. How do we predict in AR and MA if $t \leq p$ or $t \leq q$? We must set initial conditions, often we assume $y_t = 0$ for $t \leq 0$ in the AR case; or $\epsilon_t = 0$ for $t \leq 0$ for the MA case. By

[1] Often the ϵ_t is not multiplied by σ, but itself has variance σ^2. In order for the MA to be equivalent to the Cholesky space formulation we use the $\sigma\epsilon_t$.

putting these systems in the GPTS framework we get a natural procedure for handling missing data, both for $t \leq 0$ and general missing observations. By converting our representation to a stationary covariance function $k \in \mathbb{R} \to \mathbb{R}$ we can naturally account for the effects of non-uniform sampling and therefore extend these methods to continuous time.

The ACFs (1.7) for stationary AR and MA systems are typically not computed in the context of edge effects. Meaning, if we compute the equivalent covariance matrix for an AR system, the covariance matrix will only become Toeplitz with respect to the AR ACF when $p \ll t$. Likewise, we take a Toeplitz covariance matrix $\mathbf{K}$ with a stationary ACF the $\mathbf{A}$ matrix will only stabilize to an AR system with constant weights after $p \ll t$.

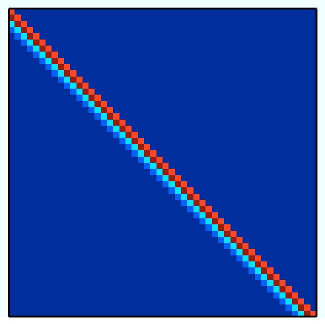
This is illustrated in Figures 3.5 and 3.6. We use a AR(3) process assuming $y_t = 0$ for $t \leq 0$ and show the GP covariance and MA process for the exact same distribution. It takes ≈ 20 time steps for the GP covariance to lose these edge effects and become Toeplitz (stationary covariance). In the next row of Figure 3.5 we show what happens when we force the GP covariance to be Toeplitz and find the equivalent AR and MA processes. The AR process has perturbed weights for the first few steps and the longest lag column of the MA process is non-Toeplitz. We show the weights/covariance function for each model once edge effects become negligible in Figure 3.6. Despite having more parameters the AR process basically induces a Laplace covariance function (two hyper-parameters) in the GP. The MA process still accounts for noise at large lags despite the small order of the underlying AR model.

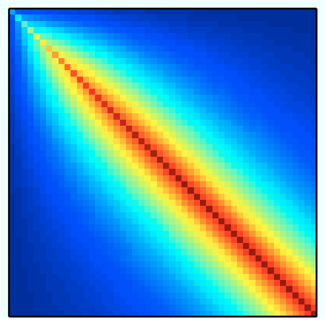
ARMA process In a standard ARMA(p,q) model

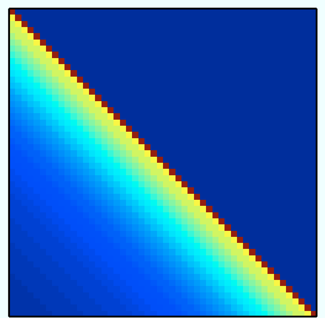
$$y_t = \mu + \sum_{i=1}^{p} w_i^{AR} y_{t-i} + \sigma \epsilon_t + \sum_{i=1}^{q} w_i^{MA} \epsilon_{t-i}\,, \quad \epsilon_t \sim \mathcal{N}(0,1)\,. \tag{3.146}$$

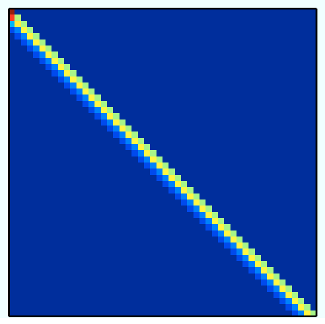
We generalize this to

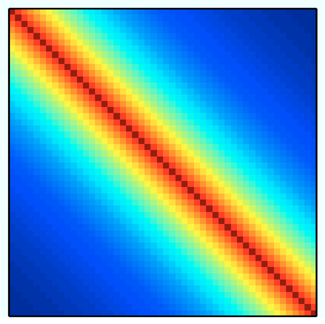
$$y_t = \mu + \mathbf{A}(t, 1{:}t-1)\mathbf{y}_{1:t-1} + \mathbf{M}(t, 1{:}t)\boldsymbol{\epsilon}_{1:t}\,, \quad \epsilon_t \sim \mathcal{N}(0,1)\,. \tag{3.147}$$

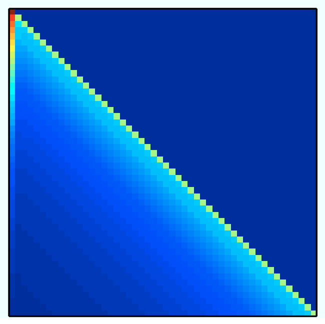

(a) **A** matrix of AR

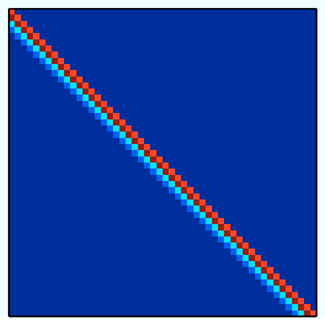

(b) **K** of GP

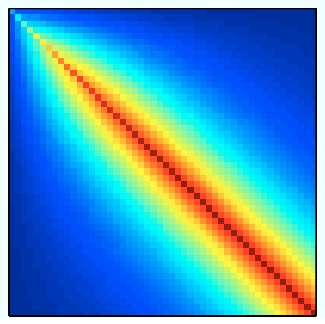

(c) **M** matrix of MA


(d) **A** matrix of AR


(e) **K** of GP


(f) **M** matrix of MA


(g) **A** matrix of AR


(h) **K** of GP

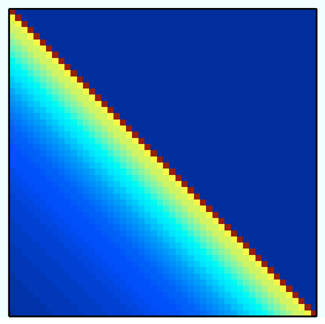
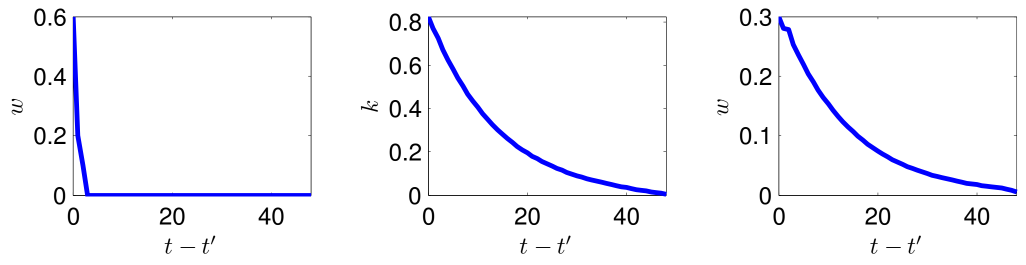

(i) **M** matrix of MA

Figure 3.5: Illustration of the relationship between AR, GP, and MA models. The first column represents the **A** matrix of an AR formulation of a process, the second column represents the **K** matrix of a GP, and the third column represents the **M** matrix of a MA process. On the first row we start with the matrix representation of an AR process with a constant weight vector $\mathbf{E}\mathbf{w}^\top = [0.1\ 0.2\ 0.6\ 0.5]$. We show the GP covariance matrix and MA matrix for the *exact* same distribution on the time series. On the second row we make GP covariance function stationary by making **K** Toeplitz. The other columns show the AR and MA matrices. On the third row we make the process a constant MA process, and show the exact AR and GP equivalents.



(a) AR weights (b) GP covariance (c) MA weights

Figure 3.6: We show the final row of the matrix for the AR ($\mathbf{A}$), GP ($\mathbf{K}$), and MA ($\mathbf{M}$) processes. These correspond to the predictive equations of each process once the edge effects are negligible.

We get the equivalent covariance matrix for the ARMA similar to how we did for (3.138):

$$\begin{aligned} y_t &\overset{(3.133)}{=} \mathbf{L}(t, 1{:}t)\boldsymbol{\epsilon}_{1:t} &(3.148)\\ &\overset{(3.134)}{=} \mathbf{A}(t, 1{:}t-1)\mathbf{y}_{1:t-1} + \mathbf{M}(t, 1{:}t)\boldsymbol{\epsilon}_{1:t} &(3.149)\\ &\overset{(3.133)}{=} \mathbf{A}(t, 1{:}t-1)\mathbf{L}(1{:}t-1, 1{:}t-1)\boldsymbol{\epsilon}_{1:t-1}\\ &+ \mathbf{M}(t, 1{:}t-1)\boldsymbol{\epsilon}_{1:t-1} + \mathbf{M}(t,t)\boldsymbol{\epsilon}_t\,. &(3.150)\end{aligned}$$

Solving for $\mathbf{L}(t, 1{:}t-1)$ we get:

$$\begin{aligned} \mathbf{L}(t, 1{:}t-1) &= \mathbf{A}(t, 1{:}t-1)\mathbf{L}(1{:}t-1, 1{:}t-1) + \mathbf{M}(t, 1{:}t-1)\,,\\ \mathbf{L}(t,t) &= \mathbf{M}(t,t)\,. &(3.151)\end{aligned}$$

Again, we find the full covariance matrix by: $\mathbf{K} = \mathbf{L}\mathbf{L}^\top$.

Differencing GPs can automatically capture the modeling effect of differencing or summing. Differencing is a linear operation:

$$\mathbf{y}' := \mathbf{D}\mathbf{y}\,, \qquad (3.152)$$

where the $\mathbf{D}$ matrix, see (A.8), acts as a differencing operator. A linear operation (multiplication by $\mathbf{D}$) on a GP results in a GP with a different covariance function.

Recall from the linear transform property (A.36) of a multivariate Gaussian that

$$\mathbf{K} := \operatorname{Cov}[\mathbf{y}] \implies \operatorname{Cov}[\mathbf{D}\mathbf{y}] =: \mathbf{K}' \overset{(A.36)}{=} \mathbf{D}\mathbf{K}\mathbf{D}^\top . \tag{3.153}$$

This gives us a model based approach to selecting between working with the original data and the differences data: We can compare the marginal likelihoods under each covariance function. Furthermore, we can do a covariance sum between the differenced covariance and the original to allow for a hybrid approach. If we apply the cumulative sum operator $\mathbf{D}^{-1}$ to the ARMA covariance matrix we also reduce the autoregressive integrated moving average (ARIMA) to the special case of GPTS.

Return space We can also do the return space transformation in a model based way. Recall that:

$$r_t := \log(y_{t+1}) - \log(y_t) \in \mathbb{R} \tag{3.154}$$

$$\implies r_t \overset{(A.7)}{=} \mathbf{D} \log \mathbf{y} \tag{3.155}$$

$$\implies \mathbf{y} \overset{(A.9)}{=} \exp \mathbf{D}^{-1}\mathbf{r} . \tag{3.156}$$

Therefore, if we put a GP on $\mathbf{r} := [r_1, \ldots, r_T]$ then we can apply $\mathbf{D}^{-1}$ to get a covariance of $\mathbf{D}^{-1}\mathbf{K}(\mathbf{D}^{-1})^\top$ on the cumulative returns. The exp represents a warping function that has been used in the context of GPs through *warped Gaussian processes* [Snelson et al., 2004; Wilson and Ghahramani, 2010]. The warped GP framework allows us to generalize the return space view to functions other than exp in $\mathbb{R} \to \mathbb{R}^+$.

Summary The GPTS is a general framework for time series modeling since so many other methods can be constructed as special cases. The reverse process of casting a GPTS as a simple method has computational benefits. Toeplitz methods and GPK provide two alternative methods of improving the computational benefit of standard GP methods.

3.5 The Sub-evidence $\star$

In (3.112) and (3.114) we described how to get the evidence for a data set $\mathbf{y}$ under a Gaussian process, with exact computation in $\mathcal{O}(N^3)$ time. We show in this section that we can compute the evidence $p(\mathbf{y}_{1:b})$ for all $1 \leq b \leq N$ for free using the Cholesky sub-sampling property (A.15). Perhaps surprisingly, we can use the Cholesky rank-1 updating methods from Seeger [2008] to also get $p(\mathbf{y}_{a:b})$ for all $1 \leq a \leq b \leq N$ in $\mathcal{O}(N^3)$ time; we call this quantity the *sub-evidence*. We can find the evidence of a GP with all starting and end points of the data for the same big-O cost as finding the evidence for the entire data set. Naive computation would be $\mathcal{O}(N^5)$.

The expensive component in computing the evidence of a GP is finding the Cholesky factorization of the kernel matrix $\mathbf{K}$. Therefore, if we can compute the Cholesky of $\mathbf{K}(a{:}b, a{:}b)$ for all a and b cheaply then we can find the sub-evidence cheaply. Once we find $\mathbf{K}(a{:}N, a{:}N)$ computation of $\mathbf{K}(a{:}b, a{:}b)$ is trivial by (A.15):

$$w := b - a + 1\,, \tag{3.157}$$

$$\operatorname{chol}(\mathbf{K}(a{:}b, a{:}b)) \overset{(A.15)}{=} \operatorname{chol}(\mathbf{K}(a{:}N, a{:}N))(1{:}w, 1{:}w)\,. \tag{3.158}$$

The simplest way to find the sub-evidence is by working from the end to start. We define the sub-evidence matrix $\mathbf{P} \in \mathbb{L}^T$ where $\mathbf{P}(a, b) := p(\mathbf{y}_{a:b})$. Meaning we start with $a = b = N$ and decrement a down to 1 by applying rank-1 methods with $\mathcal{O}(N^2)$ cost each, leaving us with $\mathcal{O}(N^3)$ total cost. Consider the following setup:

$$\mathbf{K}_{\text{new}} := \begin{bmatrix} \kappa & \mathbf{k}_{\text{new}} \\ \mathbf{k}_{\text{new}}^\top & \mathbf{K} \end{bmatrix}, \quad \mathbf{L}_{\text{new}} := \begin{bmatrix} \ell & \mathbf{0} \\ \mathbf{v} & \mathbf{M} \end{bmatrix}, \tag{3.159}$$

which implies

$$\mathbf{K}_{\text{new}} = \mathbf{L}_{\text{new}} \mathbf{L}_{\text{new}}^\top = \begin{bmatrix} \ell & \mathbf{0} \\ \mathbf{v} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \ell & \mathbf{v}^\top \\ \mathbf{0} & \mathbf{M}^\top \end{bmatrix}. \tag{3.160}$$

We can block-wise multiply out $\mathbf{L}_{\text{new}}\mathbf{L}_{\text{new}}^{\top}$ to get equations:

$$\mathbf{v}\mathbf{v}^{\top} + \mathbf{M}\mathbf{M}^{\top} = \mathbf{K} = \mathbf{L}\mathbf{L}^{\top}\,, \quad \ell^2 = \kappa\,, \quad \ell\mathbf{v}^{\top} = \mathbf{k}_{\text{new}} \tag{3.161}$$

$$\implies \ell = \sqrt{\kappa}\,, \quad \mathbf{v}^{\top} = \mathbf{k}_{\text{new}}/\ell\,, \quad \mathbf{M} = \text{chol}(\mathbf{L} - \mathbf{v}\mathbf{v}^{\top})\,. \tag{3.162}$$

The first two equations for updating $\mathbf{L}$ upon decrementing a are trivial, the downdate[1] from $\mathbf{L}$ to $\mathbf{M}$ requires clever rank-1 methods described in Seeger [2008].[2] The current algorithm is retrospective, it starts with $a = N$, we must know all the data before we start. The sub-evidence recursions are adapted to work in an online fashion by starting at $a = 1$ and incrementing through the data set by reversing the order of the rows and columns in $\mathbf{K}$. Furthermore, if the $\mathcal{O}(N^3)$ total cost becomes prohibitive we can limit the maximum window size $b - a$ by removing the largest rows and columns from $\mathbf{L}$ before applying the rank-1 downdate. The cost is then $\mathcal{O}(Nw^2)$ for a maximum window size w. Furthermore we do not have to evaluate the entire kernel matrix $\mathbf{K}$ giving us total memory complexity $\mathcal{O}(Nw + w^2)$. We summarize the sub-evidence method in Algorithm 4.

We reformulate the evidence expression of (3.112) into a form that is compatible with the sub-sampling property:

$$\boldsymbol{\alpha} := \mathbf{L}\backslash\mathbf{y}\,, \tag{3.163}$$

$$\log p(\mathbf{y}) \overset{(\text{A.14})}{=} -\frac{1}{2}\boldsymbol{\alpha}^{\top}\boldsymbol{\alpha} - \mathbf{1}^{\top}\log\text{diag}(\mathbf{L}) - \frac{N}{2}\log 2\pi\,, \tag{3.164}$$

where we have made use of finding the determinant from the Cholesky (A.14). Solving a linear system with a lower triangular matrix preserves the sub-sampling property (A.17). Therefore, we get the intermediate solutions by doing a cumulative inner product (A.16) and cumulative sum along the diagonal of $\log \mathbf{L}$.

[1] This is a downdate as opposed to an update since we are subtracting $\mathbf{v}\mathbf{v}^{\top}$ rather than adding it.

[2] The downdate is even implemented in the MATLAB built in command `cholupdate`$(\mathbf{L}, \mathbf{v}, '-')$.

Algorithm 4 The GP sub-evidence

```
1: function GP-SUB($\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{y} \in \mathbb{R}^N$)
2:     $\mathbf{L} \leftarrow [\emptyset]$                                ▷ Initialize at empty matrix
3:     $\mathbf{K} \leftarrow k(\mathbf{X}, \mathbf{X})$
4:     $b \leftarrow N$
5:     for $a = N$ down to 1 do
6:         $\ell \leftarrow \sqrt{\mathbf{K}(a, a)}$
7:         $\mathbf{v} \leftarrow \mathbf{K}(a + 1{:}N, a)/\ell$
8:         $\mathbf{M} \leftarrow \texttt{choldowndate}(\mathbf{L}, \mathbf{v})$
9:         $\mathbf{L} \leftarrow \begin{bmatrix} \ell & \mathbf{0} \\ \mathbf{v} & \mathbf{M} \end{bmatrix}$
10:        if size of $\mathbf{L}$ exceeds maximum window size $w$ then
11:            $\mathbf{L} \leftarrow \mathbf{L}(1{:}w, 1{:}w)$
12:            $b \leftarrow a + w - 1$
13:        end if
14:        $\boldsymbol{\alpha} \leftarrow \mathbf{L} \backslash y(a{:}b)$
15:        $\mathbf{t} \leftarrow 1{:}b - a + 1$
16:        $-\log \mathbf{P}(a, a{:}b) \leftarrow \frac{1}{2}\mathbf{D}^{-1}(\boldsymbol{\alpha} \odot \boldsymbol{\alpha}) + \mathbf{D}^{-1} \log \operatorname{diag}(\mathbf{L}) + \frac{1}{2} \log(2\pi)\mathbf{t}$
17:    end for
18:    return $-\log \mathbf{P}$        ▷ Return matrix containing all negative log evidences
19: end function
```

We can also find the sub-evidence with uncertain output scale by:

$$\mathbf{t}_\alpha := \alpha_0 + \frac{1}{2}\mathbf{t}\,, \quad \mathbf{t}_\beta := \beta_0 + \frac{1}{2}\mathbf{D}^{-1}(\boldsymbol{\alpha} \odot \boldsymbol{\alpha})\,, \tag{3.165}$$

$$\begin{aligned} -\log \mathbf{P}(a, a : n) &= \mathbf{t}_\alpha \odot \log(\mathbf{t}_\beta/\beta_0) + \mathbf{D}^{-1} \log \operatorname{diag}(\mathbf{L}) \\ &\quad - \log \Gamma(\mathbf{t}_\alpha) + \log \Gamma(\alpha_0) + \frac{1}{2} \log(2\pi\beta_0)\mathbf{t}\,. \end{aligned} \tag{3.166}$$

The sub-evidence is a useful quantity for adding robustness against change points. The ability to compute the sub-evidence in $\mathcal{O}(N^3)$ time, the same computation cost as a standard GP, may have wider implications. This is a significant improvement over simple rank-1 updates, $\mathcal{O}(N^4)$, and naive calculation $\mathcal{O}(N^5)$.

3.6 Autoregressive Gaussian Process

We show a second approach to time series modeling with GPs, the autoregressive Gaussian process (ARGP). The ARGP is a more general and powerful approach than the GPTS, but it is more computationally tricky than a GPTS. In an ARGP [Quiñonero-Candela et al., 2003] of order p, the past p values $\mathbf{y}_{(p)}$ are taken as the GP input while the output is y_t:

$$y_t = f(\mathbf{y}_{(p)}) + \epsilon_t\,, \quad f \sim \mathcal{GP}(0,k)\,, \quad \epsilon_t \sim \mathcal{N}(0,\sigma^2)\,. \tag{3.167}$$

Clearly, this is more general than a GPTS because we showed in Section 3.4 that we could convert a GPTS to an AR formulation with a linear relation. The ARGP has this same form, but with a nonlinear relationship.

Relationship to GPTS $\star$ The GPTS has some more elegant properties than the ARGP: Firstly, the GPTS is time reversible while the ARGP generally is not. The marginal likelihood of the data $p(\mathbf{y})$ is the same as $p(\mathbf{Ey})$ in a GPTS, but not in an ARGP. Therefore, ARGP is placing prior information on the direction dynamics evolve. In many physical systems, such as Newtonian mechanics, the laws are reversible [Tolman, 1938, Ch. 5] [Hutchison, 1993], meaning ARGP might be incompatible with our prior assumptions. Second, the prior induced by the ARGP is different depending on the sampling rate. We introduce an augmented time series $\tilde{\mathbf{y}}$ with an extra observation in between each of those in $\mathbf{y}$, but the extra observations are treated as missing. The marginal likelihood of $p(\mathbf{y}_{1:T})$ is the same as $p(\tilde{\mathbf{y}}_{1:2:T})$ in a GPTS, but not in an ARGP. Therefore, if we double the sampling rate of the time series and apply an ARGP we induce a different prior over the process, which might not be desirable if the sampling rate was chosen arbitrarily. Third, the GPTS can elegantly handle continuous time while the ARGP cannot. Arbitrary spacing of observations in time can easily be accounted for with a GPTS; there is not a clear tractable method to do so in an ARGP. Nonetheless, the ARGP is still useful since it models more complex dynamics than the GPTS.

Correcting the ARGP Two properties of the GPTS could be instilled through sufficient model selection. We could perform model selection on the direction of time. We could also perform model selection on the sampling rate by trying models with differing numbers of hidden nodes between observations. Additionally, continuous time could be handled by placing a GP directly on the differential equation for the system. These proposals are listed in order of increasing computational difficulties.

Efficient implementation As in the GPTS, re-learning the GP from scratch every time step requires $\mathcal{O}(T^3)$ operations, which implies $\mathcal{O}(T^4)$ time for the entire time series. In the GPTS case, we dramatically improved the efficiency by using either Toeplitz methods (Section 3.4.1) or GPK (Section 3.4.2). In the GPTS, the input is the time index t (often a controlled quantity) where as in ARGP it is y_t, a random quantity. Therefore, the covariance matrix is not Toeplitz, prohibiting the use of the Yule-Walker method. The inputs do not increase in time and are not one-dimensional, unless $p = 1$, which hinders our ability to use GPK. When adding a new data point in a kernel machine it is sometimes recommended to update the inverse of the covariance matrix $\mathbf{K}^{-1}$ using *rank 1 updates* [Schölkopf and Smola, 2001, Ch. 10], which requires $\mathcal{O}(T^2)$ operations as opposed to $\mathcal{O}(T^3)$ for complete recomputation. However, doing rank-1 updating of the inverse of the covariance is highly numerically unstable.

Therefore, we work with rank-1 methods on the Cholesky instead. We could use the methods of Section 3.5 to get rolling predictions from an ARGP. However, if we are only increasing the end point of the time series, i.e. we fix $a = 1$ and only increase b in (3.158), the Cholesky updates methods in Section 3.5 are "overkill." Instead, we use the Cholesky updates described in Nguyen-Tuong et al. [2010]. These are faster than `choldowndate` by a constant factor. The update equations are:

$$\mathbf{v} := \mathbf{L}_{t-1} \backslash \mathbf{K}_\star\,, \quad \sigma_\star^2 \overset{(3.16)}{=} \mathbf{K}_{\star\star} - \mathbf{v}^\top \mathbf{v}\,, \tag{3.168}$$

$$\mathbf{K}_t := \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{K}_\star \\ \mathbf{K}_\star^\top & \mathbf{K}_{\star\star} \end{bmatrix} \implies \mathbf{L}_t = \begin{bmatrix} \mathbf{L}_{t-1} & \mathbf{0} \\ \mathbf{v}^\top & \sqrt{\sigma_\star^2} \end{bmatrix}\,. \tag{3.169}$$

We use the base case of $\mathbf{K}_0 = \mathbf{L}_0 = [\emptyset]$ being the empty matrix. We can verify these recursions by multiplying out $\mathbf{L}_t\mathbf{L}_t^\top$ and showing it is equivalent to $\mathbf{K}_t$. The updating occurs almost for free as we must compute $\mathbf{v}$ and $\sigma_\star^2$ when making rolling predictions anyway. Once $\mathbf{L}_t$ is obtained we can predict $\mathbf{y}_t$ using (3.15) or (3.38) for uncertain output scale.

Using (3.169) we can now justify the trick in (3.28) to find the SSE:

$$\mathbf{L}_t \backslash \mathbf{y}_{1:t} \overset{(3.28)}{=} \mathbf{s}_{1:t} \tag{3.170}$$

$$\implies \begin{bmatrix} \mathbf{L}_{t-1} & \mathbf{0} \\ \mathbf{v}^\top & \sqrt{\sigma_\star^2} \end{bmatrix} \Big\backslash \begin{bmatrix} \mathbf{y}_{1:t-1} \\ y_t \end{bmatrix} \overset{(3.169)}{=} \begin{bmatrix} \mathbf{s}_{1:t-1} \\ s_t \end{bmatrix} \tag{3.171}$$

$$\implies \begin{bmatrix} \mathbf{y}_{1:t-1} \\ y_t \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{t-1} & \mathbf{0} \\ \mathbf{v}^\top & \sqrt{\sigma_\star^2} \end{bmatrix} \begin{bmatrix} \mathbf{s}_{1:t-1} \\ s_t \end{bmatrix} . \tag{3.172}$$

If we multiply out the matrices to solve for y_t we get:

$$y_t = \mathbf{v}^\top \mathbf{s}_{1:t-1} + \sqrt{\sigma_\star^2} s_t \,, \tag{3.173}$$

$$\mathbf{v}^\top \mathbf{s}_{1:t-1} \overset{(3.168)}{=} \mathbf{K}_\star^\top (\mathbf{L}_{t-1}^{-1})^\top \mathbf{L}_{t-1}^{-1} \mathbf{y}_{1:t-1} \tag{3.174}$$

$$= \mathbf{K}_\star^\top \mathbf{K}_{t-1}^{-1} \mathbf{y}_{1:t-1} \overset{(3.15)}{=} \mu_\star \,, \tag{3.175}$$

substituting back in we get

$$s_t = \frac{y_t - \mu_\star}{\sqrt{\sigma_\star^2}} \,. \tag{3.176}$$

Therefore, by back-substituting the Cholesky factorization $\mathbf{L}_T$ into the time series $\mathbf{y}$ we get the standardized error $\mathbf{s}_{1:T}$ of the one-step-ahead predictives.

3.6.1 External Inputs and Efficiency

We may be in a situation where we want to use external inputs $\mathbf{z}_t \in \mathbb{R}^E$, or co-variates, in the model. For instance, when building a predictive model of snowfall we might want to use temperature as an input. One option is model these two time series jointly. However, it may be more direct to build a conditional model, as is the case in discriminative generative distinction in iid data. If the external inputs are a quantity that might be controlled, as is the case in feedback control systems, it is more appropriate to build a conditional model.

In the GPTS setup, the conditional model would be:

$$y_t = f(t, \mathbf{z}_t) + \epsilon_t \,, \quad f \sim \mathcal{GP}(0, k) \,, \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2) \,. \tag{3.177}$$

Using an ARGP, we would have:

$$y_t = f(\mathbf{y}_{(p)}, \mathbf{z}_t) + \epsilon_t \,, \quad f \sim \mathcal{GP}(0, k) \,, \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2) \,. \tag{3.178}$$

Like in ARGP, the GPTS with external inputs no longer has a Toeplitz covariance matrix in general. If the external inputs are sampled on an appropriate grid they can still have a Toeplitz structure [Storkey, 1999]. If the external inputs are not on an appropriate grid, we must use either the same inference methods used for ARGP or use a linear dependence on the inputs. We can apply a set of basis functions ϕ for the dependence on $\mathbf{z}_t$:

$$y_t = f(t) + \mathbf{w}^\top \phi(\mathbf{z}_t) + \epsilon_t \,, \quad f \sim \mathcal{GP}(0, k) \,, \tag{3.179}$$

$$\mathbf{w} \sim \mathcal{N}(0, \mathbf{S}) \,, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_n^2) \,, \tag{3.180}$$

where $\mathbf{S} \in \mathbb{S}^E$ is the prior covariance on the weights $\mathbf{w} \in \mathbb{R}^E$. If we use the ARMA covariance for k we get the autoregressive moving average with exogenous inputs (ARMAX) model.

3.6.2 Multivariate Gaussian Process Time Series

While GPs typically are used for modeling functions with a univariate output, they can also be extended to multiple outputs. In the kriging community this is known as *cokriging*. This is considered the joint approach of modeling $\mathbf{y}$ and $\mathbf{z}$ together rather than the conditional approach used in Section 3.6.1.

The simplest way to model a multivariate function $f \in \mathbb{R}^D \to \mathbb{R}^E$ with a GP is to use E independent GPs for each output dimension; each output dimension typically gets its own covariance hyper-parameters ξ, although they can be tied together. If we want to allow for correlations between the dimensions we can have M GPs on latent functions g, typically $M = E$, and have a mixing matrix $\mathbf{C} \in \mathbb{R}^{E \times M}$ such that

$$g_i \sim \mathcal{GP}(\mu_i, k_i) \quad \text{for} \quad i \in 1{:}M\,, \tag{3.181}$$

$$f(\mathbf{x}) = \mathbf{C}g(\mathbf{x})\,, \quad \mathbf{y} = f(\mathbf{x}) + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, \mathbf{\Sigma}_n)\,. \tag{3.182}$$

The ith latent GP has mean function μ_i and covariance k_i. Linear operations on latent GPs is as general as we can get while preserving a predictive distribution that is Gaussian [Boyle and Frean, 2005]. If the covariance function on each of the M latent dimensions is the same, the distribution on $\mathbf{Y} \in \mathbb{R}^{N \times E}$ the data can be specified in terms of a matrix normal [Dawid, 1981]. Although (3.182) is the most standard approach to multiple output GPs, we can also represent the output dimension as in input:

$$\mathrm{Cov}\left[y_i^m, y_j^n\right] = k((\mathbf{x}_i, m), (\mathbf{x}_j, n))\,, \tag{3.183}$$

where the covariance function also considers a correlation between the output dimensions m and n on data points i and j respectively. This yields a more general approach.

Both the GPTS and ARGP are directly extendable to the multivariate case using the setup of (3.182). We substitute a time series for $\mathbf{Y}$ and time indices for $\mathbf{x}$ in (3.182) and obtain a multivariate GPTS. In the multivariate GPTS $D = 1$ and E is the dimensionality of the time series. In ARGP of order one, $D = E$ is the dimensionality of the time series. We substitute the time series at $t - 1$ for $\mathbf{x}$ and the time series at t for $\mathbf{y}_t$ in (3.182) to obtain the multivariate ARGP.

GPTS is already quite general given that most standard methods can be constructed as a special case. However, the ARGP extends this even more as the ARGP is arguably more general than the GPTS. Computational optimizations are more difficult than in GPTS, but the rank-1 Cholesky updates and the sub-evidence can be utilized for computational advantages.

3.7 Conclusions

We have provided a brief introduction to GPs, albeit emphasizing aspects not covered elsewhere. For instance, we have provided detailed calculations of how to handle uncertain output scale, sometimes called the “t-process.” We have also covered two different approaches to the autoregressive time series modeling: GPTS and ARGP.

The GPTS encompasses many of the standard linear time series models as special cases. GPTS can be converted to these models for computational advantages. Standard models such as AR can be converted to GPTS to handle effects such as non-uniform sampling and edge effects. Additionally, we can constrain the parameter space in GPTS to handle long range effects, while limiting the number of parameters and therefore the over fitting potential.

The alternative approach, the ARGP, is even more flexible than the GPTS. However, it requires more parameters to model long range effects in its standard formulation. We have developed methodologies to do rolling updates of an ARGP in an online manner using rank-1 updates. Furthermore, we can find the sub-evidence, the evidence of the data at every starting and end point of the data in the same $\mathcal{O}(T^3)$ cost as finding the evidence of the data in its entirety.

Just as many standard time series methods can be cast as GPs, GPs have connections to many other methods. For instance, GPs are limiting cases of neural networks. We have discussed connections to Bayesian linear models, RBF, and the rarely discussed Cholesky space view.

An overall theme of this chapter has been to describe aspects often neglected in an applied setting in a more concrete manner. We have discussed the existence of GPs in a computationally defined way. We have more thoroughly described the connections between GP methods and standard time series methods as well as regression methods more generally.

Chapter 4

State Space Models

Imagine tracking the location of a car based on odometer and GPS sensors, both of which are noisy. Sequential measurements from both sensors are combined to overcome the noise in the system to obtain an accurate estimate of the car's position and velocity, known as the *system state*. Furthermore, suppose the engine temperature is unobserved, it can still be inferred via the nonlinear relationship with acceleration. To exploit this relationship appropriately, inference techniques in nonlinear models are required; they play an important role in many practical applications. Filtering in linear dynamical systems (LDS) and nonlinear dynamical systems (NLDS) is frequently used in many areas, such as signal processing, state estimation, control, and finance/econometric models [Harvey, 1991; Lefebvre et al., 2004; Thrun et al., 2005]. Filtering (inference) aims to estimate the state of a system from a stream of noisy measurements.

LDS and NLDS belong to a class of models known as state space models. A state space model assumes that there exists a sequence of latent states $\mathbf{x}_t$ that evolve over time according to a Markovian process specified by a transition function $f \in \mathbb{R}^M \to \mathbb{R}^M$. The latent states are observed indirectly in $\mathbf{y}_t$ through a measurement function $g \in \mathbb{R}^M \to \mathbb{R}^D$. We consider a more constrained set of state space models than (1.18), but still relatively general, given by

$$\begin{aligned} \mathbf{x}_t &= f(\mathbf{x}_{t-1}) + \boldsymbol{\epsilon}_t\,, \quad \mathbf{x}_t \in \mathbb{R}^M\,, \\ \mathbf{y}_t &= g(\mathbf{x}_t) + \boldsymbol{\nu}_t\,, \quad \mathbf{y}_t \in \mathbb{R}^D\,. \end{aligned} \tag{4.1}$$

Here, the system noise $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ and the measurement noise $\boldsymbol{\nu}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ are both iid Gaussian. We refer to the entire sequence as $\mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_T]$ and $\mathbf{Y} := [\mathbf{y}_1, \dots, \mathbf{y}_T]$. In the LDS case, f and g are linear functions, whereas the NLDS covers the general nonlinear case. For notational clarity, in the LDS case we write

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t\,, \quad \mathbf{x}_t \in \mathbb{R}^M\,, \quad \mathbf{A} \in \mathbb{R}^{M\times M}\,, \\ \mathbf{y}_t &= \mathbf{C}\mathbf{x}_t + \boldsymbol{\nu}_t\,, \quad \mathbf{y}_t \in \mathbb{R}^D\,, \quad \mathbf{C} \in \mathbb{R}^{D\times M}\,. \end{aligned} \tag{4.2}$$

Both the LDS and the NLDS have an initial state distribution $p(\mathbf{x}_1|\emptyset) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ with $\boldsymbol{\mu}_0 \in \mathbb{R}^M$ and $\boldsymbol{\Sigma}_0 \in \mathbb{S}^M$. Therefore, we summarize the parameter space of the LDS as $\Theta = \{\mathbf{A}, \mathbf{C}, \mathbf{Q}, \mathbf{R}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\}$.

When state space modeling is used in control, the equations are typically augmented with a set of *control inputs* $\mathbf{u}$,

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \boldsymbol{\epsilon}_t\,, \quad \mathbf{u}_t \in \mathbb{R}^E\,, \quad \mathbf{B} \in \mathbb{R}^{M\times E}\,, \\ \mathbf{y}_t &= \mathbf{C}\mathbf{x}_t + \mathbf{D}\mathbf{u}_t + \boldsymbol{\nu}_t\,, \quad \mathbf{D} \in \mathbb{R}^{D\times E}\,. \end{aligned} \tag{4.3}$$

where $\mathbf{D}$ is often assumed to be zero: $\mathbf{D} = \mathbf{0}$. Since none of our problems involve designing or benchmarking feedback control units, we neglect the control input, setting $\mathbf{B} = \mathbf{0}$. However, the methods in this chapter can be augmented to include a control input by treating it equivalently to the previous latent state in the system equation, albeit with no state uncertainty.

When high dimensional systems, i.e. large observation dimensionality D, can accurately be modeled using a small latent state dimensionality M, state space approaches have a natural "edge" over their autoregressive counterparts. The smaller latent space naturally constrains the number of effective parameters to be learned.

The outline of this chapter is as follows: We begin by surveying the standard approaches to filtering and learning in state space models. We include a discussion of the pitfalls and challenges in these methods. Second, we use GPs for learning the system dynamics of a state space model in a nonparametric fashion. Third, we discuss a separate GP-based learning procedure that helps avoid some of the pitfalls in a standard approach, the unscented Kalman filter. Finally, we evaluate these methodologies on nonlinear filtering and learning tasks.

4.1 Existing Work

Kalman filtering [Kalman, 1960] corresponds to exact (and fast) inference in LDS, which only models a limited set of phenomena. For the last few decades, there has been interest in NLDS for more general applicability. In the state space formulation, nonlinear systems do not generally yield analytically tractable algorithms and some form of approximation must be used.

The most widely used approximations for filtering in NLDS are the extended Kalman filter (EKF) [Maybeck, 1979] and the unscented Kalman filter (UKF) [Julier and Uhlmann, 1997]. There are also Monte Carlo based approximations known as particle filters [del Moral, 1996]. The EKF linearizes f and g at the current estimate of $\mathbf{x}_t$. This can be considered a nonstationary (local) linear approximation of a nonlinear system. The UKF propagates several representative estimates of $\mathbf{x}_t$ through f and g and reconstructs a Gaussian predictive distribution assuming the propagated values came from a linear system. The locations of the representative estimates of $\mathbf{x}_t$ are known as the *sigma points*. Many heuristics have been developed to help set the sigma point locations [Julier and Uhlmann, 2004].

In this section we integrate Kalman filtering and its approximations the EKF and UKF into a common framework for filtering methods. These filters use the same algorithm as the Kalman filter, but use different methods of approximating the moments, which are intractable to calculate for general nonlinear f and g. We discuss subtleties of implementing these filtering methods. These approximations lead to GP filtering methods in the next section, which compute moments exactly when f and g are modeled by GPs, unlike the EKF and UKF.

4.1.1 Kalman Filtering

Inference in state space models follows a common pattern, corresponding to message passing algorithms [Ghahramani and Roweis, 1999; Pearl, 1986] in models whose graphical structure have the same structure as an LDS. Filtering methods are typically described as having two steps: the time update and the measurement update. They iterate in a predictor-corrector setup. We further break the measurement update into the measurement prediction and the Bayes' update to get a three step process. In the time update we find $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, d\mathbf{x}_{t-1}\,, \tag{4.4}$$

using $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$. In the measurement prediction step we predict the observed space, $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$ using $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$:

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})\, d\mathbf{x}_t\,. \tag{4.5}$$

Finally, in the Bayes' update we use Bayes' rule to find $p(\mathbf{x}_t|\mathbf{y}_t)$ using information from how good (or bad) the prediction in the measurement prediction step is:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \propto p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})\,. \tag{4.6}$$

In the linear Gaussian case all of these equations can be done analytically using matrix multiplications, which is the standard Kalman filter.

Due to the Markovian properties present in all state space models from (4.1), we can do exact filtering with Algorithm 5. Except in small states spaces, the probability updates here may require high dimensional integrals. In LDS case, tractability is obtained since we only need to propagate the moments of a Gaussian efficiently updated with the transformation and conditioning properties of Gaussians, which is shown in Algorithm 6. All deterministic filters in common usage for approximate inference in nonlinear systems, as shown in Deisenroth and Ohlsson [2011], can also be cast in the form of Algorithm 6. The subroutines in Algorithm 7 only apply to the standard Kalman filter. In nonlinear methods, such as the EKF and UKF, they merely approximate functions `predict-`x and `predict-`y. In this chapter we cover the GP assumed density filter (GP-ADF), which also fits into this framework. In GP-ADF, we compute the exact means and covariances of `predict-`x and `predict-`y by modeling the latent functions f and g with GPs. By contrast, the EKF and UKF return approximate moments that are only exact when f and g are linear.

Algorithm 5 General form of exact inference in state space models

```
1: $p(\mathbf{x}_1|\emptyset) \leftarrow (\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$    ▷ Initialize the recursion
2: for $t = 1$ to $T$ do
3:     Measurement prediction step: $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$ using $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$    ▷ Using (4.5)
4:     Bayes' update: $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ using $\mathbf{y}_t$    ▷ Using (4.6)
5:     Time update: find $p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t})$ using $p(\mathbf{x}_t|\mathbf{y}_{1:t})$    ▷ Using (4.4)
6: end for
```

Algorithm 6 General deterministic filtering setup

```
1: function FILTER($\mathbf{Y} \in \mathbb{R}^{T \times D}, \theta, \boldsymbol{\mu}_0 \in \mathbb{R}^D, \boldsymbol{\Sigma}_0 \in \mathbb{S}^D$)
2:     Use parameters $\theta$ to configure predict-$y$ and predict-$x$
3:     $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow (\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$    ▷ Initialize state mean $\boldsymbol{\mu}_x$ and covariance $\boldsymbol{\Sigma}_x$
4:     for $t = 1$ to $T$ do
5:             ▷ Find $\mathbb{E}[\mathbf{y}_t|\mathbf{y}_{1:t-1}]$, $\mathrm{Cov}[\mathbf{y}_t|\mathbf{y}_{1:t-1}]$, and $\mathrm{Cov}[\mathbf{x}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1}]$
6:         $(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy}) \leftarrow$ predict-$y(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$
7:             ▷ Find $\mathbb{E}[\mathbf{x}_t|\mathbf{y}_{1:t}]$ and $\mathrm{Cov}[\mathbf{x}_t|\mathbf{y}_{1:t}]$
8:         $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow$ filter-update$(\mathbf{y}_t, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x, \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy})$
9:         $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow$ predict-$x(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ ▷ Find $\mathbb{E}[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}]$ and $\mathrm{Cov}[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}]$
10:    end for
11:    return Value of $\boldsymbol{\mu}_y$ and $\boldsymbol{\Sigma}_y$, or $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$, for each iteration
12: end function
13: function FILTER-UPDATE($\mathbf{y}, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x, \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy}$)
14:    $\boldsymbol{\iota} \leftarrow \mathbf{y} - \boldsymbol{\mu}_y$    ▷ Find the innovation
15:    $\mathbf{K} \leftarrow \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_y^{-1}$    ▷ Find the Kalman gain
16:    $\boldsymbol{\mu}_x \leftarrow \boldsymbol{\mu}_x + \mathbf{K}\boldsymbol{\iota}$
17:    $\boldsymbol{\Sigma}_x \leftarrow \boldsymbol{\Sigma}_x - \mathbf{K}\boldsymbol{\Sigma}_y\mathbf{K}^\top$
18:    return $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$
19: end function
```

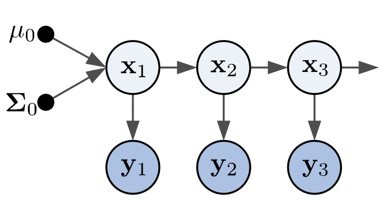
Implementation details The updates of the covariance matrices in Algorithm 6 can become asymmetric due to numerical imprecision. One approach to this obstacle is to symmetrize the covariance matrices after each update, i.e. $\boldsymbol{\Sigma}_x \leftarrow \frac{1}{2}(\boldsymbol{\Sigma}_x + \boldsymbol{\Sigma}_x^\top)$. There can still be numerical problems whereby parameters, especially the system noise $\mathbf{Q}$, of the filter are ill-conditioned and result in non-positive definite covariance matrices $\boldsymbol{\Sigma}_x$. The most robust solution is to use a

Algorithm 7 Updating in a standard Kalman filter

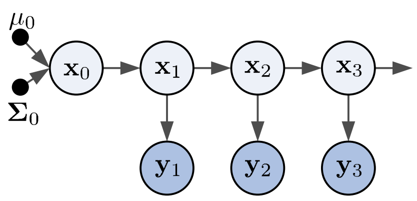
```
 1: function PREDICT-$y$($\boldsymbol{\mu}_x \in \mathbb{R}^D, \boldsymbol{\Sigma}_x \in \mathbb{S}^D$)    ▷ Kalman filter only
 2:     $\boldsymbol{\mu}_y \leftarrow \mathbf{C}\boldsymbol{\mu}_x$
 3:     $\boldsymbol{\Sigma}_y \leftarrow \mathbf{C}\boldsymbol{\Sigma}_x\mathbf{C}^\top + \mathbf{R}$
 4:     $\boldsymbol{\Sigma}_{xy} \leftarrow \boldsymbol{\Sigma}_x\mathbf{C}^\top$
 5:     return $(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy})$
 6: end function
 7: function PREDICT-$x$($\boldsymbol{\mu}_x \in \mathbb{R}^D, \boldsymbol{\Sigma}_x \in \mathbb{S}^D$)    ▷ Kalman filter only
 8:     $\boldsymbol{\mu}_x \leftarrow \mathbf{A}\boldsymbol{\mu}_x$
 9:     $\boldsymbol{\Sigma}_x \leftarrow \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^\top + \mathbf{Q}$
10:     return $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$
11: end function
```

square-root (SR) filter [van der Merwe and Wan, 2001], which propagates the Cholesky factorization of the covariance matrices instead.

Initialization convention An inconsistency in the literature has developed for initializing a state space model. In this thesis, we adopt the convention that our first observation is $\mathbf{y}_1$ and $\mathbf{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. However, it is sometimes adopted that $\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ followed by a transition to $\mathbf{x}_1$ with the first observation at $\mathbf{y}_1$. We take the view that this introduces a superfluous node into the graphical model that can be integrated out and ignored. The motivation for using the $\mathbf{x}_0$ is that the filtering recursions are done in the order `predict-`x, `predict-`y, `filter-update`, which is perhaps more logical than the `predict-`y, `filter-update`, `predict-`x order adopted in this thesis. We compare the conventions visually in their respective graphical models in Figure 4.1.



(a) Our convention



(b) Other convention

Figure 4.1: We contrast the two initialization conventions. In our convention $\mathbf{x}_0$ is integrated out, while in the other convention there is an extra time update.

Learning in a Kalman filter Model based learning of an LDS can be done using the EM algorithm as introduced by Shumway and Stoffer [1982]. Although we could directly apply a gradient based method to the marginal likelihood, the linearities in the model allow for analytic EM updates. The two approaches are related as EM corresponds to coordinate descent in the marginal likelihood [Neal and Hinton, 1998].

To apply EM, we must smooth the state distribution using a Kalman smoother, known as *forward-backward* or Rauch-Tung-Striebel (RTS) smoothing, for the E-step [Rauch et al., 1965]. In the M-step, we optimize the expected complete likelihood $\mathbb{E}_{\mathbf{X}}[p(\mathbf{X}, \mathbf{Y}|\Theta)]$ of the observed variables and the latent states with respect to the parameters Θ.

In the linear case, once the E-step is complete the optimization in the M-step can be computed analytically. Additionally, coordinate descent has advantages over gradient descent in certain circumstances [Salakhutdinov et al., 2003]. In many of the nonlinear extensions considered in this chapter, there are no analytic EM updates, negating any of the computational benefits of using an EM approach. The EM advantage of not needing derivatives does not apply in the case of nonlinear systems, where the M-step itself may require gradient based optimization methods.

Linear systems allow for SVD based methods for learning, which are commonly used in many communities. There is no clear and principled way to extend these methods to nonlinear systems.

Latent dimensionality Before learning the LDS parameters Θ we must select a latent dimensionality M. We cannot merely try each M and select the dimensionality with the best (largest) marginal likelihood, known as the evidence, after EM training. Since each latent dimension forms a superset of models of the lower dimension the MLE solution of dimension $M + 1$ will always be yield a higher likelihood than the MLE solution for dimension M. Ever larger latent dimensionalities will be favored, eventually leading to over-fitting once the number of parameters becomes too large. The Bayesian solution to this is to place priors on the parameters Θ and compare different dimensionality by the marginal likelihood:

$$p(M|\mathbf{Y}) \propto p(\mathbf{Y}|M)P(M) = P(M)\int p(\mathbf{Y}|\Theta, M)d\Theta\,, \tag{4.7}$$

where $P(M)$ is the prior on the latent dimensionality. The marginal likelihood automatically introduces a complexity penalty against overly flexible models, or more parameters roughly speaking. This gives the marginal likelihood an automatic *Occam's razor* effect [Rasmussen and Ghahramani, 2001]. A good intuition for this effect is given in MacKay [2003, Ch. 28].

The marginal likelihood $p(\mathbf{Y}|M)$ in this and many other cases is notoriously difficult to compute. MCMC estimates of the evidence typically have unacceptably large variance. However, there have been attempts at advanced methods to compute the evidence more accurately [Murray, 2007, Ch. 4]. We could get samples from the posterior on the parameters Θ across different dimensions using reversible jump MCMC (RJ-MCMC) [Green, 1995]. Bounds to the marginal likelihood in an LDS can also be found by variational methods [Ghahramani and Beal, 2001]. Both these methodologies are beyond the typical complexity encountered for an LDS. Indeed one of the advantages of the Gaussian process time series (GPTS) methods, as opposed to LDS, is that we need not do model selection over parameter spaces of varying dimension.

There are simple evidence approximations we can apply to the LDS. A simple approach for model selection is to approximate the marginal likelihood using the Bayesian information criterion (BIC) [Schwarz, 1978]:

$$-2\log p(\mathbf{Y}|M) \approx \text{BIC} = -2\log p(\mathbf{Y}|\widehat{\Theta}, M) + |\Theta|\log N\,, \tag{4.8}$$

which poses no extra computational burden over finding the MLE solution. The BIC approximately introduces an Occam's penalty through the parameter counting penalty. A similar and popular criterion is the Akaike information criterion (AIC) [Akaike, 1974].

In order to apply the BIC, we must count the parameters in an LDS. This is not as trivial as it may seem. The nominal cardinality of the parameters is

$$|\Theta| = M^2 + M^2 + DM + D^2 + M + M^2 = 3M^2 + DM + D^2\,. \tag{4.9}$$

However, the effective degrees of freedom is lower since the latent space is unidentifiable and can always be transformed so that $\mathbf{A}$ is diagonal. Therefore, the matrix $\mathbf{A}$ only has M degrees of freedom. Additionally, a positive definite covariance matrix ($\mathbb{S}^D$) has $D(D+1)/2$ degrees of freedom since it is bijective through the matrix logarithm to the symmetric real matrices. The initial $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ will only have a small effect on the solution and therefore should not be included in the final parameter count. The fair parameter count is

$$|\Theta| = (3M + 2DM + M^2 + D^2 + D)/2\,. \tag{4.10}$$

This can now be used in an AIC/BIC procedure.

4.1.2 Extended Kalman Filtering

The EKF explicitly linearizes f and g at the point $\mathbb{E}\left[\mathbf{x}_t|\mathbf{y}_{1:t-1}\right]$ at each step. We substitute $\mathbf{A}$ and $\mathbf{C}$ for the Jacobian matrix of f and g evaluated at $\boldsymbol{\mu}_t$. Therefore, the EKF requires knowledge of the derivatives of f and g. In `predict-`y and `predict-`x we substitute

$$\mathbf{C}_t \approx \left.\frac{dg(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbb{E}[\mathbf{x}_t]}\,, \quad \mathbf{A}_t \approx \left.\frac{df(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbb{E}[\mathbf{x}_{t-1}]} \tag{4.11}$$

for $\mathbf{C}$ and $\mathbf{A}$, respectively. This approximation can be quite good if the function is close to linear. However, if the covariance of $\mathbf{x}$ is large enough to enclose nonlinearities in f or g then the approximation can fail, as shown in Figure 4.2. Gradients are a local property and can be completely unrepresentative of the function. The EKF is attractive in that it has no free parameters that must be set.

4.1.3 Unscented Kalman Filtering

The UKF uses the whole distribution on $\mathbf{x}_t \in \mathbb{R}^D$, not only the mean, to place sigma points and implicitly linearize the dynamics, which we call the *unscented transform* (UT) [Julier et al., 1995]. In one dimension the sigma points roughly correspond to the mean and α-standard deviation points; the UKF generalizes this idea to higher dimensions. The exact placement of sigma points depends on the unitless parameters $\{\alpha, \beta, \kappa\} \in \mathbb{R}^+$ through

$$\mathcal{X}^0 := \boldsymbol{\mu}\,, \quad \mathcal{X}^i := \boldsymbol{\mu} \pm (\sqrt{(D+\lambda)\boldsymbol{\Sigma}})_i\,, \tag{4.12}$$

$$\lambda := \alpha^2(D+\kappa) - D\,, \tag{4.13}$$

where $\sqrt{\cdot}_i$ refers to the ith row of the Cholesky factorization.[1] The sigma points have weights assigned by:

$$w_m^0 := \frac{\lambda}{D+\lambda}\,, \quad w_c^0 := \frac{\lambda}{D+\lambda} + (1-\alpha^2+\beta)\,, \tag{4.14}$$

$$w_m^i := w_c^i := \frac{1}{2(D+\lambda)}\,, \tag{4.15}$$

where $\mathbf{w}_m$ is used to reconstruct the predicted mean and $\mathbf{w}_c$ used for the predicted covariance. We interpret the unscented transform as approximating the input distribution by $2D+1$ point masses at $\mathcal{X}$ with weights $\mathbf{w}$. Once the sigma points $\mathcal{X}$ have been calculated the filter accesses f and g as black boxes to find $\mathcal{Y}_t$, either $f(\mathcal{X}_t)$ or $g(\mathcal{X}_t)$ depending on the step. The UKF reconstructs a Gaussian predictive distribution with the mean and covariance determined from $\mathcal{Y}_t$ pretending the dynamics *had* been linear. In other words, the equation used to find the approximating Gaussian is such that the UKF is exact for a linear dynamics model f and observation model g. It does not guarantee the moments will match the moment of the true non-Gaussian distribution. The UKF is a *black box filter* as opposed to the EKF which requires derivatives of f and g.

Both the EKF and the UKF approximate the nonlinear state space as a nonstationary linear system. The UKF defines its own *generative process*, which

[1] If $\sqrt{\mathbf{P}} = \mathbf{A} \Rightarrow \mathbf{P} = \mathbf{A}^\top\mathbf{A}$, then we use the rows in (4.12). If $\mathbf{P} = \mathbf{A}\mathbf{A}^\top$, then we use the columns.

linearizes the nonlinear functions f and g wherever in $\mathbf{x}_t$ a UKF filtering the time series would expect $\mathbf{x}_t$ to be. Therefore, it is possible to sample synthetic data from the UKF by sampling from its one-step-ahead predictions as seen in Algorithm 8. The sampling procedure augments the filter: predict-sample-correct. If we use the UKF with the same $\{\alpha, \beta, \kappa\}$ used to generate synthetic data, then the one-step-ahead predictive distribution will be the exact same distribution the data point was sampled from.

Algorithm 8 Sampling data from UKF's implicit model

```
1: $p(\mathbf{x}_1|\emptyset) \leftarrow (\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$
2: for $t = 1$ to $T$ do
3:     Measurement prediction step: $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$ using $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$
4:     Sample $\mathbf{y}_t$ from measurement prediction step distribution
5:     Bayes' update: $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ using $\mathbf{y}_t$
6:     Time update: find $p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t})$ using $p(\mathbf{x}_t|\mathbf{y}_{1:t})$
7: end for
```

4.1.3.1 Setting the Parameters

We summarize all the parameters as $\theta := \{\alpha, \beta, \kappa\}$. For any setting of θ the UKF will give identical predictions to the Kalman filter if f and g are both linear. Many of the heuristics for setting θ assume f and g are linear (or close to it), which is not the problem the UKF solves. For example, one of the heuristics for setting θ is that $\beta = 2$ is optimal if the state distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is exactly Gaussian [Wan and van der Merwe, 2000]. However, the state distribution will seldom be Gaussian unless the system is linear, in which case any setting of θ gives exact inference! We use the often recommended parameters $\alpha = 1$, $\beta = 0$, and $\kappa = 3$ [Thrun et al., 2005, Ch. 3], as the default UKF.

4.1.3.2 The Achilles' Heel of the UKF

The UKF, like the EKF, can have embarrassingly poor performance because its predictive variances can be far too small if the sigma points are placed in unlucky locations. Insufficient predictive variance will cause observations to have too much weight in the Bayes' update, which causes the UKF to fit to noise.

Meaning, the UKF will perform poorly even when evaluated on root-mean-square-error (RMSE), which only uses the predictive mean. On the NLL (log loss), the situation is even worse where too small predictive variances are heavily penalized.

In the most extreme case, the UKF can give a delta spike predictive distribution. We call this *sigma point collapse*. As seen in Figure 4.2, when the sigma points are arranged together horizontally the UKF has no way to know the function varies anywhere. Using a different set of sigma points we get either a completely degenerate solution (a delta spike) or a near optimal approximation within the class of Gaussian approximations.

A key contribution of this chapter is a method to learn the parameters θ in such a way that collapse becomes unlikely. Anytime collapse happens in training the marginal likelihood will be substantially lower. Hence, the learned parameters will avoid anywhere this delta spike occurred in training. Maximizing the marginal likelihood is tricky since it is not well-behaved for settings of θ that cause sigma point collapse.

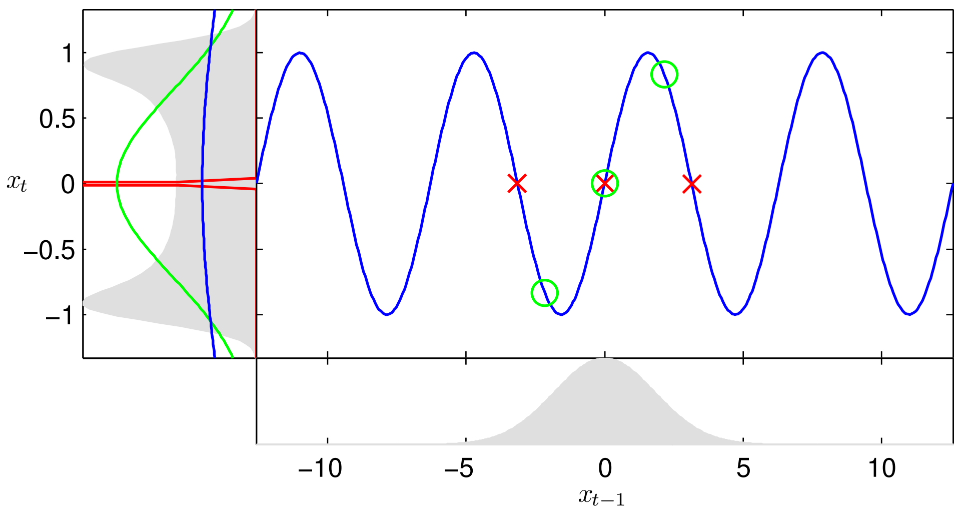


Figure 4.2: An illustration of a good and bad assignment of sigma points. The lower panel shows the true input distribution. The center panel shows the sinusoidal system function f (blue) and the sigma points for $\alpha = 1$ (red crosses $\times$) and $\alpha = 0.68$ (green circles $\bigcirc$). The left panel shows the true predictive distribution (shaded), the predictive distribution under $\alpha = 1$ (red spike) and $\alpha = 0.68$ (green). The parameters β and κ are fixed at their defaults in this example. We also show the EKF predictive distribution as the (widest) blue distribution.

The Kalman filter is the standard and most general exact filtering method used in continuous state spaces. The EKF and UKF form the standard approximate methods in the nonlinear cases. Although more exotic filtering algorithms exist, all approximate filters fit in a framework by approximating the predictive distribution from the system function and observation function.

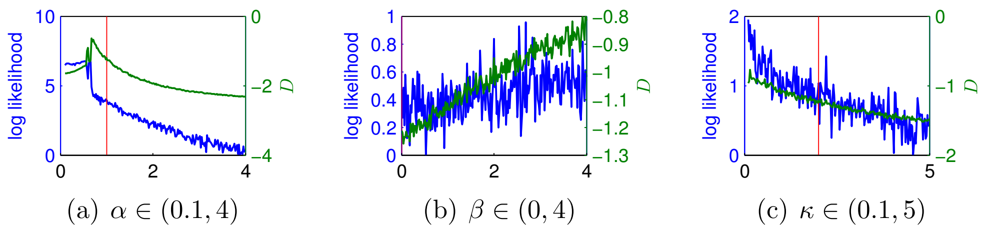
4.2 UKF Learning $\star$

In this section, we discuss a novel learning method for the UKF that avoids the sigma point collapse pitfall discussed in Section 4.1. Unlike the EKF, the UKF has free parameters that determine where to put the sigma points. Our contribution is a strategy for improving the UKF through a novel learning algorithm for appropriate sigma point placement: We call this method UKF-L. The key idea in the UKF-L is that the UKF and EKF are doing exact inference in a model that is somewhat "perverted" from the original model described in the state space formulation. The interpretation of EKF and UKF as models, not merely approximate methods, allows us to better identify their underlying assumptions. This interpretation also enables us to *learn* the free parameters in the UKF in a model based manner from training data. If the settings of the sigma point are a poor fit to the underlying dynamical system, the UKF can make horrendously poor predictions. Although, the UKF-L requires a learning to determine θ once the parameters are determined it has the same computational complexity as a standard UKF.

A common approach to estimating model parameters θ in general is to maximize the log marginal likelihood from the one-step-ahead predictions:

$$\ell(\theta) := \log p(\mathbf{Y}|\theta) = \sum_{t=1}^{T} \log p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \theta)\,. \tag{4.16}$$

Ordinarily it is reasonable to try a gradient based optimizer on the marginal likelihood, but as seen in Figure 4.3 the marginal likelihood shows substantial instability/local optima. The instability in the likelihood is likely the result of the phenomenon explained in Section 4.1.3.2, where a slight change in parameterization can avoid problematic sigma point placement. This makes the application of a gradient-based optimizer hopeless.



(a) $\alpha \in (0.1, 4)$ (b) $\beta \in (0, 4)$ (c) $\kappa \in (0.1, 5)$

Figure 4.3: Illustration of the UKF when applied to a pendulum system. We plot the cross-sections of the marginal likelihood (blue line) in nats per obs. when varying the parameters one at a time from their defaults (red vertical line). The dashed green line is the total variance diagnostic $D := \mathbb{E}\left[\log(|\mathbf{\Sigma}| \,/\, |\mathbf{\Sigma}_0|)\right]$, where $\mathbf{\Sigma}$ is the predictive variance in one-step-ahead prediction. We divide out the variance $\mathbf{\Sigma}_0$ of the time series when treating it as iid to make D unitless. We use D to help visualize when sigma point collapse is occurring, which is further addressed in Section 4.4.4. Values of θ with small predictive variances closely track the θ with low marginal likelihood. Note that the "noise" is a result of the sensitivity of the predictions to parameters θ due to sigma point collapse, not randomness in the algorithm as is the case with a particle filter.

We could apply Markov chain Monte Carlo (MCMC) and integrate out the parameters. However, this is usually "overkill" as the posterior on θ is usually highly peaked unless T is very small. Even in the case when T is small enough to spread the posterior out, we would still like a single point estimate for computational speed on the test set.[1]

We focus on learning using a Gaussian process based optimizer [Osborne et al., 2009; Srinivas et al., 2010], which is described in the next section. Since the marginal likelihood surface has an underlying smooth function but contains what amounts to additive noise, a probabilistic regression method seems a natural fit for finding the maximum.

[1] If we want to integrate the parameters out we must run the UKF with each sample of $\theta|\mathbf{Y}$ during test and average. To get the optimal point estimate of the posterior we would like to compute the *Bayes' point* [Snelson and Ghahramani, 2005].

4.2.1 Gaussian Process Optimizers

Estimating the parameters amounts to finding the maximum of a structured function: the log marginal likelihood. Therefore, it seems natural to use a prior over functions to guide our search. Given that Gaussian processes form a prior over functions we can use them for *global optimization*. The same principle has been applied to integration in Rasmussen and Ghahramani [2003]. Note that the use of GPs in this section is completely incidental to their use elsewhere in the chapter. The appropriateness of GPs for filtering is unrelated to their appropriateness for optimization.

GP optimization (GPO) allows for effective *derivative free* optimization. We consider the maximization of a likelihood function $\ell(\theta)$. GPs allow for derivative information $\partial_\theta \ell$ to be included as well, but in our case this is not useful due to the function's instability.

GPO treats optimization as a sequential decision problem in a probabilistic setting, receiving reward r when using the right input θ to get a large function value output $\ell(\theta)$. This setup is also known as a *Markov decision process* (MDP) [Sutton and Barto, 1998, Ch. 1]. At each step GPO uses its posterior over the objective function $p(\ell(\theta))$ to look for θ it believes has a large function value $\ell(\theta)$. A maximization strategy that is *greedy* will always evaluate the function $p(\ell(\theta))$ where the mean function $\mathbb{E}[\ell(\theta)]$ is the largest. A strategy that trades-off *exploration* with *exploitation* will take into account the posterior variance $\mathrm{Var}[\ell(\theta)]$. Areas of θ with high variance carry a possibility of having a large function value or high reward r. The optimizer is programmed to evaluate at the maxima of

$$J(\theta) := \mathbb{E}[\ell(\theta)] + K\sqrt{\mathrm{Var}[\ell(\theta)]} \in \mathbb{R}\,, \tag{4.17}$$

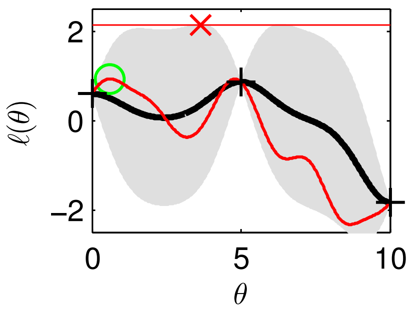
where $K \in \mathbb{R}^+$ is a constant to control the exploration exploitation trade-off. The objective J is known as the upper confidence bound (UCB) [Auer, 2000], since it optimizes the maximum of a confidence interval on the function value. The UCB has recently been analyzed theoretically by Srinivas et al. [2010]. The optimizer must also find the maximum of J, but since it is a combination of the GP mean and variance functions it is easy to optimize with gradient methods.

We summarize the method in Algorithm 9; the subroutine to compute J is shown in Algorithm 10. GPO assumes we provide a feasible set of θ to search within. Algorithm 10 optionally adds a barrier to make sure that new candidate points for evaluation stay within the feasible set. Alternatively, a constrained optimization routine could be used. Every iteration we consider another set of C candidate points to evaluate $\ell(\theta)$.

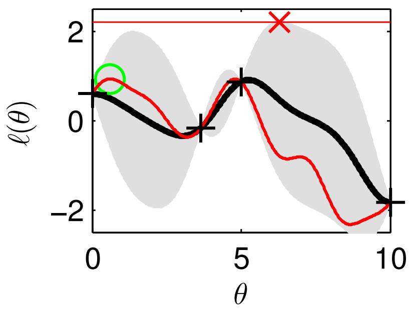
We illustrate the iterations of GPO in Figure 4.4. The function in the figure is highly non-convex and the global approach of the GP helps greatly. We consider a feasible region of $\theta \in [0, 10]$ and initialize the search by evaluating the function at the edges and the midpoint: $\{0, 5, 10\}$. The figure illustrates how the UCB criterion J trades off exploitation with exploration. For instance, a purely explorative strategy would evaluate the function at $\theta = 9$ in Figure 4.4(f), since the error bars are the largest even though the optima does not occur there. Likewise, a purely (greedy) exploitative strategy would get stuck evaluating the function at $\theta = 5$ in Figure 4.4(a) and never discover the maximum at $\theta = 1$.

Algorithm 9 Gaussian process optimization

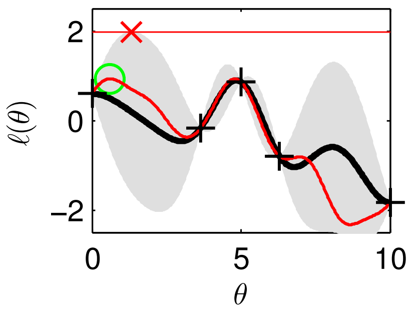
```
1: function GPO(Range $\in \mathbb{R}^{D\times 2}$, $f \in \mathbb{R}^D \to \mathbb{R}$, $M \in \mathbb{N}$, $C \in \mathbb{N}$)
2:                                   ▷ Use the end points and midpoints of the feasible set
3:     $\mathbf{X} \leftarrow$ multiGrid(Range) $\in \mathbb{R}^{D\times 3D}$
4:                                   ▷ Evaluate the function $f$ at each of the grid points
5:     $\mathbf{y} \leftarrow f(\mathbf{X}) \in \mathbb{R}^{3D}$
6:     for $i = 1$ to $M$ do
7:         Precompute $\mathbf{L}$, Cholesky of GP covariance matrix
8:         for $j = 1$ to $C$ do
9:                                   ▷ Sample a random initial point
10:            $\mathbf{x}_0 \sim \mathcal{N}(\mathbb{E}[\mathbf{x}], \mathrm{Cov}[\mathbf{x}]) \in \mathbb{R}^D$
11:                                  ▷ Maximize the UCB criterion $J$ w.r.t. $\mathbf{x}_\star$
12:            $(\mathbf{S}_j, F_j) \leftarrow \max \mathrm{UCB}(\mathbf{x}_\star, \mathbf{X}, \mathbf{y}, K, \mathbf{L})$, $\mathbf{x}_\star$ initialized at $\mathbf{x}_0$
13:        end for
14:        Append $\mathbf{S}(\operatorname{argmax} F)$ to $\mathbf{X}$          ▷ $\mathbf{X}$ now $\mathbb{R}^{D\times 3D+i}$
15:        Append $f(\mathbf{S}(\operatorname{argmax} F))$ to $\mathbf{y}$      ▷ $\mathbf{y}$ now $\mathbb{R}^{3D+i}$
16:    end for
17:    return $\mathbf{X}(\operatorname{argmax} \mathbf{y})$
18: end function
```



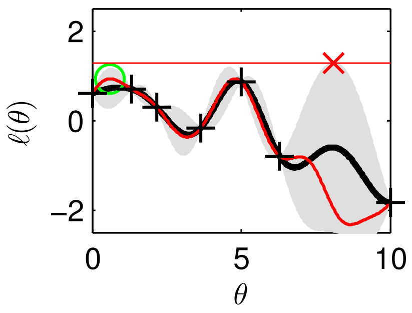
(a) Iteration 1



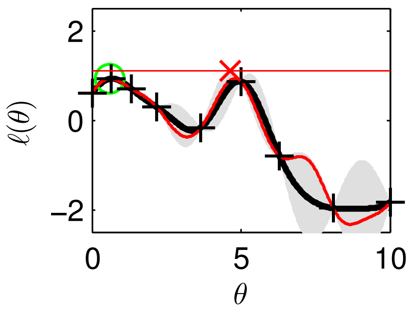
(b) Iteration 2



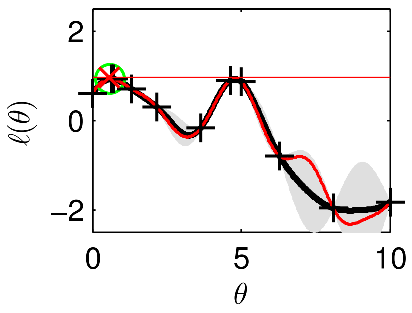
(c) Iteration 3



(d) Iteration 5



(e) Iteration 7



(f) Iteration 9

Figure 4.4: Illustration of GPO in one dimension. The red line represents the (highly non-convex) function we are attempting to optimize with respect to its input θ. Its true maximum is shown by the green circle. The black line and the shaded region represent the GP predictive mean and two standard deviation error bars. The black plus ($+$) show the points that have already been evaluated. The red cross ($\times$) is the maximum of $J(\theta)$ shown by the red horizontal line. Here we use $K = 2$ so the UCB criterion J corresponds to the top of the two standard deviation error bars shown in the plot. At each iteration GPO evaluates the function where the top of the error bars is highest.

Algorithm 10 Upper confidence bound

```
1:  function UCB($\mathbf{x}_\star \in \mathbb{R}^D, \mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{y} \in \mathbb{R}^N, K \in R^+, \mathbf{L} \in \mathbb{L}^N$)
2:          ▷ Compute $\mathbb{E}[\mathbf{x}_\star|\mathbf{X}, \mathbf{y}]$ and $\mathrm{Var}[\mathbf{x}_\star|\mathbf{X}, \mathbf{y}]$ using a GP in numerically stable way
3:      Find $\mathbf{K}_{\star\star} \in \mathbb{R}^+$ and $\mathbf{K}_{\mathbf{X},\star} \in \mathbb{R}^{N \times 1}$
4:      $\boldsymbol{\alpha} \leftarrow \mathbf{L}^\top \backslash (\mathbf{L} \backslash \mathbf{y})$
5:      $\bar{f} \leftarrow \mathbf{K}_{\mathbf{X},\star}^\top \alpha$                                  ▷ $\mathbb{E}[\mathbf{x}_\star|\mathbf{X}, \mathbf{y}]$
6:      $\boldsymbol{\beta} \leftarrow \mathbf{L}^\top \backslash (\mathbf{L} \backslash \mathbf{K}_{\mathbf{X},\star})$
7:      $f_{\mathrm{sd}} \leftarrow \sqrt{\mathbf{K}_{\star\star} - \mathbf{K}_{\mathbf{X},\star}^\top \boldsymbol{\beta}}$                 ▷ $\sqrt{\mathrm{Var}[\mathbf{x}_\star|\mathbf{X}, \mathbf{y}]}$
8:      $J \leftarrow \bar{f} + K f_{\mathrm{sd}}$                               ▷ Find the UCB
9:      Add a barrier to $J$, e.g. $\|\mathbf{x}_\star\|_{20}$                 ▷ Optional
10:     Compute $dJ$ w.r.t. $\mathbf{x}_\star$
11:     return $J$ and $dJ$
12: end function
```

4.3 Gaussian Process Inference and Learning ⋆

In this section we discuss a novel methodology for learning state space dynamics in a nonparametric manner using GPs. To do so, we first need to cover sparse Gaussian processes. We then combine sparse Gaussian processes with the uncertain input methodology in Section 3.2 to obtain a GP filter that can be placed in the general framework of Algorithm 6. The reader should note that the methods in this section are largely unrelated to the UKF-L.

Typically, a parametric form for the transition dynamics and observation model is assumed. General forms of the dynamics model for inference and learning were proposed in terms of radial basis functions (RBF) [Ghahramani and Roweis, 1999] and neural networks [Honkela and Valpola, 2005]. In the context of modeling human motion, Gaussian processes (GPs) have been used for inference in Wang et al. [2008] and Ko and Fox [2009]. Recently, GPs were used for filtering in the context of the UKF, the EKF [Ko and Fox, 2009], and the assumed density filter (ADF) [Deisenroth et al., 2009].

For nonlinear systems these methods encounter problems: The local linearizations of the EKF and the UKF can lead to the problems explained in the previous section such as sigma point collapse. Furthermore, learning using the UKF and the EKF typically requires a parametric form of the dynamics and measurement functions to be specified in advance. Neural network [Honkela and Valpola, 2005] and RBF [Ghahramani and Roweis, 1999] approaches have a constant level of uncertainty in the dynamics and measurement functions, which means they do not appropriately quantify uncertainty in f and g. Although probabilistic GPs are used in Wang et al. [2008] and Ko and Fox [2009], the maximum-a-posteriori (MAP) estimation (point estimate) of the latent states can lead to overconfident predictions because the uncertainty in the latent states is not accounted for. Other GP approaches proposed solely for filtering [Deisenroth et al., 2009; Ko and Fox, 2009] do take the state uncertainty into account, but require ground truth observations of the latent states during training, a strong assumption in many applications.

In this section, we address the shortcomings of the methods above by proposing the Gaussian process inference and learning (GPIL) algorithm for inference and learning in NLDS. Our flexible framework uses non-parametric GPs to model both the transition function and the measurement function. The GPs naturally account for three kinds of uncertainties in the real dynamical system: system noise, measurement noise, and model uncertainty. Our model integrates out the latent states unlike Ko and Fox [2009]; Wang et al. [2008], where a MAP approximation to the distribution of the latent state is used. The non-parametric nature of our model does not require a prespecified parametric form of the measurement and/or dynamics model. At the same time, GPIL does not require any ground truth observations of the latent states $\mathbf{x}$.

The main contributions of this section are twofold: We propose a tractable algorithm for approximate inference (filtering) in GP state space models. Using GP models for f and g, see (4.1), we propose learning without the need of ground truth observations $\mathbf{x}_i$ of the latent states.

4.3.1 Sparse Gaussian Processes

A prerequisite for applying the uncertain input methodology for Gaussian processes in state space models is the notion of a sparse Gaussian process. Therefore, we briefly cover the relevant portions of the sparse Gaussian process literature.

Gaussian processes form a fully connected model; there are no conditional independence assumptions between the elements of the output vector $\mathbf{f} := f(\mathbf{X}) \in \mathbb{R}^N$, where $\mathbf{X} \in \mathbb{R}^{N\times D}$ is a matrix of inputs. Most sparse Gaussian processes (SGP) introduce conditional independence assumptions. SGPs were originally introduced for computational speed ups during training: The $\mathcal{O}(N^3)$ cost for training becomes prohibitive for large data sets even for a standard regression task. In this thesis we use SGPs for a different purpose: When GPs are used in latent space, the training inputs are not observed exactly, and SGP methods must be used even if we are willing to pay a $\mathcal{O}(N^3)$ computational price during training. The methods in Section 3.2 can handle a single uncertain input to a GP; here we are dealing with an entire training set of uncertain inputs, which requires additional methodology. SGPs were first utilized in this context for *Gaussian process latent variable models* (GP-LVM) in Lawrence [2004].

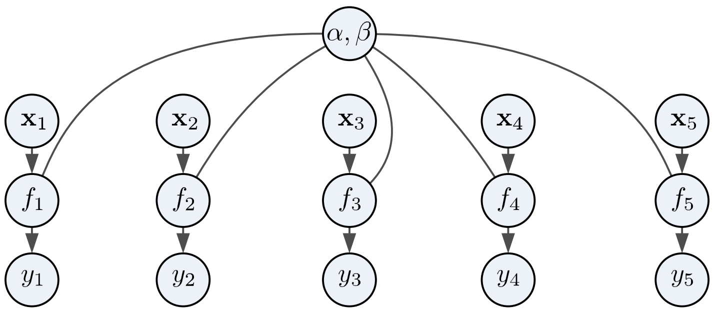


Figure 4.5: Graphical model for sparse Gaussian process using FITC approximation. Once the pseudo-points $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are conditioned on, all the function values f become independent.

Several different SGP methods have been developed. Many of them were unified into a common framework by Quiñonero-Candela and Rasmussen [2005]; certain approaches such as compact support kernels [Wendland, 2005, Ch. 9], conjugate gradient matrix inversion [Gibbs, 1997, Ch. 3], or projected processes [Csató and Opper, 2002] do not fit into the framework of Quiñonero-Candela and Rasmussen [2005], but we do not consider them here. In this section we summarize the main results from the *fully independent training conditional* (FITC) SGP found in Snelson and Ghahramani [2006]. In FITC there exists a set of M *pseudo-inputs* such that if we knew $\boldsymbol{\beta} := f(\boldsymbol{\alpha}) \in \mathbb{R}^M$, the pseudo-output, at all the pseudo-inputs $\boldsymbol{\alpha} \in \mathbb{R}^{M\times D}$ then our prediction of the function value at all other places would be independent:

$$p(\mathbf{f}|\mathbf{X}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{i=1}^{N} p(f(\mathbf{x}_i)|\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{x}_i) \tag{4.18}$$

$$= \mathcal{N}(\mathbf{K}_{\mathbf{x},\boldsymbol{\alpha}}\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}}^{-1}\boldsymbol{\beta}, (\mathbf{K}_{\mathbf{x},\mathbf{x}} - \mathbf{K}_{\mathbf{x},\boldsymbol{\alpha}}\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}}^{-1}\mathbf{K}_{\boldsymbol{\alpha},\mathbf{x}}) \odot \mathbf{I})\,, \tag{4.19}$$

where we have introduced the notation $\mathbf{K}_{\mathbf{x},\mathbf{x}} := k(\mathbf{X}, \mathbf{X}) \in \mathbb{S}^N$, $\mathbf{K}_{\boldsymbol{\alpha},\mathbf{x}} := k(\boldsymbol{\alpha}, \mathbf{X}) \in \mathbb{R}^{M\times N}$, $\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}} := k(\boldsymbol{\alpha}, \boldsymbol{\alpha}) \in \mathbb{S}^M$, and so on. This setup is summarized in terms of a graphical model in Figure 4.5. For notational brevity we also define the operator $\mathbf{Q}_{a,b} := \mathbf{K}_{a,\boldsymbol{\alpha}}\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}}^{-1}\mathbf{K}_{\alpha,b} \in \mathbb{R}^{|a|\times|b|}$. We also use $\mathbf{A} \odot \mathbf{I}$ to "zero out" all but the diagonal elements of a matrix $\mathbf{A}$. This type of approximation to a standard GP, referred to as a *full GP* in the SGP literature, introduces approximate low-rank covariance matrices of the training data. FITC induces the following marginal distribution on the training $\mathbf{f} \in \mathbb{R}^N$ and test points $\mathbf{f}_\star := f(\mathbf{x}_\star) \in \mathbb{R}^{N'}$ with $\mathbf{x}_\star \in \mathbb{R}^{N'\times D}$:

$$p(\mathbf{f}, \mathbf{f}_\star) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_{\mathbf{x},\mathbf{x}} - (\mathbf{Q}_{\mathbf{x},\mathbf{x}} - \mathbf{K}_{\mathbf{x},\mathbf{x}}) \odot \mathbf{I} & \mathbf{Q}_{\mathbf{x},\star} \\ \mathbf{Q}_{\star,\mathbf{x}} & \mathbf{K}_{\star,\star} \end{bmatrix}\right)\,. \tag{4.20}$$

Therefore, the FITC prediction after observing $\mathbf{y}$, noisy observations of $\mathbf{f}$, is

$$p(\mathbf{f}_\star|\mathbf{y}) = \mathcal{N}(\mathbf{K}_{\star,\boldsymbol{\alpha}}\boldsymbol{\Sigma}\mathbf{K}_{\boldsymbol{\alpha},\mathbf{x}}\boldsymbol{\Lambda}^{-1}\mathbf{y}, \mathbf{K}_{\star,\star} - \mathbf{Q}_{\star,\star} + \mathbf{K}_{\star,\boldsymbol{\alpha}}\boldsymbol{\Sigma}\mathbf{K}_{\boldsymbol{\alpha},\star})\,, \tag{4.21}$$

$$\boldsymbol{\Sigma} := (\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}} + \mathbf{K}_{\boldsymbol{\alpha},\mathbf{x}}\boldsymbol{\Lambda}^{-1}\mathbf{K}_{\mathbf{x},\boldsymbol{\alpha}})^{-1} \in \mathbb{S}^M\,, \tag{4.22}$$

$$\boldsymbol{\Lambda} := (\mathbf{K}_{\mathbf{x},\mathbf{x}} - \mathbf{Q}_{\mathbf{x},\mathbf{x}} + \sigma_0^2\mathbf{I}) \odot \mathbf{I} \in \mathbb{R}^{N\times N}\,. \tag{4.23}$$

This allows for the inversion of the covariance matrix to be done in less than $\mathcal{O}(N^3)$ time, lowering the computational complexity of GP training. One point of ambiguity is whether the we should assume there is observation noise on the pseudo-outputs $\boldsymbol{\beta}$; if yes, a noise variance is added to the diagonal of $\mathbf{K}_{\boldsymbol{\alpha},\boldsymbol{\alpha}}$. All of the calculations should follow through the same regardless. We assume the pseudo-outputs have noise.

The locations of the pseudo-inputs are now free parameters, which must be learned. A naive approach is to pick a random subset of the real inputs as pseudo-inputs. Note that pseudo-inputs do *not* have to be real inputs of the data points.

Although a random subset of the data is often used to initialize the pseudo-inputs, better performance is often obtained by optimizing the pseudo-inputs as hyper-parameters. We pick the locations that maximize the evidence. In a standard SGP setup we marginalize out the pseudo-outputs $\boldsymbol{\beta}$, which has less potential for overfitting than optimizing them. Unfortunately, when using SGPs for latent GP models, we often have to optimize the pseudo-outputs as well.

Therefore, although useful in sparse regression, in latent space models such as GPIL, (4.20) and (4.21) are irrelevant since they are found by integrating out $\boldsymbol{\beta}$. We only care about distributions conditioned on $\boldsymbol{\beta}$ since we optimize them as parameters. We use (4.19) for test set prediction. For a single test case, as is the case in GPIL, (4.19) is equivalent to a vanilla GP prediction with training inputs $\boldsymbol{\alpha}$ and training outputs $\boldsymbol{\beta}$.

SGP hyper-parameters are often optimized jointly with pseudo-inputs. Maximizing these regularization terms simultaneously with other parameters can lead to local optima difficulties. These types of problems were one of the reasons for moving from neural networks to GPs in the first place. Indeed, Lázaro-Gredilla [2010, Ch. 3] has represented certain SGP setups as two stage neural networks with the second layer weights (parameters) integrated out and the first layer weights corresponding to the pseudo-inputs. Methods of training SGP and avoiding local optima problems is an ongoing area of research [Titsias, 2009; Yan and Qi, 2010]. SGPs hyper-parameters are frequently initialized by learning them via a full GP on a random subset of the training data that is small enough to fit inside the given computational budget.

4.3.2 Model and Algorithmic Setup

We consider an NLDS, where the stochastic Markovian transition dynamics of the hidden states and the corresponding measurement function are given by (4.1). The transition function f and the measurement function g, in (4.1), are both unknown. In order to make predictions, we have to *learn* them solely given the information of T sequential observations $\mathbf{Y} := [\mathbf{y}_1, \dots, \mathbf{y}_T]$.

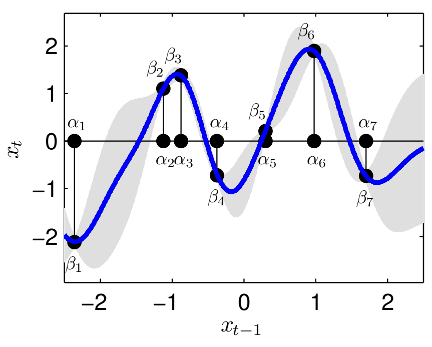
We use GPs to model both the unknown transition function f and the unknown measurement function g, and write $f \sim \mathcal{GP}_f$, $g \sim \mathcal{GP}_g$, respectively.

Throughout this section, we use the squared exponential kernel (3.5), SE-ARD, and a prior mean of zero. Independent GPs are used for each target dimension of f and g.

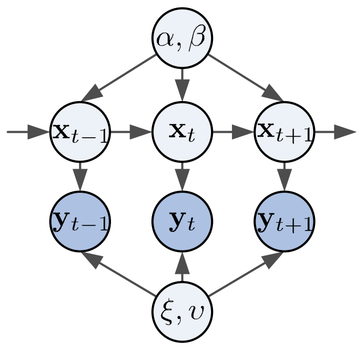
Since the latent states $\mathbf{X}$ are unobserved, we cannot learn f and g directly. Let us have a look at the "parameters" of a GP. In a standard GP setup, the GP can be considered effectively parameterized by the hyper-parameters, the training inputs, and the training targets. In the considered state space model, (4.1), the training inputs are never observed directly. Due to the difficulty of GPs with uncertain inputs on the entire training set, we use SGPs and parameterize a GP by a *pseudo training set*, which is considered a set of free parameters for learning. In other words, we learn parameter estimates of α and β from Figure 4.5, for both f and g, during training.

These parameters are related to the pseudo training inputs used for sparse GP approximations in the previous section. The pseudo inputs that parameterize the transition function f are denoted by $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_i \in \mathbb{R}^M\}_{i=1}^N$ and the corresponding pseudo targets are denoted by $\boldsymbol{\beta} = \{\boldsymbol{\beta}_i \in \mathbb{R}^M\}_{i=1}^N$. Intuitively, the pseudo inputs $\boldsymbol{\alpha}_i$ can be understood as the locations of the Gaussian basis functions (SE kernel), whereas the pseudo targets $\boldsymbol{\beta}_i$ are related to the function value at this location. We interpret the pseudo training set as N pairs of independent observations of transitions from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Note that the pseudo training set does *not* form a time series, i.e. $\boldsymbol{\beta}_i \neq \boldsymbol{\alpha}_{i+1}$. To parameterize the measurement function g, we follow the same approach and use the pseudo inputs $\boldsymbol{\xi} = \{\boldsymbol{\xi}_i \in \mathbb{R}^M\}_{i=1}^N$ and pseudo outputs $\boldsymbol{\upsilon} = \{\boldsymbol{\upsilon}_i \in \mathbb{R}^D\}_{i=1}^N$. We use $\boldsymbol{\upsilon}$ instead of training to $\mathbf{Y}$ directly since often $N \ll T$. An example of a pseudo training set and the corresponding GP model is in Figure 4.6(a).

Once the pseudo training set ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$) is determined, *predicting* $\mathbf{x}_t$ from $\mathbf{x}_{t-1}$ is a GP prediction [Deisenroth et al., 2009; Quiñonero-Candela et al., 2003]. Here, $\mathbf{x}_{t-1}$ serves as the (uncertain) test input, while $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are used as a standard GP training set. Likewise, predicting $\mathbf{y}_t$ from $\mathbf{x}_t$ corresponds to a GP prediction at uncertain inputs with $\mathbf{x}_t$ as the test input and a training set defined through $\boldsymbol{\xi}$ and $\boldsymbol{\upsilon}$. The model setup for predictions (conditioned on the pseudo training set)



(a) Pseudo points in GPIL



α, β
$\mathbf{x}_{t-1}$ $\mathbf{x}_t$ $\mathbf{x}_{t+1}$
$\mathbf{y}_{t-1}$ $\mathbf{y}_t$ $\mathbf{y}_{t+1}$
ξ, υ

(b) Graphical model for GPIL

Figure 4.6: Use of SGPs in GPIL. On the left we show an example of a function distribution inferred from a pseudo training set. The $\boldsymbol{\alpha}_i$ are the pseudo training inputs, while the $\boldsymbol{\beta}_i$ are the pseudo training targets. The shaded area is the 95% confidence region of the function (solid blue). On the right we show the graphical model for GPIL. The free parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and $\boldsymbol{\xi}, \boldsymbol{\upsilon}$ serve as a pseudo training set for $\mathcal{GP}_f$ and $\mathcal{GP}_g$, respectively. Note that $\mathcal{GP}_f$ and $\mathcal{GP}_g$ are *not* full GPs, but rather sparse GPs that impose the condition $\mathbf{x}_{t+1} \perp\!\!\!\perp \mathbf{x}_{t-1} | \mathbf{x}_t, \boldsymbol{\alpha}, \boldsymbol{\beta}$.

is

$$x_{ti} \sim \mathcal{GP}_f(\mathbf{x}_{t-1} | \boldsymbol{\alpha}, \boldsymbol{\beta}_i)\,, \quad y_{tj} \sim \mathcal{GP}_g(\mathbf{x}_t | \boldsymbol{\xi}, \boldsymbol{\upsilon}_j)\,,$$

where x_{ti} is the ith dimension of $\mathbf{x}_t$ and y_{tj} is the jth dimension of $\mathbf{y}_t$. Note that $\mathbf{x}_{t+1} \perp\!\!\!\perp \mathbf{x}_{t-1} | \mathbf{x}_t, \boldsymbol{\alpha}, \boldsymbol{\beta}$, which preserves the Markovian property in (4.1). It would therefore be more accurate to describe the SGP approximation as a fully independent conditional (FIC) rather than a FITC in the terminology of SGPs. Once the Markovian property is intact inference follows the structure of Algorithm 5 as is the case with the EKF and UKF. The corresponding graphical model is shown in Figure 4.6(b).

Without conditioning on the pseudo training set, the conditional independence property would be $\mathbf{x}_{t+1} \perp\!\!\!\perp \mathbf{x}_{t-1} | \mathbf{x}_t, f$, which requires conditioning on the infinite dimensional object f. This makes it difficult to design practical inference algorithms that exploit the Markovian property.

The hyper-parameters for the dynamics GP and the measurement GP are denoted by $\theta_f \in (\mathbb{R}^+)^{M+2\times M}$ and $\theta_g \in (\mathbb{R}^+)^{M+2\times D}$, respectively. Just as with the

LDS, the prior on the initial state is a Gaussian with mean $\boldsymbol{\mu}_0 \in \mathbb{R}^M$ and covariance $\boldsymbol{\Sigma}_0 \in \mathbb{S}^M$. We fix $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Sigma}_0 = \mathbf{I}$ since the scale of the latent states $\mathbf{X}$ is arbitrary, the other parameters can be rescaled to make any assumption on the moments of the initial state realistic. The entire parameter space is summarized as $\Theta := \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\xi}, \boldsymbol{\upsilon}, \theta_f, \theta_g, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\}$.

4.3.3 GP-ADF

In this section we describe the GP-ADF, which serves as a filter that does exact moment matching and is a subroutine in the GPIL algorithm.

The EKF and UKF algorithms are equivalent to the standard Kalman filter except that the moments, $\boldsymbol{\mu}_x$ and $\boldsymbol{\Sigma}_x$, propagated through the transition and measurement functions are substituted for their respective approximations according to the unscented transform (UKF) or first order Taylor series linearization (EKF). When the functions f and g are represented with a GP posterior, GP-ADF computes the moments, $\mathbb{E}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$, $\mathrm{Cov}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$, $\mathbb{E}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$, $\mathrm{Cov}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$, and $\mathrm{Cov}\left[\mathbf{x}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$, exactly unlike the EKF and UKF where the moments are seldom exact outside the linear case. When an SE-ARD covariance function (3.5) is used these exact moments are easily computed in a GP even for uncertain inputs (see Section 3.2). The ability to handle uncertain inputs is essential since we will be propagating the latent state, which will seldom be known with certainty.

If f and g are modeled by GPs, then finding $\mathbb{E}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$ and $\mathrm{Cov}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$ can be reduced to finding the predictive moments from the system function f with an uncertain test input. Likewise, finding $\mathbb{E}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$ and $\mathrm{Cov}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$ correspond to the predictive moments from the observation function g with an uncertain input. The variable $\mathrm{Cov}\left[\mathbf{x}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$ can be thought of as the cross-covariance between the input and the output when predicting with g.

It may seem paradoxical that although we cannot compute the exact moments for arbitrary f and g black box functions, when we place a GP prior on these functions and try to estimate from a finite number of evaluations, we can get exact moments when propagating moments through the posteriors on f and g. The explanation is that GP posteriors can always be represented as an RBF and therefore we are approximating f and g by an RBF; the basis functions are

Gaussian when an SE kernel is used. Once they are represented by an RBF computational speedups can be used to get exact moments. The predictive variance will take into account the extra uncertainty introduced by approximating f and g with a RBF containing a finite number of basis functions. Therefore, the GP-ADF is a superior approach over an ad-hoc approximation to f and g with an RBF representation.

Time update The time update corresponds to computing the GP-ADF approximation to `predict-`x: the one-step-ahead predictive distribution of the hidden state $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ using $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ as a (Gaussian) prior on $\mathbf{x}_{t-1}$. Propagating a density on $\mathbf{x}_{t-1}$ to a density on $\mathbf{x}_t$ corresponds to GP prediction (under model $\mathcal{GP}_f$) with uncertain inputs, addressed in Section 3.1. The exact mean $\boldsymbol{\mu}_x$ and covariance $\boldsymbol{\Sigma}_x$ of the predictive distribution can be computed analytically. The predictive distribution on $\mathbf{x}_t$ is therefore approximated by a Gaussian $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ using exact moment matching. Therefore, we utilize Algorithm 1 to implement the `predict-`x step:

$$(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow \texttt{GPUR}(\theta_f, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)\,. \tag{4.24}$$

The predictive distribution $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{1:t-1})$ of the latent state explicitly models *three* sources of uncertainty: a) the uncertainty about the underlying function, b) the uncertainty induced by the system noise, and c) the uncertain inputs.

Measurement prediction Analogously, we approximate the predictive distribution in *observed space* using `predict-`y. We approximate the measurement prediction $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$ by a Gaussian $\mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$ with the exact mean and the exact covariance using the above prediction $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ as a prior on $\mathbf{x}_t$:

$$(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy}) \leftarrow \texttt{GPUR}(\theta_g, \boldsymbol{\xi}, \boldsymbol{\upsilon}, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)\,. \tag{4.25}$$

Both the time update and measurement prediction scale quadratically in the number pseudo inputs N.

Bayes' update The Bayes' update computes a posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ by refining the predictive distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ by incorporating the most recent measurement $\mathbf{y}_t$. Since we already have exact moments, we merely plug them into the `filter-update` to complete the ADF [Deisenroth et al., 2009].

4.3.4 Learning in GPIL

Following from the approach of the UKF-L, we optimize the implied marginal likelihood by the GP-ADF directly,

$$\ell(\Theta) := \log p(\mathbf{Y}|\Theta) = \sum_{t=1}^{T} \log p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \Theta)\,, \tag{4.26}$$

except that here we can use gradient based optimization. The predictive distribution $p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \Theta)$ is provided by the GP-ADF. This is in contrast to the EM approach to learn state space models.

Following the example in LDS we could apply the EM algorithm in the state space models for inference and learning. The shortcoming of this approach in nonlinear systems is that the smoothing step requires further approximations. Additionally in the GPIL the expectation in the M-step cannot be computed exactly either. By optimizing a corrupt likelihood, the learning algorithm does not account for the biases introduced by the approximate inference algorithm. Furthermore, it may not even converge since the M-step itself is approximate. By optimizing the implied marginal likelihood directly we avoid these problems.

Initialization There exist some inherent traps in learning an algorithm like GPIL: We are attempting to optimize parameters, $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\xi}, \boldsymbol{\upsilon}\}$, and their regularizers, θ_f and θ_g, simultaneously. This allows for the parameters to fall into bad local optima. For instance, explanations of the data as all noise are always plausible. This dilemma became highly apparent with the development of regularized neural networks [Neal, 1996]. The regularization predicament also plagues sparse GP approximations in general. Therefore, good solutions require good initializations that are within the domain of influence of an optimum that is reasonable.

The key difficulty in learning the GPIL is the existence of the latent states $\mathbf{X}$; indeed, this is the motivation behind using EM in state space models. Therefore, we choose to initialize GPIL using hard estimates for latent states from an LDS. In order to infer the latent states with an LDS we must use a Kalman filter and estimate its parameters. Although EM is common for learning in LDS, and will converge to a local optimum, EM would need to be initialized too. Therefore, the *N4SID* algorithm [van Overschee and de Moor, 1994], also known as subspace identification, is popular in practice for initializing nonlinear methods. The N4SID algorithm is a closed form algorithm for estimating the parameters in an algorithm based on an SVD; N4SID comes with guarantees of being sufficiently "close" to the true parameters although not being at an optimum of the likelihood. Once we infer the parameters with N4SID, we get a hard estimate of the latent states $\widehat{\mathbf{X}}$ using a Kalman filter for initialization purposes.

Next, we initialize the hyper-parameters of the GPIL, θ_f and θ_g. In the GPIL we have an advantage over typical sparse GP approximations. We can learn the hyper-parameters (the regularizers) on the full data set, and then leave them fixed when learning the pseudo training set. In standard sparse GPs this is not practical since the point is to avoid the computational cost of using the full set, and learning the hyper-parameters on the full set would defeat the purpose of using a sparse GP in the first place. In GPIL we use SGPs due to the latent states $\mathbf{x}$, not the size of the training set.

To learn the pseudo-points we can do standard sparse GP learning since the inputs to the GPs are not uncertain if we are using $\widehat{\mathbf{X}}$. Therefore, we only have to optimize the pseudo-inputs and not the pseudo-outputs, limiting the overfitting potential. We follow the standard practice in sparse GPs and initialize the pseudo-inputs with a random subset of the real inputs.

Once the pseudo-inputs are learned we can find the pseudo-outputs by evaluating the posterior mean of sparse GPs evaluated at the pseudo-inputs. We have now initialized all the parameters of the GPIL, and it can be followed by direct likelihood maximization to correct for signal missed in initialization, biases in initialization, and biases in the approximations in the GP-ADF filtering algorithm.

Summary of GPIL algorithm We summarize both the learning and filtering aspects of the algorithm in Algorithm 11. The GP-ADF likelihood is compatible with complex step [Martins et al., 2003] trick for calculating the derivatives in Line 7.

The GP-ADF fits in the standard framework of approximate filters by providing exact predictive moments from the system and observation functions by modeling them as GPs. Once modeled as GPs, exact moment methods for predicting with uncertain inputs can be used. We can use sparse GP methods to do learning of the system and observation functions used by the GP-ADF.

Algorithm 11 Gaussian process inference and learning (GPIL)

```
1: function GPIL($\mathbf{Y} \in \mathbb{R}^{T\times D}, N \in \mathbb{N}, M \in \mathbb{N}$)
2:     Find initial LDS using N4SID with $M$ latent dimensions
3:     $(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \leftarrow (\mathbf{0}, \mathbf{I})$
4:     Filter initial states using Kalman filtering
5:     Initialize $\theta_f$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ using SGP on latent state means
6:     Initialize $\theta_g$, $\boldsymbol{\xi}$, $\boldsymbol{\upsilon}$ using SGP on latent state means to $\mathbf{Y}$    $\triangleright$ Both SGPs use $N$ pseudo-points
7:     Maximize predictive likelihood of GP-ADF using (4.26)
8:     return $\Theta \leftarrow \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\xi}, \boldsymbol{\upsilon}, \theta_f, \theta_g, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\}$
9: end function
10: function GP-ADF($\mathbf{Y}, \theta_f, \boldsymbol{\alpha}, \boldsymbol{\beta}, \theta_g, \boldsymbol{\xi}, \boldsymbol{\upsilon}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0$)
11:     $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow (\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$
12:     for $t = 1$ to $T$ do
13:             $\triangleright$ Find $\mathbb{E}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$, $\mathrm{Cov}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$, and $\mathrm{Cov}\left[\mathbf{x}_t, \mathbf{y}_t|\mathbf{y}_{1:t-1}\right]$ using Alg. 1
14:         $(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy}) \leftarrow$ GPUR$(\theta_g, \boldsymbol{\xi}, \boldsymbol{\upsilon}, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$
15:                 $\triangleright$ Find $\mathbb{E}\left[\mathbf{x}_t|\mathbf{y}_{1:t}\right]$ and $\mathrm{Cov}\left[\mathbf{x}_t|\mathbf{y}_{1:t}\right]$ using Alg. 6
16:         $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow$ filter-update$(\mathbf{y}_t, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x, \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y, \boldsymbol{\Sigma}_{xy})$
17:                     $\triangleright$ Find $\mathbb{E}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$ and $\mathrm{Cov}\left[\mathbf{x}_{t+1}|\mathbf{y}_{1:t}\right]$
18:         $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \leftarrow$ GPUR$(\theta_f, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ using Algorithm 1
19:     end for
20:     return value of $\boldsymbol{\mu}_y$ and $\boldsymbol{\Sigma}_y$ for each iteration
21: end function
```

4.4 Results for Filtering

We test our method on filtering in three dynamical systems: the sinusoidal dynamics, the Kitagawa dynamics used in Deisenroth et al. [2009]; Kitagawa [1996], and pendulum dynamics used in Deisenroth et al. [2009]. The sinusoidal dynamics are described by

$$x_{t+1} = 3\sin(x_t) + \epsilon_t\,, \quad \epsilon_t \sim \mathcal{N}(0, 0.1^2)\,, \tag{4.27}$$

$$y_t = \sigma(x_t/3) + \nu_t\,, \quad \nu_t \sim \mathcal{N}(0, 0.1^2)\,. \tag{4.28}$$

where $\sigma(\cdot)$ represents a sigmoid. The Kitagawa model is described by

$$x_{t+1} = 0.5x_t + \frac{25x_t}{1+x_t^2} + \epsilon_t\,, \quad \epsilon_t \sim \mathcal{N}(0, 0.2^2)\,, \tag{4.29}$$

$$y_t = 5\sin(2x_t) + \nu_t\,, \quad \nu_t \sim \mathcal{N}(0, 0.01^2)\,. \tag{4.30}$$

The Kitagawa model was presented as a filtering problem in Kitagawa [1996]. The pendulum dynamics are described by a discretized ordinary differential equation (ODE) at $\Delta_t = 400\,\text{ms}$. The pendulum possesses a mass $m = 1\,\text{kg}$ and a length $l = 1\,\text{m}$. The pendulum angle φ is measured anti-clockwise from hanging down. The state $\mathbf{x} = [\varphi, \dot\varphi]^\top$ of the pendulum is given by the angle φ and the angular velocity $\dot\varphi$. The ODE is

$$\frac{d}{dt}\begin{bmatrix}\dot\varphi \\ \varphi\end{bmatrix} = \begin{bmatrix}\frac{-mgl\sin\varphi}{ml^2} \\ \dot\varphi\end{bmatrix}\,, \tag{4.31}$$

where g the acceleration of gravity. The measurement function is

$$\mathbf{y}_t = \begin{bmatrix}\arctan\left(\frac{p_1 - l\sin(\varphi_t)}{p_1 - l\cos(\varphi_t)}\right) \\ \arctan\left(\frac{p_2 - l\sin(\varphi_t)}{p_2 - l\cos(\varphi_t)}\right)\end{bmatrix}\,, \quad \begin{bmatrix}p_1 \\ p_2\end{bmatrix} = \begin{bmatrix}1 \\ -2\end{bmatrix}\,, \tag{4.32}$$

which corresponds to *bearings only measurement* since we do not directly observe the velocity. We use system noise $\mathbf{Q} = \text{diag}(\begin{bmatrix}0.1^2 & 0.3^2\end{bmatrix})$ and $\mathbf{R} = \text{diag}(\begin{bmatrix}0.2^2 & 0.2^2\end{bmatrix})$ as observation noise.

For all the problems we compare the UKF-L with the UKF-D, EKF, GP-UKF, GP assumed density filter (GP-ADF), and time independent model (TIM); we use UKF-D to denote a UKF with default parameter settings, and UKF-L for learned parameters. The TIM treats the data as iid normal and is inserted as a reference point. The GP-UKF and GP-ADF use GPs to approximate f and g and exploit the properties of GPs to make tractable predictions. The Kitagawa and pendulum dynamics were used by Deisenroth et al. [2009] to illustrate the performance of the GP-ADF and the poor performance of the UKF. Deisenroth et al. [2009] used the default settings of $\alpha = 1$, $\beta = 0$, $\kappa = 2$ for all of the experiments. We used exploration trade off $K = 2$ for the GPO in all the experiments. Additionally, GPO used the squared-exponential with automatic relevance determination (SE-ARD) covariance function plus a noise term of 0.01 nats per obs. We set the GPO to have a maximum number of function evaluations of 100; even better results can be obtained by letting the optimizer run longer to hone the parameter estimate. We show that by *learning* appropriate values for $\theta = \{\alpha, \beta, \kappa\}$ we can match, and exceed, the performance of the GP-ADF and other methods.

The models were evaluated on one-step-ahead prediction. The evaluation metrics were the negative log-predictive likelihood (NLL), the mean squared error (MSE), and the mean absolute error (MAE) between the mean of the prediction and the true value. Note that unlike the NLL, the MSE and MAE do not account for uncertainty. Also, the MAE will be more difficult for approximate methods than MSE. For MSE, the optimal action is to predict the mean of the predictive distribution, while for the MAE it is the median. Most approximate methods attempt to moment match to a Gaussian and preserve the mean; the median of the true predictive distribution is implicitly assumed to be the same as the mean.

4.4.1 Sinusoidal Dynamics

The models were trained on $T = 1000$ observations from the sinusoidal dynamics and tested on $R = 10$ restarts with $T = 500$ points each. The initial state was sampled from a standard normal $x_1 \sim \mathcal{N}(0, 1)$. The GP-ADF inherently needs to do learning and cannot merely accept a known system function like the EKF and UKF. Although this somewhat disadvantages the GP-ADF, we gave it access to samples from the latent state to learn f as accurately as possible. The UKF optimizer found the optimal values $\alpha = 2.0216$, $\beta = 0.2434$, and $\kappa = 0.4871$.

The predictive results are shown in Table 4.1. The measures are supplied with 95% confidence intervals. NLL is reported in nats per obs., while MSE and MAE are in the units of $\mathbf{y}^2$ and $\mathbf{y}$, respectively.

Table 4.1: **Sinusoid**: Comparison of the methods on the sinusoidal dynamics. The p-values from a paired two-sided t-test under the null hypothesis that the UKF-L has the same mean loss as the other methods are $p < 0.0001$ for all alternate methods and loss measures. Since NLL on continuous variables has an arbitrary additive constant corresponding to the parameterization of the space (see Section B.1), the NLL values have been shifted so the best performing method has NLL zero.

Method	NLL $\times 10^1$	MSE $\times 10^3$	MAE $\times 10^2$
EKF	3.91±0.38	30.0±1.3	13.60±0.30
GP-ADF	1.40±0.17	26.03±0.99	12.96±0.27
GP-UKF	1.72±0.18	27.3±1.1	13.25±0.28
TIM	3.22±0.16	36.1±1.2	15.98±0.29
UKF-D	1.03±0.17	23.04±0.89	12.21±0.25
UKF-L	**0.00 ± 0.24**	**18.81 ± 0.77**	**10.91 ± 0.24**

Table 4.2: **Kitagawa**: Comparison of the methods on the Kitagawa dynamics. The p-values from a paired two-sided t-test under the null hypothesis UKF-L is the same as the other methods are $p < 0.0001$ for all alternate methods and loss measures, except a where $p = 0.0082$.

Method	NLL $\times 10^1$	MSE $\times 10^0$	MAE $\times 10^1$
EKF	960±720[a]	7.24±0.82	14.66±0.99
GP-ADF	10.18±0.16	18.03±0.43	40.26±0.60
GP-UKF	> 1000	23.01±0.96	41.7±1.1
TIM	10.10±0.12	18.23±0.42	40.57±0.59
UKF-D	15.8±5.5	5.07±0.58	12.71±0.82
UKF-L	**0.0 ± 2.1**	**3.40 ± 0.45**	**10.46 ± 0.67**

Table 4.3: **Pendulum**: Comparison of the methods on the pendulum dynamics. The p-values from a paired two-sided t-test under the null hypothesis UKF-L is the same as the other methods are $p < 0.0001$ for all alternate methods and loss measures. Since the observations in the pendulum data are angles we projected the means and the data to the complex plane before computing MSE and MAE.

Method	NLL $\times 10^2$	MSE $\times 10^2$	MAE $\times 10^2$
EKF	23.4±4.5	$\mathbf{20.42 \pm 0.45}$	$\mathbf{61.96 \pm 0.63}$
GP-ADF	47.74±0.94	39.06±0.48	95.96±0.66
GP-UKF	241.3±3.8	81.20±0.90	139.88±0.93
TIM	41.7±1.2	40.93±0.43	101.16±0.60
UKF-D	251.4±8.0	54.98±0.81	111.51±0.99
UKF-L	$\mathbf{0.0 \pm 2.6}$	24.75±0.68	66.54±0.70

4.4.2 Kitagawa

The Kitagawa model has a tendency to stabilize around $x = \pm 7$, where it is linear. The challenging portion for filtering is away from the stable portions where the dynamics are highly nonlinear. Deisenroth et al. [2009] evaluated the model using $R = 200$ independent starts of the time series allowed to run only $T = 1$ time steps, which we find somewhat unrealistic. Therefore, we allow for $T = 10$ time steps with $R = 200$ independent starts. In this example, $x_1 \sim \mathcal{N}(0, 0.5^2)$.

The learned values of the UKF-L parameters were $\alpha = 0.3846$, $\beta = 1.2766$, $\kappa = 2.5830$; quantitative results are shown in Table 4.2.

4.4.3 Pendulum

The models were tested on $R = 100$ runs of length $T = 200$ each, with:

$$\mathbf{x}_1 \sim \mathcal{N}(\begin{bmatrix}-\pi & 0\end{bmatrix}^\top, \mathrm{diag}(\begin{bmatrix}0.1^2 & 0.2^2\end{bmatrix})) . \tag{4.33}$$

The initial state mean of $\begin{bmatrix}-\pi & 0\end{bmatrix}^\top$ corresponds to the pendulum being in the downward position. The models were trained on $R = 5$ runs of length $T = 200$. We found that in order to perform well on NLL, but not on MSE and MAE, multiple runs of the time series were needed during training; otherwise, TIM had the best NLL. This is because if the time series is initialized in one state the model will not have a chance to learn the needed parameter settings to avoid rare, but still present, sigma point collapse in other parts of the state space. A short period of sigma point collapse in a long time series can give the models a worse NLL than even TIM due to incredibly small likelihoods. The MSE and MAE losses are more bounded, so a short period of poor performance will be offset by good performance periods. Even when $R = 1$ during training, sigma point collapse is much rarer in UKF-L than UKF-D. The UKF-L found optimal values of the parameters to be $\alpha = 0.5933$, $\beta = 0.1630$, $\kappa = 0.6391$. This is further evidence that the correct θ is hard to proscribe a priori and must be learned empirically. We compare the predictions of the default and learned settings in Figure 4.7. Quantitative results are shown in Table 4.3.

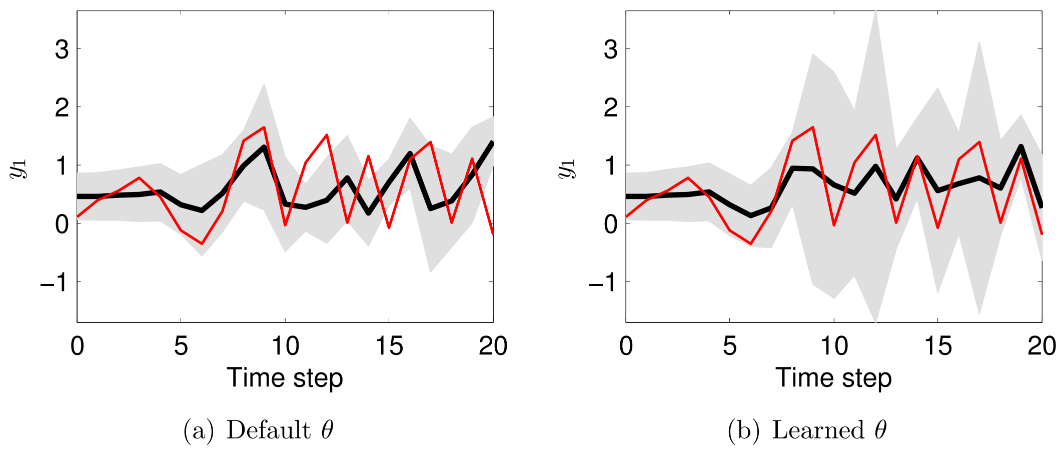


Figure 4.7: Comparison of UKF-D and UKF-L for one-step-ahead prediction; we show the first element of $\mathbf{y}_t$ in the Pendulum model for the first 20 time steps. The red line is the truth, while the black line and shaded area represent the mean and 95% confidence interval of the predictive distribution.

4.4.4 Analysis of Sigma Point Collapse

We find that the marginal likelihood is extremely unstable in regions of θ that experience sigma point collapse. When sigma point collapse occurs, the predictive variances become far too small, making the marginal likelihood much more susceptible to noise. Hence, the marginal likelihood is smooth near its maximum where sigma point collapse is unlikely, as seen in Figure 4.3. As a diagnostic D for sigma point collapse we look at the mean $|\mathbf{\Sigma}|$ of the predictive distribution.

4.5 Results for Learning

In this section we illustrate the learning ability of the GPIL on some synthetic examples. We thoroughly benchmark its performance on real data sets, the data described in Chapter 2, in Section 5.5. We evaluate on the sinusoidal dynamics of (4.27) and (4.28) with an added twist of adding two noise dimensions to the observations distributed according to a standard normal. The results were produced using a pseudo training set of size $N = 10$ with $T = 200$ training set duration.

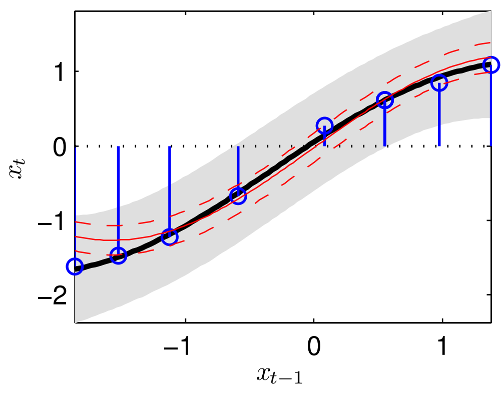
The true dynamics and observation model are compared to the learned model, $\mathcal{GP}_f$ and $\mathcal{GP}_g$, in Figure 4.8. Since the latent function is unidentifiable we transformed the latent state coordinate system in the figures. It seems that the GPIL set the latent coordinate system to be approximately $x' = -3x$. On the far right of the system function, it appears the GPIL under-estimates the true function. However, when we consider the observation function, very negative values are under-valued when projecting back into observable space. Therefore, the GPIL learned in a slightly nonlinear reparameterization of the latent coordinate system. The error bars (shaded area) on the learned dynamics model are larger than the true noise level since they include both system noise *and* model uncertainty.

Quantitative results for the sinusoidal dynamics are shown in Table 4.4 using one-step-ahead prediction. We compared GPIL to TIM and the ARGP as a reference.

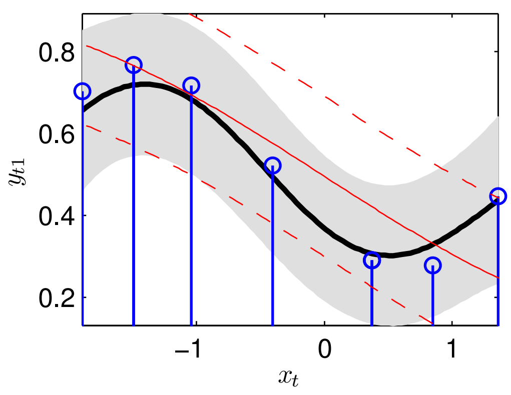
We use $T' = 1000$ test set observations for the evaluation procedure. The evaluation metrics were the negative log-predictive likelihood (NLL), the mean squared error (MSE), and mean absolute error (MAE).

4.5.1 Discussion

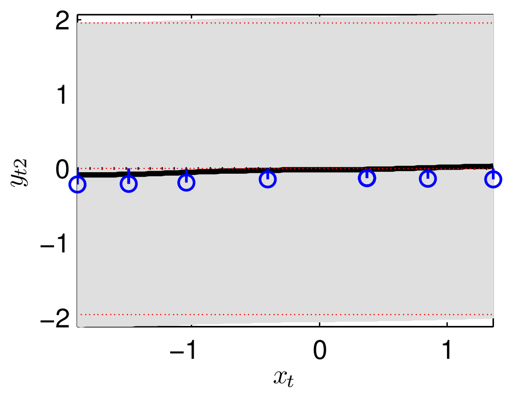
The learned parameters of the UKF performed significantly better than the default UKF for all error measures and data sets. Likewise, it performed signifi-



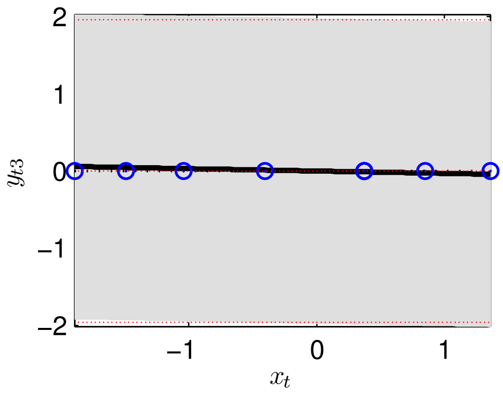
(a) System function



(b) Observation function 1



(c) Observation function 2



(d) Observation function 3

Figure 4.8: True (red) and learned (black) transition/observation function. The 95% error bars of the learned system are represented by the gray area and 95% error bars for the true system are shown by the (red, dashed) lines. The blue stem plots are the learned pseudo-inputs. We show the observation function for each of the three observation dimensions. The system is unidentifiable and it seems GPIL has roughly learned in a coordinate system of $x' = -3x$. The true functions have been transformed accordingly.

Table 4.4: **Sinusoid**: Evaluation of the GPIL on the sinusoidal dynamics example. We report the NLL per data point, MSE, and MAE with the NLL 95% error bars. The p-values from a paired two-sided t-test under the null hypothesis GPIL is the same as the other methods are $p < 0.0001$ for all alternate methods and loss measures.

Method	NLL $\times 10^1$	MSE $\times 10^0$	MAE $\times 10^1$
ARGP	1.90±0.77	2.12±0.13	17.69±0.54
GPIL	**0.00 ± 0.76**	**2.06 ± 0.13**	**17.32 ± 0.54**
TIM	3.66±0.73	2.11±0.13	17.93±0.55

cantly better than all other methods except against the EKF on the pendulum data on MAE/MSE. We found that results could be improved further by averaging the predictions of the UKF-L and the EKF.

There exists another set of parameters rarely addressed in the UKF. The Cholesky factorization is typically used for $\sqrt{\mathbf{\Sigma}}$. However, any matrix square-root can be used. Meaning, we can apply a rotation matrix to the Cholesky factorization $\mathbf{\Sigma}$ and get a valid UKF, which gives us $\mathcal{O}(D^2)$ extra degrees of freedom (parameters). We could attempt to learn these as well. However, the beauty of the UKF-L is that the number of parameters to learn is three regardless of D. Global optimization using GPO, or other methods, could become more difficult in the presence of an extra $D - 1$ parameters.

Another noteworthy extension of the UKF-L methodology is to particle filters [Doucet et al., 2001]. Particle filters are known to have even more arbitrary parameters than the UKF, such as in the proposal distribution. Although the Monte Carlo nature of particle filters could present challenges, the model based learning approach could add some robustness to particle filtering.

GP-ADF and UKF-L A somewhat surprising result in filtering is that UKF-L does better than GP-ADF on the problems tried, even on MSE. Since the GP-ADF uses exact moment matching, i.e. an exact mean, it is reasonable to expect that it would do better than UKF-L on MSE, where the optimal action is to report the mean. However, the GP-ADF gives the exact moments *only* for a single propagation, such as `predict-`x, through a GP. Since the true input distribution in the next step, such as `predict-`y, is not Gaussian the approximation compound and the GP-ADF no longer gives the exact predictive moments. There are also compound approximations in the UKF-L as well, which we might expect to cancel out with the GP-ADF approximations. However, since the UKF-L finds parameters θ that work empirically we get approximations that work globally. In contrast, the GP-ADF uses purely local moment-matching approximations.

Additionally, the GP-ADF requires training points to get an RBF representation of the dynamics. For filtering, this gives it a distinct disadvantage to the EKF and UKF, where we can pass it the "platonic" function (a parametric form rather than data points). However, the GP-ADF has an advantage when little is known about the system dynamics. The GP-ADF learns the dynamics in a nonparametric manner, and only being able to learn from training points is not a disadvantage if training points are all we have. A possible middle ground, for future research, is to apply the UKF-L methods to the GP-UKF.

Computational complexity The UKF-L, UKF, and EKF have test set computational time $\mathcal{O}(DT(D^2 + M))$. The GP-UKF and GP-ADF have complexity $\mathcal{O}(T(N^2DM^2 + N^2D^3))$, where N is the number of points used in training to learn f and g. This means that if $N^2 \gg D$ then UKF-L will have a much smaller computation complexity than the GP-ADF, which also attempts to avoid sigma point collapse. Typically it will take much more than D points to approximate a function in D dimensions, which implies we will almost always have $N^2 \gg D$. The D^3 term in GP-ADF comes from the covariance calculations in the GP. This is usually not problematic as D is typically small. If a large number of training points N is needed to approximate f and g then the GP-ADF and GP-UKF become much slower than the UKF.

4.6 Conclusions

Filtering We presented an automatic and model based mechanism to learn the parameters of a UKF, $\{\alpha, \beta, \kappa\}$, in a principled way. The UKF can be reinterpreted as a generative process that performs inference on a slightly different NLDS than desired through specification of f and g. We demonstrated how the UKF can fail arbitrarily badly in very nonlinear problems through sigma point collapse. Learning the parameters makes sigma point collapse less likely to occur. When the UKF learns the correct parameters from data, it can outperform other filters on common benchmark dynamical systems problems. This includes filters designed to avoid sigma point collapse, such as the GP-ADF. The GP-ADF has the strength of being able to learn system dynamics in a nonparametric manner. However, this is also a disadvantage since the GP-ADF can *only* learn this way and cannot accept a known parametric form as is the case for the UKF-L. This gives the UKF-L more information to exploit.

Learning Independent to the UKF-L, we have introduced the GPIL algorithm. In GPIL, the latent states $\mathbf{x}_t$ are *never* observed directly. Only observations $\mathbf{y}_t$ can be accessed to train the latent dynamics and measurement functions. Contrarily, direct access to ground truth observations of a latent state sequence was required in Deisenroth et al. [2009]; Ko and Fox [2009] for training. Parameterizing a GP using the pseudo training set is one way to *train* a GP with unknown inputs. In principle, we would train the model by integrating the pseudo training set out. However, this approach is analytically intractable.

With GPIL we introduced a general method for inference and learning in nonlinear state space models. Both the transition function between the hidden states and the measurement function are modeled by GPs allowing for quantifying model uncertainty and flexible modeling. The free parameters of the GPs are their hyper-parameters and a pseudo training set, which are jointly learned using a gradient-based optimizer. We demonstrated that GPIL successfully learned nonlinear (latent) dynamics based on noisy measurements only.

Chapter 5

Change Point Detection ⋆

Many processes have *change point* structure: Stock markets exhibit change points when some significant economic event causes an increase in volatility, weather systems may exhibit change points during years of El Niño, and electronic systems show change points when devices begin to fail or configurations change. An inability to react to regime changes can have a detrimental impact on predictive performance. Change point detection (CPD) attempts to reduce this impact by recognizing regime change events and adapting the predictive model appropriately. As a result, CPD is a useful tool in a diverse set of application domains including robotics, process control, and finance.

CPD is especially relevant to finance where risk resulting from parameter changes is often neglected in models. For example, Gaussian copula models used in pricing collateralized debt obligations (CDOs) had two key flaws: assuming that subprime mortgage defaults have a fixed correlation structure, and using a point estimate of these correlation parameters learned from historical data prior to the burst of the real-estate bubble [Jones, 2009; Li, 2000]. Bayesian change point analysis avoids both of these problems by assuming a change point model of the parameters and integrating out the uncertainty in the parameters rather than using a point estimate.

We also assume that our application demands online change point detection: An autonomous robot must detect change points in its sensors online, an automatic trading system must detect change points as soon as possible, and electronic monitoring applications must alert to change points in measurements as soon as possible. Many of the previous Bayesian approaches to CPD have been retrospective, where the central aim is to infer change point locations in batch mode [Barry and Hartigan, 1993; Xuan and Murphy, 2007]. While these methods are useful for analyzing a variety of time series data sets, they are not designed for online prediction systems that need to adapt predictions in light of incoming regime changes. Examples of such systems include dialog systems, autonomous navigation systems, and adaptive compression algorithms, to name a few.

The Bayesian Online CPD (BOCPD) algorithm was recently introduced by Adams and MacKay [2007], and similar work has been done by Fearnhead and Liu [2007]. Central to the online predictor is the time since the last change point, namely the *run length*. We perform exact online inference about the run length at every time step, given an *underlying predictive model* (UPM) and a *hazard function*. Given all the observations up to time t, $y_1 \dots y_t \in \mathcal{Y}$, the UPM is used to compute $p(y_t|\mathbf{y}_{(r)}, \theta_m)$ for any $r \in [0, \dots, (t-1)]$ and $(r) := (t-r){:}(t-1)$. The UPM can be thought of as a simpler base model whose parameters η change at every change point according to fixed hyper-parameters θ_m; for instance, the UPM could be iid Gaussian (TIM) with a different mean and variance within each regime.

We have a great deal of flexibility in determining how the run length affects the change point hazard. The run length $r_t \in \mathbb{N}_0$ is incremented every time step until there is a change point when it is reset to zero. The hazard function $H(r|\theta_h) \in \mathbb{N} \to [0, 1]$ is the prior probability of a change point occurring given a run length r_t. We define the change point vector $c_t \in \{0, 1\}$ to be true (1) if there is a change point at time t and false (0) otherwise. Notice that through $H(r|\theta_h)$ we can specify, a priori, arbitrary duration distributions for parameter regimes.

In prior applications of BOCPD, e.g. Adams and MacKay [2007], it is assumed that data in each regime is iid and from an exponential family UPM with known hyper-parameters. However, many data sets are not well-described by a generative model that is piecewise iid. Indeed, many temporal sequences clearly exhibit pronounced temporal smoothness within each regime. We therefore extend BOCPD to allow for time series UPMs such as GPTS (Section 3.4) and ARGP (Section 3.6). Additionally, prior BOCPD setups treat its hyper-parameters, $\theta := \{\theta_h, \theta_m\}$, as fixed and known. Empirically, it is clear that the performance of the algorithm is highly sensitive to hyper-parameter settings. We improve the flexibility of BOCPD by providing a principled mechanism to learn hyper-parameters from the incoming data stream without sacrificing the online nature of the algorithm.

Garnett et al. [2009] added robustness to change points in Gaussian process methods through the use of a "change point kernel." The change point kernel is a non-stationary kernel that allows an abrupt change in some form of behavior in a GPTS model. This approach has three shortcomings compared to BOCPD based approaches: Firstly, the kernel only allows for a single change point at a *known* location. Therefore, to infer the change point location we must integrate over different kernels, which must be done using sampling methods or Bayesian Monte Carlo (BMC). Extensions of this work, such as Álvarez et al. [2010], often resort to optimizing the change point locations with maximum likelihood, which carries an over fitting risk. BOCPD can compute the posterior over change point locations analytically and online. Secondly, the change point kernel only allows for a single change point and often necessitates the use of a sliding window based approach, and thereby introducing another free parameter, to find multiple change points. Third, it narrowly focuses on GPTS based UPMs. BOCPD treats the UPM as a black box and could be an ARGP or even a non-kernel method such as a Dirichlet process.

Notational conventions For simplicity we assume that $r_0 = 0$, meaning there is a change point at $t = 0$, as an initial condition. However, more general initial state distributions are possible in this framework. We define τ_i as the time of the ith change point, where $\tau_0 := 0$. The quantities are related by

$$r_t = \begin{cases} 0 & , c_t = 1 \\ r_{t-1} + 1, & c_t = 0 \end{cases}, \quad \sum_{t=1}^{\tau_i} c_t = i\,, \quad c_{\tau_i} = 1\,. \tag{5.1}$$

Therefore, there is a change point at time $t \Leftrightarrow c_t = 1 \Leftrightarrow r_t = 0 \Leftrightarrow \tau_i = t$ for some i, which is exemplified in Table 5.1. Meaning, $\boldsymbol{\tau}$ is a sorted list of change point

t:	0	1	2	3	4	5	6	7	8	9
$\mathbf{c}$:	1	0	0	0	1	0	1	1	0	0
$\mathbf{r}$:	0	1	2	3	0	1	0	0	1	2
$\boldsymbol{\tau}$:	τ_0				τ_1		τ_2	τ_3		
		♣	♣	♣	♣	◇	◇	♠	♡	♡

Table 5.1: Example of the relationship between $\mathbf{c}$, $\mathbf{r}$, and $\boldsymbol{\tau}$. We represent which regime the data is in using different card suits (♣, ◇, ♠, ♡).

times. Since the $\mathbf{r}$, $\mathbf{c}$, $\boldsymbol{\tau}$ are discrete and related via a bijection:

$$P(\boldsymbol{\tau}) = P(\mathbf{c}) = P(\mathbf{r}) = \prod_{t=1}^{T} H(r_{t-1}+1)^{c_t}(1-H(r_{t-1}+1))^{1-c_t}. \tag{5.2}$$

We use $\mathbf{c} \in \{0,1\}^T$, $\mathbf{r} \in \mathbb{N}_0^T$, and $\boldsymbol{\tau} \in \{1,\ldots,T\}^M$ to refer to the latent change points and $\tilde{\mathbf{c}}$, $\tilde{\mathbf{r}}$, and $\tilde{\boldsymbol{\tau}}$ to refer to the labeled (observed) change points.[1] Equation (5.1) also applies to $\tilde{\mathbf{c}}$, $\tilde{\mathbf{r}}$, and $\tilde{\boldsymbol{\tau}}$.

We observe y_t for $t \in \{1,\ldots,T\}$. The indexing convention in BOCPD is that if $c_t = 1$ then y_{t+1} is the first data point sampled from the new model and y_t is sampled from the old model. Hence, $P(r_t = 0|\mathbf{y}_{1:t})$ is the probability there is a change point at time t without observing the first data point in the new regime. To include the first data point in the new regime we look at $P(r_t = 0|\mathbf{y}_{1:t+1})$.

We summarize the model using the probability of everything (POE):

$$\begin{aligned}\log p(\mathbf{y}_{1:T}, \mathbf{c}_{1:T}|\theta) &= \log p(\mathbf{y}_{1:T}|\mathbf{c}_{1:T}, \theta_m) + \log P(\mathbf{c}_{1:T}|\theta_h) &(5.3)\\ &= \sum_{i=0}^{M} \log p(\mathbf{y}_{\tau_i+1:\tau_{i+1}}|\theta_m) &(5.4)\\ &+ \sum_{t=1}^{T} \log(H(r_{t-1}+1))c_t + \log(1-H(r_{t-1}+1))(1-c_t)\,. \end{aligned}$$

[1] Since M is itself random, we could more rigorously write $\boldsymbol{\tau} \in \mathcal{P}(1{:}T)$.

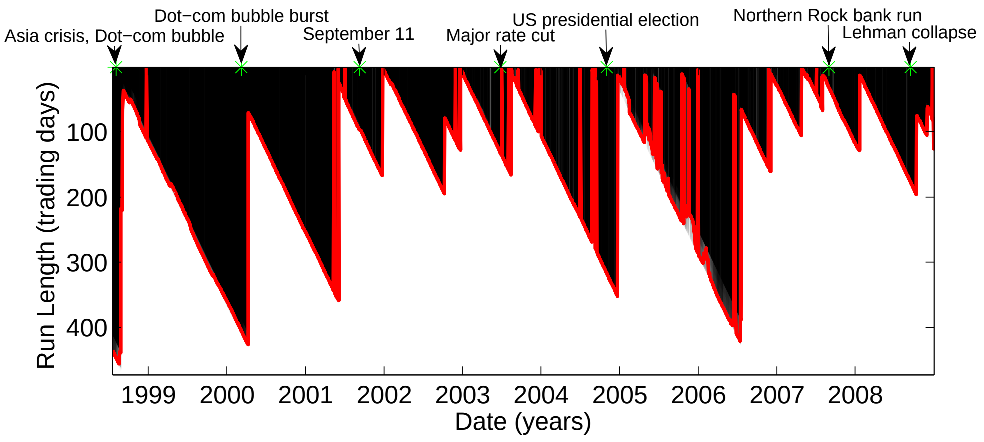


Figure 5.1: **Industry Portfolios**: BOCPD run length distribution in 1998–2008. The shading represents the CDF of the run length distribution, while the solid red line represents the median of the distribution. Areas of a quick transition from black (CDF of zero) to white (CDF of one) indicate a sharply peaked run length distribution. Many events of market impact create change points. Some of the other change points correspond to minor rallies or rate cuts but not a historical event. Note that we plot the filtering distribution, so the vertical spikes are change points BOCPD thinks are plausible but then realizes are false upon receiving more data.

5.1 The BOCPD Algorithm

BOCPD calculates the posterior run length at time t, i.e. $P(r_t|\mathbf{y}_{1:t})$, sequentially. This posterior is used to make online predictions robust to underlying regime changes through marginalization of the run length variable:

$$p(y_{t+1}|\mathbf{y}_{1:t}) = \sum_{r_t} p(y_{t+1}|\mathbf{y}_{1:t}, r_t)P(r_t|\mathbf{y}_{1:t}) \tag{5.5}$$

$$= \sum_{r_t} p(y_{t+1}|\mathbf{y}_{(r)})P(r_t|\mathbf{y}_{1:t}) \,, \tag{5.6}$$

where $\mathbf{y}_{(r)}$ refers to the last r_t observations of y, and $p(y_{t+1}|\mathbf{y}_{(r)})$ is computed using the UPM.

The run length posterior is found by normalizing the joint likelihood:

$$P(r_t|\mathbf{y}_{1:t}) = \frac{p(r_t, \mathbf{y}_{1:t})}{\sum_{r_t} p(r_t, \mathbf{y}_{1:t})} \,. \tag{5.7}$$

The joint likelihood is updated online using a recursive message passing scheme

$$\chi_t := p(r_t, \mathbf{y}_{1:t}) = \sum_{r_{t-1}} p(r_t, r_{t-1}, \mathbf{y}_{1:t}) \tag{5.8}$$

$$= \sum_{r_{t-1}} p(r_t, y_t|r_{t-1}, \mathbf{y}_{1:t-1}) p(r_{t-1}, \mathbf{y}_{1:t-1}) \tag{5.9}$$

$$= \sum_{r_{t-1}} \underbrace{P(r_t|r_{t-1})}_{\text{hazard}} \underbrace{p(y_t|r_{t-1}, \mathbf{y}_{(r)})}_{\text{UPM}} \underbrace{p(r_{t-1}, \mathbf{y}_{1:t-1})}_{\chi_{t-1}} \,. \tag{5.10}$$

This defines a forward message passing scheme to recursively calculate χ_t from χ_{t-1}. The conditional can be restated in terms of messages as $P(r_t|\mathbf{y}_{1:t}) \propto \chi_t$. All the distributions mentioned so far are implicitly conditioned on the set of hyper-parameters θ.

Example BOCPD model A simple example of BOCPD would be to use a constant hazard function $H(r|\theta_h) := \theta_h$, meaning $P(r_t = 0|r_{t-1}, \theta_h)$ is independent of r_{t-1} and is constant, giving rise, a priori, to geometric inter-arrival times for change points. We can use time TIM (1.2) as the UPM, which is the predictive distribution obtained when placing a normal-inverse-gamma prior on iid Gaussian observations (i.e., a Student's t predictive):

$$y_t \sim \mathcal{N}(\mu, \sigma^2) \,, \tag{5.11}$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma^2/\kappa) \,, \quad \sigma^{-2} \sim \text{Gamma}(\alpha, \beta) \,. \tag{5.12}$$

In this example the parameters are $\eta := \{\mu, \sigma^2\}$ and the model hyper-parameters are $\theta_m := \{\mu_0, \kappa, \alpha, \beta\}$. A new value for μ and σ^2 are sampled at each change point. To illustrate this setup, we plot synthetic data sampled from BOCPD in Figure 5.2.

We preview results from applying BOCPD with a variant of TIM as the UPM to the industry portfolios data in Figure 5.1. We see that the inferred change points have a correspondence with historical events.

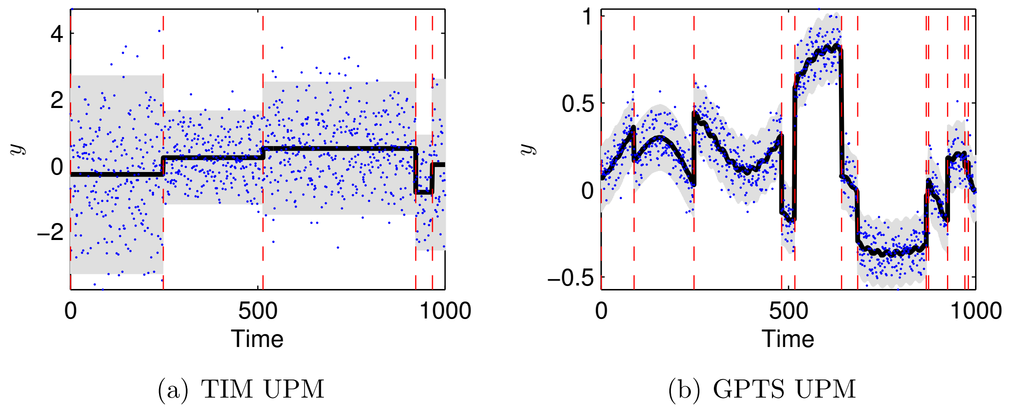


Figure 5.2: We plot synthetic data sampled from a BOCPD model. The blue dots represent the data points sampled from the black mean function and 95% error bars represented by the shaded area. The dashed red vertical lines represent the change points. On the left figure we show a time series y sampled from a TIM base model (UPM). By contrast, on the right we see a time series sampled from a GPTS UPM (SE and periodic covariance functions), which shall be one of the contributions of this chapter.

The posterior on η is updated at every new data point for each run length (each possible starting point for the data). We consider fully Bayesian integration of η while using the type-II MLE for the hyper-parameters θ. Since θ does not change at change points its posterior $p(\theta|\mathbf{y}_{1:T})$ will become peaked in large T allowing for a MLE approximation for computational convenience. As in related work, the Bayesian in BOCPD refers to the integration of parameters η (μ, σ^2).

5.1.1 Hyper-Parameter Learning

We can evaluate the (log) marginal likelihood of the BOCPD model at time T, as it can be decomposed into the one-step-ahead predictive likelihoods (as is done in Section 4.3.4):

$$\log p(\mathbf{y}_{1:T}|\theta) = \sum_{t=1}^{T} \log p(y_t|\mathbf{y}_{1:t-1}, \theta) \,. \tag{5.13}$$

Hence, we can compute the derivatives of the log marginal likelihood using the derivatives of the one-step-ahead predictive likelihoods. These derivatives are found in the same recursive manner as the predictive likelihoods. Using the derivatives of the UPM, $\frac{\partial}{\partial\theta_m} p(y_t|r_{t-1}, \mathbf{y}_{(r)}, \theta_m)$, and those of the hazard function, $\frac{\partial}{\partial\theta_h} P(r_t|r_{t-1}, \theta_h)$, the derivatives of the one-step-ahead predictors are propagated forward using the chain rule, as shown in Algorithm 12. The derivatives with respect to the hyper-parameters are plugged into a conjugate gradient optimizer to perform hyper-parameter learning. An alternative approach to computing the marginal likelihood of BOCPD is shown in Section 5.3.4.

Algorithm 12 Learning (Lines marked with ⋆) and Run Length Estimation in BOCPD. Lines marked with † directly call the UPM.

```
1:  function GETLIKELIHOOD($\mathbf{y} \in \mathbb{R}^T, \theta_m, \theta_h$)   ▷ All multiplications in function are element-wise
2:      $(\boldsymbol{\chi}_0, \partial_h\boldsymbol{\chi}_0, \partial_m\boldsymbol{\chi}_0) \leftarrow (1, 0, 0)$   ▷ Initialize the recursion, set hazard and UPM deriv. to 0
3:      † Initialize $\mathcal{S}$ to sufficient statistics of UPM prior
4:      Define $\tilde{\boldsymbol{\chi}}_t$ as $\boldsymbol{\chi}_t(2{:}t+1)$
5:      for $t = 1$ to $T$ do
6:          † $\boldsymbol{\pi}_{(r)} \leftarrow p(y_t|\mathcal{S})$
7:          $\mathbf{h} \leftarrow H(1{:}t)$
8:          $\tilde{\boldsymbol{\chi}}_t \leftarrow \boldsymbol{\chi}_{t-1}\boldsymbol{\pi}_{(r)}(1-\mathbf{h})$   ▷ Update the messages, no new change point, (5.10)
9:          ⋆ $\partial_h\tilde{\boldsymbol{\chi}}_t \leftarrow \boldsymbol{\pi}_{(r)}(\partial_h\boldsymbol{\chi}_{t-1}(1-\mathbf{h}) - \boldsymbol{\chi}_{t-1}\partial_h\mathbf{h})$
10:         ⋆ $\partial_m\tilde{\boldsymbol{\chi}}_t \leftarrow (1-\mathbf{h})(\partial_m\boldsymbol{\chi}_{t-1}\boldsymbol{\pi}_{(r)} + \boldsymbol{\chi}_{t-1}\partial_m\boldsymbol{\pi}_{(r)})$
11:         $\boldsymbol{\chi}_t(1) \leftarrow \sum \boldsymbol{\chi}_{t-1}\boldsymbol{\pi}_{(r)}\mathbf{h}$   ▷ Update the messages, there is a new change point, (5.10)
12:         ⋆ $\partial_h\boldsymbol{\chi}_t(1) \leftarrow \sum \boldsymbol{\pi}_{(r)}(\partial_h\boldsymbol{\chi}_{t-1}\mathbf{h} + \boldsymbol{\chi}_{t-1}\partial_h\mathbf{h})$
13:         ⋆ $\partial_m\boldsymbol{\chi}_t(1) \leftarrow \sum \mathbf{h}(\partial_m\boldsymbol{\chi}_{t-1}\boldsymbol{\pi}_{(r)} + \boldsymbol{\chi}_{t-1}\partial_m\boldsymbol{\pi}_{(r)})$
14:         $P(r_t|y_{1:t}) \leftarrow$ normalize $\boldsymbol{\chi}_t$
15:         † Update sufficient statistics of posteriors $\mathcal{S}$
16:     end for
17:     $p(y_{1:T}) \leftarrow \sum \boldsymbol{\chi}_T$   ▷ $1 \times 1$ Calculate the Evidence, message normalization constant
18:     ⋆ $\partial p(y_{1:T}) \leftarrow (\sum \partial_h\boldsymbol{\chi}_T, \sum \partial_m\boldsymbol{\chi}_T)$
19:     return $(p(y_{1:T}), \partial p(y_{1:T}))$
20: end function
```

Sufficient statistics In the case of the TIM, we can store the sample moments, $\sum y$ and $\sum y^2$, for each run length and recover the posterior predictive. These quantities are computed in a streaming manner. We need to maintain two variables for each run length and merely accumulate y and y^2 at each time step. The predictive distribution requires the sample mean and sample variance. We can construct this from the sample moments

$$\frac{1}{t}\sum_{i=1}^{t}\left(y_i - \frac{1}{t}\sum_{j=1}^{t} y_j\right)^2 = \frac{1}{t}\sum_{i=1}^{t} y_i^2 - \left(\frac{1}{t}\sum_{i=1}^{t} y_i\right)^2 \tag{5.14}$$

$$\implies \widehat{\mathrm{Var}\,[y]} = \widehat{\mathbb{E}\,[y^2]} - \widehat{\mathbb{E}\,[y]}^2 . \tag{5.15}$$

This streaming method of computing the sample variance is known to be prone to numerical problems. For instance if the mean is 10^6 and the variance is 10^{-3}, we will be subtracting two large numbers to get a small number in (5.15), which is a classic case of *catastrophic cancellation*, a numerically unstable operation [Chan and Lewis, 1979]. During training, we avoid this by standardization. In the test set during streaming operation we do not know the scale of the data. However, if we know the approximate scale of the data from the training set we can standardize the data sufficiently to avoid gross numerical problems.

Once we have the sufficient statistics $\mathcal{S}$ for each run length we get the posterior on η for each run length as

$$\mu_t = \frac{\kappa_0\mu_0 + \sum y}{\kappa_0 + t} \in \mathbb{R}\,, \quad \kappa_t = \kappa_0 + t \in \mathbb{R}^+\,, \tag{5.16}$$

$$\alpha_t = \alpha_0 + \frac{1}{2}t \in \mathbb{R}^+\,, \quad \beta_t = \beta_0 + \frac{1}{2}\mathrm{SSE} + \frac{\kappa_0 t(\bar{y} - \mu_0)^2}{2(\kappa_0 + t)} \in \mathbb{R}^+\,, \tag{5.17}$$

$$\mathrm{SSE} := \sum_{i=1}^{t}(y_i - \bar{y})^2 = \sum y^2 - \frac{1}{t}(\sum y)^2 \in \mathbb{R}^+ \tag{5.18}$$

which gives a posterior predictive distribution on y_{t+1} of

$$p(y_{t+1}|\mathbf{y}_{(r)}) = \mathrm{St}_{2\alpha}\left(\mu, \frac{\beta}{\alpha}\frac{\kappa+1}{\kappa}\right) . \tag{5.19}$$

In the multivariate case, we use either the independent factor model (IFM) or a full joint normal. In the IFM, each dimension of $\mathbf{y}_t \in \mathbb{R}^D$ is considered conditionally independent given the parameters η. Therefore, we only need to accumulate $\mathbf{y}_t$ and $\mathbf{y}_t^2$ independently on each dimension. We also apply (5.18) and (5.19) to each dimension independently.

In the joint case, $\mathbf{y}_t \in \mathbb{R}^D$ is distributed from a normal with a full covariance matrix, where the mean and covariance changes at each change point. We place a Gaussian-Wishart prior[1] on the mean vector and precision matrix:

$$\boldsymbol{\Sigma}^{-1} \sim \mathcal{W}_{\nu_0}(\boldsymbol{\Lambda}_0) \in \mathbb{S}^D \,, \tag{5.20}$$

$$\boldsymbol{\mu}|\boldsymbol{\Sigma} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}/\kappa_0) \in \mathbb{R}^D \,, \tag{5.21}$$

$$\mathbf{y}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^D \,. \tag{5.22}$$

Therefore, in this case we must accumulate $\sum \mathbf{y}$ and $\sum \mathbf{y}\mathbf{y}^\top$. In the full covariance case, $\eta = \{\mu, \boldsymbol{\Sigma}\}$ and $\theta_m = \{\boldsymbol{\mu}_0, \kappa_0, \nu_0, \boldsymbol{\Lambda}_0\}$. Following Gelman et al. [2004, Ch. 3], we get updates of

$$\boldsymbol{\mu}_t = \frac{\kappa_0}{\kappa_0 + t}\mu_0 + \frac{t}{\kappa_0 + t}\bar{\mathbf{y}} \in \mathbb{R}^D \,, \quad \kappa_t = \kappa_0 + t\mathbb{R}^+ \,, \quad \nu_t = \nu_0 + t \in \mathbb{R}^+ \,, \tag{5.23}$$

$$\boldsymbol{\Lambda}_t^{-1} = \boldsymbol{\Lambda}_0^{-1} + \mathrm{SSE} + \frac{\kappa_0 t}{\kappa_0 + t}(\bar{\mathbf{y}} - \boldsymbol{\mu}_0)(\bar{\mathbf{y}} - \boldsymbol{\mu}_0)^\top \in \mathbb{S}^D \,, \tag{5.24}$$

$$\mathrm{SSE} = \sum_{i=1}^{t}(\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^\top = \sum \mathbf{y}\mathbf{y}^\top - \frac{1}{t}\left(\sum \mathbf{y}\right)\left(\sum \mathbf{y}\right)^\top \in \mathbb{S}_0^D \,. \tag{5.25}$$

This also gives a Student's t predictive:

$$p(\mathbf{y}_{t+1}|\mathbf{y}_{(r)}) = \mathrm{St}_{\nu_t - d + 1}\left(\boldsymbol{\mu}_t, \frac{\boldsymbol{\Lambda}_t^{-1}}{\nu_t - d + 1}\frac{\kappa + 1}{\kappa}\right) \,. \tag{5.26}$$

Scaling factors In practice the messages χ, the joint of the data and run length probability, will become very small in large T. This leaves the potential for numerical underflow issues. We correct this in one of two ways: by scaling factors or moving the messages to log scale. To use scaling factors we merely renormalize

[1] A Wishart distribution on the precision matrix $\boldsymbol{\Lambda} \in \mathbb{S}^D$ takes the form $\mathcal{W}_n(\boldsymbol{\Lambda}|\mathbf{V}) \propto |\boldsymbol{\Lambda}|^{\frac{n-D-1}{2}} \exp(-\frac{1}{2}\mathrm{tr}(\mathbf{V}^{-1}\boldsymbol{\Lambda}))$ with expectation $\mathbb{E}[\boldsymbol{\Lambda}] = n\mathbf{V}$. If the precision matrix is of dimension one it corresponds to a Gamma distribution with parameters $\alpha = n/2$ and $\boldsymbol{\beta} = \mathbf{V}^{-1}/2$.

χ every iteration on line 14 of Algorithm 12. Then, the model evidence is the product of the normalization constants at every iteration rather than the normalization constant on χ at the end of the algorithm. The normalizer on χ is now the one-step-ahead predictive probability. Alternatively, we can move χ to log scale. The product in line 8 becomes a sum, and the summation in line 11 is implemented with the *log-sum-exp trick* (see Section A.3). Log scale is more robust during learning when an initial setting of the hyper-parameters is potentially so bad that we get numerical underflow, from small predictive probabilities, even when rescaling every iteration. Scaling factors and the log-sum-exp trick were compared in the context of HMMs in Mann [2006].

5.1.2 Improving Efficiency

In a historical context, it is surprising that the Bayesian updating in BOCPD can be done analytically. Many of the previous approaches to Bayesian segmentation, such as Barry and Hartigan [1993], required at best particle filtering if not reversible jump MCMC (RJ-MCMC), where it is notoriously difficult to obtain convergence.[1] As first noticed by Fearnhead [2006], it was falsely assumed that if we did not know the number of change points in advance, we had to allow for a change parameter space as in general model comparison. This type of situation is a classic example of when RJ-MCMC is necessary. However, the special temporal structure in BOCPD allows for the dynamic programming solution.

Although the exact inference algorithm in BOCPD is a great improvement over RJ-MCMC, there is still room for improvement. The posterior $P(r_t|\mathbf{y}_{1:t})$ forms a vector of length t, which requires $t+1$ updates to propagate it to the next time step. Each update in TIM (or any parametric exponential family) can be done in a streaming manner, requiring $\mathcal{O}(1)$ computation, exact BOCPD inference on the entire time series is $\mathcal{O}(T^2)$. In practice, the run length distribution will be highly peaked. We make the approach more efficient by *pruning* out run lengths with probability below a threshold, e.g. $\epsilon = 10^{-3}$, or considering only the K most probable run lengths and setting the remaining probabilities to zero.

[1] Professors will often refer to their graduate students as having a “driver’s license” for RJ-MCMC. Furthermore, you may hear them refer to crashes causing the students to “lose their license.”

These modifications run in $\mathcal{O}(T/\epsilon)$ and $\mathcal{O}(TK)$, respectively.

Relationship to HMMs There exist some overlap between HMMs and BOCPD. Any BOCPD model with a finite number of possible UPM parameters η and a constant hazard is equivalent to an HMM. Likewise, any HMM where the transition matrix has identical rows, i.e. there is no state dependence in the transitions, is a BOCPD model. We must select the number of states Q in an HMM. The computational complexity in an HMM grows $\mathcal{O}(Q^2T)$.

If we further consider Hidden semi-Markov models (HSMMs) [Ferguson, 1980] then we can create more overlap by alleviating the constant hazard requirement. However, HSMMs will still require a finite number of states, whereas BOCPD can have an (uncountably) infinite number of possible UPM parameter settings η. Furthermore, in HSMMs a duration D must arbitrarily be specified on regime duration, which makes them less appropriate for settings where regime durations are long, regime changes are rate, or there are large number of (possibly novel) states. The complexity of an HSMM is $\mathcal{O}(D^2Q^2T)$, although this can often be reduced to $\mathcal{O}(DQ^2T)$ [Johnson, 2005].

5.2 Change Point Detection with External Inputs

The change point detection model does not have to be completely generative. We can allow external inputs $\mathbf{x} \in \mathbb{R}^D$ into the UPM. The clear extension of the TIM to include external inputs would be a changing Bayesian linear regression (BLR) model:

$$y_t \sim \mathcal{N}(\mathbf{x}_t^\top \mathbf{w}, \sigma_n^2)\,, \tag{5.27}$$

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_0, \sigma_n^2 \mathbf{S})\,, \quad \sigma_n^{-2} \sim \mathrm{Gamma}(\alpha_0, \beta_0)\,. \tag{5.28}$$

For simplicity we set $\boldsymbol{\mu}_0 = \mathbf{0} \in \mathbb{R}^D$ and $\mathbf{S} = \kappa_0^{-1}\mathbf{I} \in \mathbb{S}^D$. We summarize the parameters as $\eta = \{\mathbf{w}, \sigma_n^2\}$ and the hyper-parameters as $\theta_m = \{\kappa_0, \alpha_0, \beta_0\}$.

In this case we must accumulate $\mathbf{x}\mathbf{x}^\top \in \mathbb{R}^{D\times D}$ and $\mathbf{x}y \in \mathbb{R}^D$. As shown in

Rasmussen and Williams [2006, Ch. 2], the predictive distribution given these sufficient statistics for a known noise variance is

$$p(y_{t+1}|\mathbf{y}_{(r)}) = \mathcal{N}(\sigma_n^{-2}\mathbf{x}_{t+1}^\top \mathbf{A}^{-1}\mathbf{X}\mathbf{y}_{(r)}, \mathbf{x}_{t+1}^\top \mathbf{A}^{-1}\mathbf{x}_{t+1} + \sigma_n^2)\,, \tag{5.29}$$

$$\mathbf{A} := \sigma_n^{-2}(\mathbf{X}\mathbf{X}^\top + \kappa_0 \mathbf{I}) \in \mathbb{S}^D\,, \tag{5.30}$$

where $\mathbf{X} \in \mathbb{R}^{D\times r}$ is the matrix of the most recent inputs and $\mathbf{y}_{(r)} \in \mathbb{R}^r$ is the vector of the most recent outputs. In this formulation, we must compute $\mathbf{A}^{-1}\mathbf{X}\mathbf{y}_{(r)}$, which is $\mathcal{O}(D^3)$, but that should be negligible if $D \ll T$. To update the sufficient statistics $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}\mathbf{y}$, for each run length, we must apply the update rule for every new data point

$$\mathbf{X}\mathbf{X}^\top \leftarrow \mathbf{X}\mathbf{X}^\top + \mathbf{x}_t\mathbf{x}_t^\top\,, \tag{5.31}$$

$$\mathbf{X}\mathbf{y} \leftarrow \mathbf{X}\mathbf{y} + \mathbf{x}_t y_t\,. \tag{5.32}$$

Recall that BLR is a special case of a GP, therefore we can extend BLR to the scale with output scale uncertainty using (3.38) [Minka, 2001]. In BLR, adding an uncertain noise output is effectively the same as output scale uncertainty in a GP since the prior on the weights are rescaled according to the noise:

$$p(y_{t+1}|\mathbf{y}_{(r)}) = \mathrm{St}_{2\alpha}\left(\mu_\star, \frac{\beta}{\alpha}\sigma_\star^2\right)\,, \tag{5.33}$$

$$\alpha = \alpha_0 + \frac{r}{2}\,, \quad \beta = \beta_0 + \frac{1}{2}\mathbf{y}_{(r)}^\top \mathbf{K}^{-1}\mathbf{y}_{(r)} \overset{(3.28)}{=} \beta_0 \frac{1}{2}\mathrm{SSE}\,, \tag{5.34}$$

where $\mathbf{K}$ is the implicit GP kernel from BLR. Using (3.28) we keep track of β by accumulating the SSE avoiding the need to compute a quadratic matrix form every time step.[1] Also recall that BLR in a feature space, i.e. using E basis functions, is also a GP, therefore we can apply (5.33) to BLR with basis functions.

[1] We do not need to consider BLR as a special case of GP to do these calculations, but doing so reduces redundant derivation.

In this case we must accumulate $\mathbf{\Phi}(\mathbf{x})\mathbf{\Phi}(\mathbf{x})^\top$, $\mathbf{\Phi}(\mathbf{x})y$, and the SSE

$$p(y_{t+1}|\mathbf{y}_{(r)}) = \mathcal{N}(\sigma_n^{-2}\boldsymbol{\phi}(x_{t+1})^\top\mathbf{A}^{-1}\mathbf{\Phi}\mathbf{y}_{(r)}, \phi(\mathbf{x}_{t+1})^\top\mathbf{A}^{-1}\phi(\mathbf{x}_{t+1}))\,, \tag{5.35}$$
$$\mathbf{A} := \sigma_n^{-2}(\mathbf{\Phi}\mathbf{\Phi}^\top + \kappa_0\mathbf{I}) \in \mathbb{S}^E\,, \tag{5.36}$$

where $\mathbf{\Phi} := \phi(\mathbf{X}) \in \mathbb{R}^{E\times r}$ is the basis function representation of each of the past inputs $\mathbf{x}$. We can keep the $\boldsymbol{\mu}_0 = 0$ simplification without losing any generality by adding a constant basis function where $\phi(\mathbf{x}) = 1$. We extend this to multivariate $\mathbf{y}_t$ by applying it to each output dimension independently; or, in analogy to the full covariance TIM UPM, we could use a linear model with noise outputs that covary with an unknown covariance matrix. We can get predictive equations using Minka [2001]. Just as in the TIM case, standardization issues that could lead to numerical problems apply to each of the UPMs presented here. Therefore, standardization in both $\mathbf{y}$ and $\mathbf{x}$ is also a good measure to take in the external inputs case as well.

5.3 Gaussian Process Change Point Detection

In this section, we first extend BOCPD by implementing UPMs that exploit temporal structure within each regime, using two time series models based on Gaussian processes (GPs). As mentioned in Chapter 3 there are two ways to model a time series with a GP: GPTS and ARGP. GPs can be used to model time series data directly (i.e. the mapping $\mathcal{T} \to \mathcal{Y}$, where $\mathcal{T}$ is the set of time indices). This gives rise to a GP time series (GPTS) model. Alternatively, they can be used to learn the mapping between observations y_t and y_{t+1} to produce a nonlinear autoregressive (AR) model (an ARGP). GPs are an attractive choice for time series UPMs because we can integrate out the functions representing the GPTS and ARGP mappings, thus improving predictive performance.

A similar sequential nonstationary GPTS model was introduced in Garnett et al. [2009]. In this approach GPs are made robust to incoming change points through the introduction of an additional hyper-parameter in the covariance function, which represents the location at which a change point occurs inside a past window of data. The size of the window is prespecified and it is implicitly assumed that there can only be one change point inside it. Next-step predictions are improved using hyper-parameter marginalization (this includes the change point hyper-parameter). As the required integral is intractable for a GP model, Bayesian Monte Carlo (BMC) [Rasmussen and Ghahramani, 2003] is used to perform *quadrature* integration. BMC requires defining a separate GP to model the marginal likelihood surface in addition to the GP in the model itself. This is problematic because the marginal likelihood function is *positive* by definition. Hence the approximation by a GP, whose range is the entire real line, can be poor. This shortcoming gets worse as the dimensionality of the hyper-parameter space increases, which occurs when we apply change point detection to higher-dimensional time series data.

In principle, GPIL and its variants could also be used [Deisenroth et al., 2009; Turner et al., 2010; Wang et al., 2008], but inference in Gaussian process state space models is computationally intensive even without change points so we do not apply it here.

When using a GPTS UPM we have the option of integrating out the output scale. If we have an uncertain output scale and an additive constant term in the covariance, the TIM UPM forms a *nested model* with respect to GPTS UPM. Using GPTS, defined in (3.96), we either have a Gaussian predictive (3.15)

$$p(y_t|\mathbf{y}_{(r)}, \theta_m) = \mathcal{N}(\mu_t, \sigma_t^2)\,, \tag{5.37}$$

$$\mu_t = \mathbf{K}_\star^\top \mathbf{K}^{-1} \mathbf{y}_{(r)}\,, \quad \sigma_t^2 = \mathbf{K}_{\star\star} - \mathbf{K}_\star^\top \mathbf{K}^{-1} \mathbf{K}_\star\,, \tag{5.38}$$

$$\mathbf{K} := k((r),(r))\,, \quad \mathbf{K}_\star := k((r),t)\,, \quad \mathbf{K}_{\star\star} := k(t,t)\,, \tag{5.39}$$

or a Student's t predictive (3.38) if we are integrating out the output scale:

$$p(y_t|\mathbf{y}_{(r)}, \theta_m) = \mathrm{St}_{2\alpha_t}\left(\mu_t, \frac{\beta_t}{\alpha_t}\sigma_t^2\right)\,, \tag{5.40}$$

$$\alpha_t = \alpha_0 + \frac{r}{2}\,, \quad \beta_t = \beta_0 + \frac{1}{2}\mathbf{y}_{(r)}^\top \mathbf{K}^{-1} \mathbf{y}_{(r)}\,. \tag{5.41}$$

The UPM hyper-parameters θ_m are the GP hyper-parameters, as well as the output scale prior if applicable. In the case of a SE covariance for GPTS $\theta_m = \{\ell, \sigma_0^2, \sigma_n^2, \alpha_0\}$ we assume $\beta_0 = 1$ since we can rescale the output hyper-parameters

σ_0^2 and σ_n^2 for the same effect. This model is not necessarily more general than a TIM UPM since TIM allows for a changing mean at each change point. Therefore, we add a constant covariance term σ_c^2, which as shown in (3.9) effectively integrates out an unknown mean. We now have a GPTS model, with changing mean and variance in each regime like TIM, but with the added flexibility of temporal correlations. Notice that under such a model the GP hyper-parameters, other than output scale, are assumed to be fixed *across* different regimes. Meaning, the length scale ℓ does not change at each change point. If we desire to model changes in the GP hyper-parameters at every change point, then the BOCPD algorithm dictates that we should integrate them out within the UPM.

5.3.1 Methods to Improve Execution Time

Efficient computation is trickier with GPs than iid UPMs, and the computational bottleneck is the posterior predictive from the UPM $p(y_t|r_{t-1}, \mathbf{y}_{(r)})$ rather than updating the run length distribution (5.10). Prediction in GPs is $\mathcal{O}(T^3)$ with T training points due to computing the inverse of the covariance matrix. Therefore, if we naively recomputed the GP every time step it would run $\mathcal{O}(T^5)$. For GPTS UPMs and uniformly sampled data, we can put the prediction in AR form using Toeplitz methods, making it $\mathcal{O}(T)$ per prediction, or $\mathcal{O}(T^3)$ total. If the covariance function allows GPK then it is $\mathcal{O}(1)$ per prediction or $\mathcal{O}(T^2)$ total. We bring this down further with pruning to $\mathcal{O}(T/\epsilon)$ or $\mathcal{O}(TK)$. An efficient implementation of GPTS is shown in Algorithm 13. If we do not care about output scale uncertainty we replace line 9 with: $\mathcal{N}(y_t|\mu, \boldsymbol{\sigma}^2(1{:}t))$; the SSE is not necessary in that case.

If we are not interested in online computation, such as batch training in hyper-parameter learning, we can further improve performance (as mentioned in Section 3.4.1). The main computational bottleneck in Algorithm 13 is line 6. Note that this takes the form of a convolution. If we define an $\mathbf{M} \in \mathbb{U}^T$ matrix where $\mathbf{M}(r,t)$ is the one-step-ahead predictive mean at time t with run length r, we can precompute it as:

$$\mathbf{M}(r,t) := \mathbb{E}\left[y_{t+1}|\mathbf{y}_{(r)}\right] \implies \mathbf{M} = \boldsymbol{\alpha}\mathrm{Toeplitz}(\mathbf{y}_{1:T})\,. \tag{5.42}$$

We must pad $\mathbf{M}$ with a starting column of zeros and first row of zeros to account for the cases of predicting y_1 and predictions with run length zero. The use of a matrix by Toeplitz matrix multiplication allows for FFT, an $\mathcal{O}(T \log T)$ operation, based calculations.

We must use different tricks to improve the efficiency of the ARGP UPM. Using the sub-evidence rank-1 methods from Section 3.5 we bring the computational cost for computing the log marginal likelihood for every starting and end point down to $\mathcal{O}(T^3)$. This is precisely what is needed for efficient computation in an ARGP UPM. If pruning is applied the complexity is reduced further to $\mathcal{O}(T\tilde{R}^2)$ or $\mathcal{O}(T\tilde{R}^2)$, where $\tilde{R}$ is the typical max run length not pruned out. Recall from Section 3.5 that once we have the Cholesky factorization of the covariance for the last $\tilde{R}$ points we get the evidence of the intervening points for free. Therefore, it is only necessary to prune out a maximum run length, not remove run length hypothesis that are very unlikely yet of a small run length than the maximum considered.

The BOCPD recursions require the log predictive probabilities, which are found by differencing the column of $\log \mathbf{P}$ from Algorithm 4 since the log predictive probability is the change in the log evidence upon adding a new data point. Similar methods to the sub-evidence are used to find the predictive mean and variance for each run length without increasing the computational complexity. If we would like to achieve online computation we must adapt the sub-evidence methods and reverse the ordering of the rows and columns in the kernel matrix $\mathbf{K}$. We summarize this methodology for the ARGP UPM in Algorithm 15. We utilize a function $\mathrm{ARsplit}_p(\mathbf{y})$, see (A.31), that takes a vector of a time series and returns a matrix with the time series lagged by each order up to p. This feature matrix allows us to treat autoregression as a standard regression problem.

Forecasting Once we have estimated the run length we would like to be able to forecast into the future. We demonstrate the computations required for forecasting at all horizons from one to $f \in \mathbb{N}$ in Algorithm 16. For an iid model like TIM, conditional on the run length, our prediction of the next point will be the same as the prediction of f points into the future. However, to get a marginal prediction we have to weight each of these predictions against the run length

Algorithm 13 Efficient GPTS UPM implementation

```
1: function GPTS-CP($\mathbf{y} \in \mathbb{R}^T, \theta_m, \theta_h$)    ▷ $\theta_m$ includes $\alpha_0$ and $\beta_0$
2:     $R \leftarrow 1$
3:     SSE $\leftarrow 2\beta_0$ ▷ Initialize SSE to account for contribution from gamma prior
4:     Precompute $\alpha$ matrix and predictive variances $\sigma^2$ using Algorithm 14
5:     for $t = 1$ to $T$ do    ▷ Find the predictive distribution $p(\mathbf{y}(t)|\mathbf{y}(1{:}t-1), r_{t-1} = r-1)$
6:         $\boldsymbol{\mu} \leftarrow \alpha(1{:}t, 1{:}t-1)\mathbf{E}\mathbf{y}(1{:}t-1)$
7:         df $\leftarrow 2\alpha_0 + (0{:}t-1)^\top$
8:         $\boldsymbol{\sigma}_\star^2 \leftarrow \boldsymbol{\sigma}^2(1{:}t) \odot \mathrm{SSE}(1{:}t) \oslash \mathrm{df}$    ▷ Uses element-wise division $\oslash$
9:         $\boldsymbol{\pi} \leftarrow \mathrm{St}_{\mathrm{df}}(\mathbf{y}(t)|\boldsymbol{\mu}, \boldsymbol{\sigma}_\star^2)$
                                              ▷ Keep track of the SSE in streaming manner
10:        $\mathrm{SSE}(2{:}t+1) \leftarrow \mathrm{SSE}(1{:}t) + (\boldsymbol{\mu} - \mathbf{y}(t))^2 \oslash \boldsymbol{\sigma}^2(1{:}t)$
11:        $\mathrm{SSE}(1) \leftarrow 2\beta_0$
                                    ▷ Do the ordinary BOCPD updates as in Algorithm 12
12:        $\mathbf{R}(2{:}t+1, t+1) \leftarrow \mathbf{R}(1{:}t, t) \odot \boldsymbol{\pi} \odot (1 - H(1{:}t))$
13:        $\mathbf{R}(1, t+1) \leftarrow \sum \mathbf{R}(1{:}t, t) \odot \boldsymbol{\pi} \odot H(1{:}t)$
14:        $\mathbf{Z}(t) \leftarrow \sum \mathbf{R}(1{:}t+1, t+1)$
15:        $\mathbf{R}(1{:}t+1, t+1) \leftarrow \mathbf{R}(1{:}t+1, t+1)/\mathbf{Z}(t)$    ▷ Use scaling factor
16:    end for
17:    return $\mathbf{R}$ and $\mathbf{Z}$
18: end function
```

Algorithm 14 Efficient precomputation of GP predictions

```
1: function GP-PRECOMPUTATION($T \in \mathbb{N}, \theta_m$)
2:     Find $\mathbf{K}_{\star\star} = k(0,0)$ and $\mathbf{K}_\star = k(1{:}T-1, 0)$
3:     Find $\alpha \in \mathbb{L}^{T-1}$ such that $i$th row of $\alpha$ is solution to Yule-Walker equations after $i$ iterations
4:     $\boldsymbol{\sigma}^2 \leftarrow \mathbf{K}_{\star\star} - \alpha\mathbf{K}_\star \in (\mathbb{R}^+)^{T-1}$
5:     Insert row of zeros to top of $\alpha$
6:     Insert $\mathbf{K}_{\star\star}$ to $\boldsymbol{\sigma}^2(1)$
7:     return $\alpha \in \mathbb{R}^{T \times T-1}$ and $\boldsymbol{\sigma}^2 \in (\mathbb{R}^+)^T$
8: end function
```

Algorithm 15 Efficient ARGP UPM implementation

```
1:  function ARGP-CP($\mathbf{y} \in \mathbb{R}^T, \theta_m, \theta_h, p \in \mathbb{N}_0$)
2:      $(\mathbf{L}, \text{nlml}_0, \boldsymbol{\alpha}, \mathbf{t}_\alpha, \mathbf{t}_\beta) \leftarrow [\emptyset]$
3:      $\mathbf{X} \leftarrow \text{ARsplit}(\mathbf{y})$
4:      $\tilde{R} \leftarrow 1$
5:      for $t = 1$ to $T$ do
6:          $\mathbf{K}_{\star\star} \leftarrow k(\mathbf{X}(t), \mathbf{X}(t))$
7:          $\mathbf{K}_{\star} \leftarrow k(\mathbf{EX}(t - \tilde{R} + 1{:}t - 1), \mathbf{X}(t))$
8:          if we want to get the predictive distribution then
9:              $\mathbf{v} \leftarrow \mathbf{L} \backslash \mathbf{K}_\star$
10:             $\boldsymbol{\mu}^\top \leftarrow \begin{bmatrix} 0 & \mathbf{D}^{-1}(\mathbf{v} \odot \boldsymbol{\alpha}) \end{bmatrix}$
11:             $[\boldsymbol{\sigma}_\star^2]^\top \leftarrow \begin{bmatrix} \mathbf{K}_{\star\star}\beta_0/\alpha_0 & (\mathbf{t}_\beta \oslash \mathbf{t}_\alpha) \odot (\mathbf{K}_{\star\star} - \mathbf{D}^{-1}(\mathbf{v} \odot \mathbf{v})) \end{bmatrix}$
12:             $\text{df}^\top \leftarrow \begin{bmatrix} 2\alpha_0 & 2\mathbf{t}_\alpha \end{bmatrix}$
13:         end if
14:         Update $\mathbf{L}$ using $\mathbf{K}_\star$ and $\mathbf{K}_{\star\star}$ according to Algorithm 4
15:         $\boldsymbol{\alpha} \leftarrow \mathbf{L}\,\mathbf{E} y_{t-\tilde{R}+1:t}$
16:         $(\mathbf{t}, \mathbf{t}_\alpha, \mathbf{t}_\beta) \leftarrow (1{:}\tilde{R}, \alpha_0 + \frac{1}{2}(1{:}\tilde{R}), \beta_0 + \frac{1}{2}\mathbf{D}^{-1}(\boldsymbol{\alpha} \odot \boldsymbol{\alpha}))$
17:         $\text{nlml}_t \leftarrow \mathbf{t}_\alpha \odot \log(\mathbf{t}_\beta/\beta_0) + \mathbf{D}^{-1} \log \text{diag}(\mathbf{L}) - \log\Gamma(\mathbf{t}_\alpha) + \log\Gamma(\alpha_0) + \frac{1}{2}\log(2\pi\beta_0)\mathbf{t}$
18:         $\boldsymbol{\pi} \leftarrow \exp(-(\text{nlml}_t - \begin{bmatrix} 0 & \text{nlml}_{t-1} \end{bmatrix}^\top))$
19:         Do the ordinary BOCPD updates as in Algorithm 12
20:         Set maximum run length $\tilde{R}$ based on pruning criterion
21:         Trim rows/columns to max size $\tilde{R}$ on $\mathbf{R}$, $\mathbf{L}$, $\text{nlml}_t$, $\boldsymbol{\alpha}$, $\mathbf{t}_\alpha$, and $t_\beta$
22:         Renormalize current column of $\mathbf{R}$
23:     end for
24:     return $\mathbf{R}$
25: end function
```

distribution. This run length distribution will change further into the future as it is more likely there will be an intervening change point for large forecast horizons f. More probability mass naturally shifts to the prior predictive of the UPM for longer forecast horizons. We do this by re-weighting the run length according to the hazard function in Line 7 of Algorithm 16. For non-iid UPMs like a GPTS, we also must consider that our predictions, even conditional on the run length, will change with the prediction horizon. Since the predictive variances are invariant to the data $\mathbf{y}_{1:T}$ they can be precomputed.

We would also like to summarize the predictive distributions for each run length for plotting purposes. We summarize the distribution by moment matching to a single Gaussian, using the mixture equations (A.37). Once we have the predictive means conditional on the run length μ and the run length weights $\mathbf{W}$, we get the marginal means by $\mathbf{1}^\top(\mu \odot \mathbf{W}) = \mathrm{diag}(\mu\mathbf{W})$ by (A.1). If we are using an uncertain output scale we will have to adjust the predictive variances exactly as we did for the one-step-ahead predictions in Algorithm 13. We are left with a mixture of Student's t distributions for the forecast, or mixture of Gaussians for certain output scale. If any of the Student's t distributions have fewer than two degrees of freedom the variance of the entire mixture will be undefined. Therefore, the moment matching approach will be inappropriate unless $\alpha_0 > 2$, which ensures the variance will always be defined.

Algorithm 16 Multi-step forecasting step of GPTS UPM

```
1: function GPTS-CP-FORECAST($\mathbf{y} \in \mathbb{R}^{t-1}, \mathbf{K}_{\star\star} \in \mathbb{R}^+, \mathbf{K}_\star \in \mathbb{R}^{t-1\times f}$)
2:     $\mathbf{M} \leftarrow$ Levinson($\mathbf{K}_{\star\star}, \mathbf{K}_\star(1{:}t-2,:), \mathbf{Ey}$) $\in \mathbb{L}^{t-1}$        ▷ Use Algorithm 3
3:     $\boldsymbol{\mu} \leftarrow \mathbf{MK}_\star \in \mathbb{R}^{t-1\times f}$
4:     Insert row of zeros on top of $\boldsymbol{\mu}$
5:     $\tau \leftarrow t$ where $\mathbf{R} \in [0,1]^t$
6:     for $i$ from 1 to forecast horizon $f$ do
7:                     ▷ Put all new mass into mixture weight for prior predictive
8:         $\mathbf{W}(1,i) \leftarrow \mathbf{1}^\top\mathbf{R}(1{:}i)$
9:         $\mathbf{W}(2{:}t,i) \leftarrow \mathbf{R}(i+1{:}\tau+1)$
10:        Increment $\tau$ so that $\mathbf{R} \in [0,1]^\tau$
11:        Set $\mathbf{H}$ to be the hazard of $1{:}\tau$
12:        $(\mathbf{R}(1), \mathbf{R}(2{:}\tau+1,1)) \leftarrow (\mathbf{R}^\top\mathbf{H}, \mathbf{R} \odot (1-\mathbf{H}))$      ▷ Now $\mathbf{R} \in [0,1]^{\tau+1}$
13:        Normalize $\mathbf{R}$ to sum to one
14:    end for
15:    return $\mathbf{W} \in [0,1]^{t\times f}$ and $\boldsymbol{\mu} \in \mathbb{R}^{t\times f}$     ▷ Can use precomputed predictive
   variances
16: end function
```

5.3.2 Changing Hyper-Parameters

In Section 3.1 we showed that we can integrate out the output scale analytically. Other hyper-parameters such as the length scales or the signal-to-noise ratio cannot be analytically integrated out. However, if we want to integrate out a small number of hyper-parameters, one or two, then we can place a coarse grid on the set of hyper-parameters and effectively apply numerical integration. We must, for each run length, provide a predictive distribution for each hypothesis, i.e. grid point, on the hyper-parameters and do Bayesian model averaging based on the evidence for each hypothesis. Since we have the one-step-ahead predictive log likelihoods, we can get the model evidence. This approach corresponds to putting delta spike priors on the hyper-parameters at each grid point. However, this approach scales exponentially in the number of hyper-parameters integrated out. Unless we are willing to accept a very coarse grid prior, a few hypotheses on the joint configuration of all the GP hyper-parameters, we must move to Monte Carlo methods. Variational approximations may also be possible, but we have not explored this option.

We can place a prior on the changing $p(\lambda|\phi)$ GP hyper-parameters λ with hyper-prior parameters ϕ. In the case of a GPTS with an SE-ARD kernel we use $\lambda := \{\sigma_0^2, \ell, \sigma_n^2\}$. A general principle in approximate and Monte Carlo methods is to integrate out as much analytically as possible. We can set σ_0^2 to be fixed since it is accounted for by the output scale τ which can analytically be integrated out. The hyper-parameter σ_n^2 effectively then becomes a changing signal-to-noise ratio. For simplicity we often place wide Gaussian priors on $\log \ell$ and $\log \sigma_n^2$.

As a result, the UPM becomes:

$$
\begin{aligned}
p(y_t|\mathbf{y}_{(r)}, \phi) &= \int p(y_t|\mathbf{y}_{(r)}, \lambda) p(\lambda|\mathbf{y}_{(r)}) \, d\lambda && (5.43) \\
&= \frac{1}{Z} \int p(y_t|\mathbf{y}_{(r)}, \lambda) p(\mathbf{y}_{(r)}|\lambda) p(\lambda|\phi) \, d\lambda \,, && (5.44)
\end{aligned}
$$

where $Z := \int p(\mathbf{y}_{(r)}|\lambda) p(\lambda|\phi) \, d\lambda \in \mathbb{R}^+$. We can find $p(y_t|\mathbf{y}_{(r)}, \lambda)$ with (5.41), or (5.39) for certain output scale. The log marginal likelihood of the GP $p(\mathbf{y}_{(r)}|\lambda)$, with unknown output scale, is given by:

$$
\begin{aligned}
\log p(\mathbf{y}_{(r)}|\lambda) \overset{(5.40)}{=} &- \left(\alpha_0 + \frac{r}{2}\right) \log \left(1 + \frac{1}{2\beta_0} \mathbf{y}_{(r)}^\top \mathbf{K}^{-1} \mathbf{y}_{(r)}\right) - \frac{1}{2} \log |\mathbf{K}| \\
&+ \log \Gamma \left(\alpha_0 + \frac{r}{2}\right) - \log \Gamma(\alpha_0) - \frac{r}{2} \log(2\pi\beta_0) \qquad (5.45)
\end{aligned}
$$

Since log marginal likelihood has a nonlinear dependence on λ, both the integrals in (5.44) are intractable, regardless of any Gaussianity assumptions on $p(\lambda|\phi)$.

Consequently, we approximate these integrals using two methods, each with their own set of pros and cons. In the first technique we place a grid ($\{\lambda_g\}$) over a subspace of GP hyper-parameters that is assumed to be reasonable for the problem at hand (assigning uniform prior mass for each grid point). The integrals can then be approximated with sums:

$$p(y_t|\mathbf{y}_{(r)}, \lambda) \approx \sum_{\lambda_g} p(y_t|\mathbf{y}_{(r)}, \lambda_g) \left(\frac{p(\mathbf{y}_{(r)}|\lambda_g)}{\sum_{\lambda_g} p(\mathbf{y}_{(r)}|\lambda_g)} \right) . \tag{5.46}$$

Applying more sophisticated quadrature algorithms for (5.44) is difficult as the target function is positive, and the interpolant could venture into the negative region. The grid method does not scale with increasing dimensionality. Alternatively, we can apply Hamiltonian Monte Carlo (HMC) [Duane et al., 1987; Neal, 1992], which scales much better to higher dimensions than grid based methods. In HMC we aim to compute samples $\{\lambda_s\}$ from the posterior $p(\lambda|\mathbf{y}_{(r)})$, allowing us to approximate the integral as a sum:

$$p(y_t|\mathbf{y}_{(r)}, \lambda) \approx \sum_{\lambda_s} p(y_t|\mathbf{y}_{(r)}, \lambda_s) . \tag{5.47}$$

The samples are updated sequentially for each run length hypothesis considered. The samples for $r_t = 0$ are straightforward as they come from the Gaussian prior $p(\lambda|\phi) = \mathcal{N}(\phi_m, \phi_v)$. We can trivially obtain iid samples at this stage. As the posterior $p(\lambda|\mathbf{y}_{(r)})$, represented by samples $\{\lambda_s^{(t-1)}\}$, will look similar to $p(\lambda|\mathbf{y}_{(t-r):t})$, we can initialize the HMC sampler at $\{\lambda_s^{(t-1)}\}$ and run it for a short number of iterations for each sample. In practice, we have found that 4 *trajectories* with a mean of 11 *leapfrog steps* give respectable results. Note that other sequential Monte Carlo (SMC) methods could be used also, though we have not explored these options.

5.3.3 External Inputs

We can model time and the external inputs jointly in a GP, as is the case in a standard GPTS. However, we lose the Toeplitz structure for efficient inference; we also lose the one-dimensional input space allowing for GPK. We must use the rank-1 updates approach. Therefore, we model the dependence on the external inputs with a linear function or with a linear in basis functions relationship. If we can sufficiently constrain the external inputs to an appropriate grid, as mentioned in Section 3.6.1, we can efficiently use the richer model representation of a GP for the external inputs.

5.3.4 Smoothing

Although one of the key features of the BOCPD algorithm is its online nature, we can also do efficient recursions to "smooth back" to find a retrospective distribution on the times of the change points. We can place probability distributions on various quantities by smoothing: the time of the ith change point, the number of change points, the presence of a change point at time t. We can also find the most likely segmentation. Exact samples from the BOCPD posterior were first derived in Fearnhead [2006]. The prior in BOCPD clearly forms a Markov chain by the POE in (5.4). However, Fearnhead [2006] showed it is possible to sample from the posterior on the change point time $\boldsymbol{\tau}$ in a Markovian nature. Therefore, it is possible to recast the posterior on the change point times $P(\boldsymbol{\tau}|\mathbf{y}_{1:T})$ as a Markov chain with T possible states.

We find it convenient to define the following quantities:

$$\mathbf{P}(t,s) := p(\mathbf{y}_{t:s}|\mathbf{c}_{t:s-1}=0)\,, \quad \mathbf{P} \in \mathbb{L}^T\,, \tag{5.48}$$

$$\mathbf{q}(t) := p(\mathbf{y}_{t:T}|c_{t-1}=1)\,, \quad \mathbf{q} \in (\mathbb{R}^+)^T\,. \tag{5.49}$$

We interpret $\mathbf{q}$ as the marginal likelihood of the data under the BOCPD model, *not* the UPM, for any starting point. When smoothing it is more convenient to work with the lifetime distribution $g \in \mathbb{N} \to [0,1]$ representation, as opposed to the hazard in filtering H. We also use the cumulative lifetime distribution $G \in \mathbb{N} \to [0,1]$, where

$$g(t) := \prod_{i=1}^{t-1}(1-H(i))H(t)\,, \quad G(t) := \sum_{i=1}^{t} g(i)\,. \tag{5.50}$$

Using the law of total probability we see that:

$$\mathbf{q}(t) = \sum_{s=t}^{T-1} p(\mathbf{y}_{t:T}, \tau = s) + p(\mathbf{y}_{t:T}, \mathbf{c}_{t+1:T-1} = 0)\,. \tag{5.51}$$

We can further break each element in the sum down as:

$$\begin{aligned} p(\mathbf{y}_{t:T}, \tau = s) &= p(\mathbf{y}_{t:s}, \mathbf{y}_{s+1:T} | \tau = s) P(\tau = s) && (5.52)\\ &= g(s+1-t) p(\mathbf{y}_{t:s} | \mathbf{c}_{t:s-1} = 0) p(\mathbf{y}_{s+1:T} | c_s = 1) && (5.53)\\ &\overset{(5.48)}{=} g(s+1-t)\mathbf{P}(t,s)\mathbf{q}(s+1)\,. && (5.54) \end{aligned}$$

Likewise we see that

$$p(\mathbf{y}_{t:T}, \mathbf{c}_{t+1:T-1} = 0) \overset{(5.48)}{=} \mathbf{P}(t,T)(1 - G(T-t))\,. \tag{5.55}$$

This completes the recursions to solve for $\mathbf{q}$.

Once we have solved for $\mathbf{q}$ we use Bayes' rule and plug in to get the posterior on the time of the initial state:

$$\begin{aligned} P(\tau_1 | \mathbf{y}_{1:T}) &= p(\mathbf{y}_{1:T}, \tau_1)/p(\mathbf{y}_{1:T}) && (5.56)\\ &= P(\tau_1) p(\mathbf{y}_{1:\tau_1} | \tau_1) p(\mathbf{y}_{\tau_1+1:T} | \tau_1)/\mathbf{q}(1) && (5.57)\\ &\overset{(5.48)}{=} g(\tau_1)\mathbf{P}(1,\tau_1)\mathbf{q}(\tau_1+1)/\mathbf{q}(1)\,. && (5.58) \end{aligned}$$

We must also consider the possibility there are no change points at all:

$$P(\boldsymbol{\tau}_{1:T-1} = 0 | \mathbf{y}_{1:T}) \overset{(5.48)}{=} \mathbf{P}(1,T)(1 - G(T-1))/\mathbf{q}(1)\,. \tag{5.59}$$

We can do similar manipulations to solve for the time of later change points

$$\begin{aligned} &P(\tau_j | \mathbf{y}_{1:T}, \tau_{j-1})\\ &= P(\tau_j | \mathbf{y}_{\tau_{j-1}:T}, \tau_{j-1}) && (5.60)\\ &= P(\tau_j | \tau_{j-1}) p(\mathbf{y}_{\tau_{j-1}+1:\tau_j} | \tau_{j-1}, \tau_j) p(\mathbf{y}_{\tau_j+1:T} | \tau_{j-1}, \tau_j)/p(\mathbf{y}_{1:T} | \tau_{j-1}) && (5.61)\\ &= \mathbf{P}(\tau_{j-1}+1, \tau_j)\mathbf{q}(\tau_j+1) g(\tau_j - \tau_{j-1})/\mathbf{q}(\tau_{j-1}+1)\,. && (5.62) \end{aligned}$$

Again we can do similar manipulations to find the probability there are no more change points:

$$\begin{aligned}&P(\mathbf{c}_{\tau_{j-1}+1:T} = 0|\mathbf{y}_{1:T}) \\ &= \mathbf{P}(\tau_{j-1}+1, T)(1 - G(T - \tau_{j-1} - 1))/\mathbf{q}(\tau_{j-1}+1)\,.\end{aligned} \tag{5.63}$$

We can now solve for the conditional posterior on the time of the jth change point $P(\tau_j|\mathbf{y}_{1:T}, \tau_{j-1})$ using these recursions. We can get samples from the posterior over the change point times $P(\boldsymbol{\tau}|\mathbf{y}_{1:T})$ by sequentially sampling from these conditional posteriors until we draw the option of no more change points.

We are more interested in getting the marginal posterior on the time of the jth change point $P(\tau_j|\mathbf{y}_{1:T})$, which we can do without resorting to averaging over Monte Carlo samples. Since the conditional distribution $P(\tau_j|\mathbf{y}_{1:T}, \tau_{j-1})$ in (5.62) only depends on the previous change point time τ_{j-1}, the posterior over change points forms a Markov chain. This can be stated as $\tau_j \perp\!\!\!\perp \boldsymbol{\tau}_{1:j-2}|\mathbf{y}_{1:T}, \tau_{j-1}$. Therefore, we can construct a transition matrix to transform one marginal distribution $P(\tau_{j-1}|\mathbf{y}_{1:T})$ into the next one $P(\tau_j|\mathbf{y}_{1:T})$. Furthermore, since the recursions do not directly depend on j, only the change point's time τ_j, it forms a homogeneous Markov chain.

In order to make sure that the Markov chain only has a finite number of states we restrict ourselves to states where $\boldsymbol{\tau} = 1{:}T-1$. However, we do not know in advance how many nodes will be in that Markov chain, since we do not know for what j when $\tau_j \geq T$, however we can upper-bound it as $T-1$. Although conceptually we would like to think of the change point chain running on forever beyond T we can add a Tth *sink state* to the Markov chain to represent when we have already observed the last regime in the time series. Once the Markov chain is in the sink state the Markov chain stays there permanently. In summary, we have a Markov chain with T possible states of length $T-1$.

We use (5.62) to construct the transition matrix $\mathbf{T}$. Recall that $\mathbf{T}(i,j) = P(\tau_k = j|\mathbf{y}_{1:T}, \tau_{k-1} = i)$ for $i, j < T$, which we can calculate with (5.62). If $j = T$ then we must use (5.63). Finally, $\mathbf{T}(T, i) = \mathbb{I}\{i = T\}$ to implement the sink state.

These operations are summarized in matrix form as:

$$\mathbf{G} := \operatorname{Toeplitz}(\begin{bmatrix}0 & g(1{:}T-2)\end{bmatrix}, \mathbf{0}) \in \mathbb{L}^{T-1}\,, \tag{5.64}$$

$$\mathbf{T} = \begin{bmatrix}\mathbf{P}(2{:}T, 1{:}T-1)^\top \odot \mathbf{G} \odot (\mathbf{q}(2{:}T) \cdot (1 \oslash \mathbf{q}(2{:}T))^\top) & \mathbf{0} \\ \mathbf{P}(2{:}T-1, T) \odot (1 - \mathbf{E}G(1{:}T-2)) \oslash \mathbf{q}(2{:}T-1) & 1\end{bmatrix} \in [0,1]^{T\times T}\,. \tag{5.65}$$

We must calculate the initial state distribution as $P(\tau_1|\mathbf{y}_{1:T})$ using (5.58) and (5.59).

Since the change point structure takes the form of a Markov chain we can find the most likely segmentation using the *Viterbi* algorithm [Viterbi, 1967]. Note that since $\mathbf{q}(t)$ is the marginal probability of BOCPD for all points after t, $\mathbf{q}(1)$ is equivalent to the marginal likelihood computed using the one-step-ahead predictions in (5.13).

Number of change points We can also calculate the posterior on the number of change points M. Note that $\tau_{i+1} = T \Leftrightarrow M \leq i$, where T is being used in the sense of the sink state in the Markov chain formulation above. Therefore, the posterior marginals $P(M \leq i) = P(\tau_{i+1} = T|\mathbf{y}_{1:T})$ represent the CDF on M, which can be differenced for the distribution.

Smoothed run length distribution Although we have derived the recursions for $\mathbf{q}$ to find the smoothed distribution on change point times, some slight modifications are required to find a smoothed version of the filtered run length distribution. We define the run length distribution $r_{t'}$ in terms of $c_{t-1:s-1}$, $t' := s-1$ and $i := s-t$, and then apply Bayes' rule:

$$\begin{aligned}&P(r_{t'} = i|\mathbf{y}_{1:T}) \\ &= P(c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0|\mathbf{y}_{1:T}) &(5.66)\\ &= p(\mathbf{y}_{1:T}|c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0)P(c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0)/p(\mathbf{y}_{1:T})\,. &(5.67)\end{aligned}$$

We now break the data $\mathbf{y}$ into the parts before and after the change point and utilize the conditional independence $\mathbf{y}_{1:t-1} \perp\!\!\!\perp \mathbf{c}_{t:s-1} = 0 | c_{t-1} = 1$:

$$\begin{aligned}
&P(r_{t'} = i | \mathbf{y}_{1:T}) \\
&= p(\mathbf{y}_{1:t-1} | c_{t-1} = 1) p(\mathbf{y}_{t:T} | c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0) P(c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0) / p(\mathbf{y}_{1:T}) \quad (5.68) \\
&= \frac{p(\mathbf{y}_{1:t-1}, c_{t-1} = 1) p(\mathbf{y}_{t:T} | c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0) P(\mathbf{c}_{t:s-1} = 0 | c_{t-1} = 1)}{p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) p(\mathbf{y}_{1:t-1})} \quad (5.69) \\
&= p(c_{t-1} = 1 | \mathbf{y}_{1:t-1}) \mathbf{Q}(t, s) / p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) \,, \quad (5.70) \\
\mathbf{Q}(t, s) &:= p(\mathbf{y}_{t:T} | c_{t-1} = 1, \mathbf{c}_{t:s-1} = 0) P(\mathbf{c}_{t:s-1} = 0 | c_{t-1} = 1) \quad (5.71)
\end{aligned}$$

We now define recursions by generalizing $\mathbf{q}$ to a $\mathbf{Q}$ matrix:

$$\begin{aligned}
\mathbf{Q}(t, s) &= p(\mathbf{y}_{t:T}, \mathbf{c}_{t:s-1} = 0 | c_{t-1} = 1) \quad (5.72) \\
&= \sum_{u=s}^{T-1} p(\mathbf{y}_{t:T}, \tau = u | c_{t-1} = 1) + p(\mathbf{y}_{t:T}, \mathbf{c}_{t:T-1} = 0 | c_{t-1} = 1) \quad (5.73) \\
&\implies \mathbf{Q}(t, s) = p(\mathbf{y}_{t:T}, \tau = u | c_{t-1} = 1) + \mathbf{Q}(t, s+1) \quad (5.74) \\
&= \mathbf{Q}(t, s+1) + \mathbf{P}(t, s) \mathbf{Q}(s+1, s+1) g(s+1-t) \,. \quad (5.75)
\end{aligned}$$

We must use the edge cases of

$$\mathbf{Q}(t, T) = \mathbf{P}(t, T)(1 - G(T - t)) \implies \mathbf{Q}(T, T) = \mathbf{P}(T, T) \,. \quad (5.76)$$

Note that we can find $\mathbf{q}$ from the diagonal of $\mathbf{Q}$: $\mathbf{q}(t) = \mathbf{Q}(t, t)$. The other terms in calculating $P(r_{t'} = i | \mathbf{y}_{1:T})$ in (5.70) are readily available from the filtering step.

$$\begin{aligned}
p(c_{t-1} = 1 | \mathbf{y}_{1:t-1}) &= p(r_{t-1} = 0 | \mathbf{y}_{1:t-1}) \,, \quad (5.77) \\
p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) &= \prod_{j=t}^{T} p(\mathbf{y}_j | \mathbf{y}_{1:j}) \,. \quad (5.78)
\end{aligned}$$

The one-step-ahead likelihoods here were found during the filtering steps.

Conditional independence remark As relied upon in this section the duration of each regime is independent and the parameters within a regime are also assumed to be drawn independently. This is in contrast to an HMM where the transition probability between different states is dependent on the current state, which is more appropriate in settings such as speech recognition. This is arguably the largest weakness in change point models. The tractability of the BOCPD inference algorithm stems from these conditional independence assumptions. Recently, Fearnhead and Liu [2011] has introduced a BOCPD-like model where there is dependence between the regimes and provided an accurate, but approximate, inference algorithm. Combining these regime dependence approaches with the contributions of this chapter, such as Gaussian process UPMs, is an opportunity for future research.

5.4 Supervised Change Point Detection

In many applications change points may correspond to well-defined observable events. In this section we introduce a new method to train a change point model when we have some labeled change points. For instance, we consider a data set of water levels in a dam. We use known faults of water level management in training as change points in a time series of measurements.

Completely unsupervised learning often yields undesirable results in detecting change points. It may find all sorts of changing behavior in the time series to alert as change points despite being irrelevant. Common sense, and theory, suggest including supervision in inference when labels are available. Supervision informs the algorithm of what type of changing behavior we wish to detect. However, it is unrealistic to assume the supervision labels are completely reliable. Change point times may be incorrect or missing and spurious change points may be inserted.

We adapt the unsupervised BOCPD learning method, described in Section 5.1.1, to the supervised case. We can also view the method as a supervised form of time series segmentation.

Throughout this section we assume the following task: A training set containing a time series of measurements $\mathbf{y}_{1:T}$ and noisy change points $\tilde{\mathbf{c}}_{1:T}$ is provided.[1]

[1] The use of $\tilde{\cdot}$ refers to the observed version of a variable, subject to temporal labeling errors

We focus on the case where the quantity of labeled change point data is small since it may be difficult or expensive to obtain. During test we are given *only* the time series $\mathbf{y}_{1:T'}$ and at each time step t we must provide a run length distribution $P(r_t|\mathbf{y}_{1:t})$ indicating the time since the last change point. We must account for change point label noise in test as well since only noisy labels are available for evaluation. This is in contrast to unsupervised change point detection where a common task is to provide the one-step-ahead prediction in data space $p(y_{t+1}|\mathbf{y}_{1:t})$.

Training in supervised BOCPD can be done either discriminatively or generatively; both approaches are used and compared in this section. Both methods are augmented with a noise model that accounts for *jitter* (temporal segmentation error) in the labels. We denote the noise model parameters by θ_n. We summarize all of the hyper-parameters as $\theta := \{\theta_h, \theta_m, \theta_n\}$.

We must consider that the change point labels are often hand labeled and therefore require a noise distribution. A human labeler, despite extensive domain knowledge, may not be able to specify the exact location of the change point in a time series and thereby introduces jitter in the labels. Some change points might require a judgment call on the part of the human labeler and therefore we can consider insertions and deletions in the change point labels. If the amount of noise in the change point labels is small it is computationally advantageous to do noise free training.

The key contributions of this section are: 1) extending BOCPD to the supervised framework using either generative or discriminative training. 2) We also develop a novel noise process for the training labels (i.e. for segmentation error) with an efficient learning method in training and an efficient inference method in test.

The simplest and fastest method of training BOCPD using labeled data is to assume the labels are noise free. However, the noise free assumption is often poor and we weaken it in Section 5.4.1. The hyper-parameters are trained using either the generative likelihood $p(\mathbf{y}_{1:T}, \mathbf{c}_{1:T}|\theta)$ or the discriminative likelihood $P(\mathbf{c}_{1:T}|\mathbf{y}_{1:T}, \theta)$. The generative-discriminative distinction has lately generated much interest [Bishop, 2007].

like $\tilde{c}_t$, as opposed to the latent and noise free version, like c_t.

Generative The generative likelihood p_{Gen} calculated using the decomposition in the POE (5.4). Equivalently, (5.4) can be viewed as the sum of the log marginal likelihood of y within each regime and the sum of the log probabilities of the change point inter-arrival times under the *lifetime distribution* implied by the hazard function. Note that if we marginalize $\mathbf{c}_{1:T}$ out of the generative likelihood (5.4) we get the unsupervised likelihood (5.13).

Discriminative The discriminative likelihood p_{Disc} is also easy to calculate:

$$\log p_{\text{Disc}} := \sum_{t=0}^{T} \log \underbrace{P(r_t|\mathbf{y}_{1:t}, \theta)}_{=:\mathbf{R}(r_t,t)} . \tag{5.79}$$

This means we sum the log probabilities in the path of the $\mathbf{R} \in [0,1]^{T \times T}$ matrix described by the labeled run length. This objective is more similar to the evaluation criteria than the generative objective in (5.4). This training objective is different than if we used $P(c_t|\mathbf{y}_{1:t})$ since that would merely be rewarding parameter settings that could weakly identify change points instantly. It will always take at least a few data points after a change point to identify it.

Evaluating the entire run length matrix, as is done in unsupervised and discriminative training, requires $\mathcal{O}(T^2)$ operations.[1] However, this can be decreased with pruning. By contrast, the generative training procedure only requires $\mathcal{O}(T)$ operations.

5.4.1 Noise Models

Throughout this section we use the zero mean discrete Laplacian (DL) [Inusah and Kozubowski, 2006] to model the jitter $\epsilon_i := \tilde{\tau}_i - \tau_i \in \mathbb{Z}$ as

$$P(\epsilon_i) = \text{DL}(\epsilon_i|\gamma) := \frac{1-\gamma}{1+\gamma}\gamma^{|\epsilon_i|} . \tag{5.80}$$

We also assume the noise is symmetric with mean $\mu = 0$; the noise parameters are only the DL dispersion parameter: $\theta_n := \gamma \in [0,1)$. The jitter ϵ_i is the difference

[1] We assume that a single posterior prediction and posterior update can be done in $\mathcal{O}(1)$ time, which is the case for exponential family UPMs.

between the observed time of ith change point $\tilde{\tau}_i$ and the ith latent (true) change point τ_i. We assume that the observed data $\mathbf{y}_{1:T}$ is conditionally independent of the jitter: $\mathbf{y} \perp\!\!\!\perp \tilde{\boldsymbol{\tau}}|\boldsymbol{\tau}$. Therefore, the "probability of everything" in the noisy case is

$$p(\mathbf{y}_{1:T}, \tilde{\boldsymbol{\tau}}, \boldsymbol{\tau}|\theta) = p(\mathbf{y}_{1:T}|\tau, \theta_m)P(\tilde{\boldsymbol{\tau}}|\tau, \theta_n)P(\boldsymbol{\tau}|\theta_h) \tag{5.81}$$

$$= p_{\text{Gen}} \prod_{i=1}^{M} \text{DL}(\tilde{\tau}_i - \tau_i|\gamma)\,. \tag{5.82}$$

This type of noise model is different from flipping $\tilde{c}$ with some small probability since the DL on $\tilde{\boldsymbol{\tau}}$ enforces the notion that the labeled change points $\tilde{\boldsymbol{\tau}}$ will be close to the latent change points $\boldsymbol{\tau}$. The methodology presented here is more general than the DL and is applicable to any discrete noise distribution.[1] We can find the maximum likelihood estimate (MLE) of γ after observing N data points ϵ by solving the quadratic equation

$$\widehat{\gamma}^2 + \frac{2N}{\sum_{i=1}^{N}|\epsilon_i|}\widehat{\gamma} - 1 = 0\,. \tag{5.83}$$

However, we do not directly observe the jitter ϵ and cannot directly apply (5.83), but we extend it in the next section.

5.4.2 Stochastic Expectation Maximization

We would like to learn the hyper-parameters θ by maximizing the generative likelihood $p(\mathbf{y}_{1:T}, \tilde{\boldsymbol{\tau}}|\theta)$ (the evidence) by integrating out $\boldsymbol{\tau}$ from (5.81). However, we cannot easily compute the one-step-ahead predictive in the presence of noisy change point labels. So, we cannot compute the evidence as easily as in (5.13). We can compute the evidence if we augment the data with the latent change points $\boldsymbol{\tau}$; this is the exact problem the expectation maximization (EM) algorithm [Dempster et al., 1977] is designed to solve. However, in order to use the EM algorithm we must be able to calculate the posterior $P(\boldsymbol{\tau}|\tilde{\boldsymbol{\tau}}, \mathbf{y}_{1:T}, \theta)$ in the E-step. Although we cannot do this in closed form, we approximate the posterior using

[1] We can extend it to deletions and insertions for the jitter-deletion-insertion (JEDI) noise model.

an importance sampler, giving rise to the stochastic expectation maximization (SEM) algorithm [Wei and Tanner, 1990].

Intuitively, we would like to jitter the labels around and average the likelihoods to ensure the learned hyper-parameters are not being fit to label noise. The SEM approach does this in a principled way. In the E-step we find a posterior over the latent change points given the current estimate over the hyper-parameters and the observations. In the M-step we maximize the auxiliary function $Q = \mathbb{E}\left[\log p(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}, \mathbf{y}|\theta)\right] \in \mathbb{R}$ using the posterior from the E-step. The E-step is stochastic because it is approximated using an importance sampler.

In the E-step we use samples from the *proposal distribution* q to approximate the posterior on $\boldsymbol{\tau}$:

$$P(\boldsymbol{\tau}|\theta, \mathbf{y}, \tilde{\tau}) \approx \sum_{i=1}^{N} w_i \delta(\boldsymbol{\tau}_i = \tilde{\boldsymbol{\tau}}) \,, \quad \boldsymbol{\tau}_i \sim q(\boldsymbol{\tau}_i|\tilde{\boldsymbol{\tau}}) \,, \tag{5.84}$$

$$w_i \propto \frac{P(\boldsymbol{\tau}_i|\tilde{\boldsymbol{\tau}}, \mathbf{y})}{q(\boldsymbol{\tau}_i|\tilde{\boldsymbol{\tau}})} \propto \frac{P(\tilde{\boldsymbol{\tau}}|\boldsymbol{\tau}_i) p(\mathbf{y}|\boldsymbol{\tau}_i) P(\boldsymbol{\tau}_i)}{q(\boldsymbol{\tau}_i|\tilde{\boldsymbol{\tau}})} \,, \tag{5.85}$$

where N is the number of importance samples and the weights $\mathbf{w} \in [0, 1]^N$ are normalized so they sum to 1. We use τ_{ij} for the time of the jth change point according to the ith importance sample. Likewise, r_{it} and c_{it} correspond to r_t and c_t according to the ith importance sample. For simplicity, we typically set $q(\boldsymbol{\tau}_i|\tilde{\boldsymbol{\tau}}) = P(\tilde{\boldsymbol{\tau}}|\boldsymbol{\tau}_i) \implies \tau_{ij} \sim \mathrm{DL}(\tilde{\tau}_j|\gamma)$, so that the weights $\mathbf{w}$ are simply weighted according to the noise free marginal likelihood (see (5.4)) of the data $\mathbf{y}_{1:T}$ and the labels $\boldsymbol{\tau}$ had the jittered labels from q been the actual labels: $w_i \propto p(\mathbf{y}|\boldsymbol{\tau}_i) P(\boldsymbol{\tau}_i)$.

In the M-step we get a new estimate of the hyper-parameters θ^{new} using the approximate posterior from the E-step: $\theta^{\text{new}} = \operatorname{argmax}_\theta \mathbb{E}\left[\log p(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}, \mathbf{y}|\theta)\right]$,

$$\mathbb{E}\left[\log p(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}, \mathbf{y}|\theta)\right] \approx \sum_{i=1}^{N} w_i \log p(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}, \mathbf{y}|\theta) \tag{5.86}$$

$$= \sum_{i=1}^{N} w_i \log p(\boldsymbol{\tau}_i, \mathbf{y}|\theta_h, \theta_m) + \sum_{i=1}^{N} w_i \log P(\tilde{\boldsymbol{\tau}}|\boldsymbol{\tau}_i, \theta_n) \,,$$

where the first term is used to find the model and hazard hyper-parameters and the second term is used to find the noise parameters. The optimization routine to find θ_m and θ_h need not be concerned with the second term since it only depends on θ_n.

We estimate γ in the M-step with a MLE extended to handle importance samples. Also note that here each weighted sample is a length M sequence of change points $\boldsymbol{\tau}$. The weighted formula is:

$$\widehat{\gamma}^2 + \frac{2M}{\sum_{i=1}^{N} \sum_{j=1}^{M} w_i |\tau_{ij} - \tilde{\tau}_j|} \widehat{\gamma} - 1 = 0\,. \tag{5.87}$$

To find the model θ_m and hazard θ_h hyper-parameters we simply optimize the weighted expectation of the marginal likelihoods using (5.4) as in the noise free case.

Smoothing alternative Although we use SEM in this section it may be possible to use the smoothing approach of Section 5.3.4 for a more direct approach. Since the smoothed distribution (posterior) on the change point times follows a Markov chain, we could treat the latent change point times as the hidden state in an HMM, and the observed change point times as the observed variables. We could directly maximize the likelihood $p(\mathbf{y}_{1:T}, \tilde{\boldsymbol{\tau}}|\theta)$ in this setup. However, this line of work has not been completely explored.

5.4.2.1 Efficient Implementation

We evaluate the contribution of θ_m and θ_h in the M-step efficiently. Following the same form as (5.4) we see that

$$\begin{aligned}\sum_{i=1}^{N} w_i \log p(\tau_i, \mathbf{y}|\theta_m, \theta_h) &= \sum_{i=1}^{N} w_i \sum_{t=1}^{T} \boldsymbol{\ell}_{r_{i,t-1},t} \\ &+ \log(1 - H(r_{i,t-1}+1))(1 - c_{it}) + \log H(r_{i,t-1}+1) c_{it}\,,\end{aligned} \tag{5.88}$$

where $\boldsymbol{\ell} := \log p(y_t|r_{t-1}, \theta_m) \in \mathbb{R}^{T \times T}$ is a matrix of log posterior predictive probabilities. We next define $\mathbf{W} := \mathbf{W}_0 + \mathbf{W}_1 \in [0,1]^{T \times T}$ using the following weight matrices

$$[\mathbf{W}_0]_{r',t} := \sum_{i=1}^{N} w_i(1 - c_{it})\mathbb{I}\{r_{i,t-1} = r'\}\,, \tag{5.89}$$

$$[\mathbf{W}_1]_{r',t} := \sum_{i=1}^{N} w_i c_{it}\mathbb{I}\{r_{i,t-1} = r'\}\,. \tag{5.90}$$

Given the weight matrices we re-organize the terms to simplify the UPM portion of the likelihood:

$$\begin{aligned}\sum_{i=1}^{N} w_i \sum_{t=1}^{T} \boldsymbol{\ell}_{r_{i,t-1},t} &= \sum_{t=1}^{T}\sum_{r'=0}^{T-1}\sum_{i=1}^{N} w_i \boldsymbol{\ell}_{r',t}\mathbb{I}\{r_{i,t-1} = r'\}\\ &= \sum_{t=1}^{T}\sum_{r'=0}^{T-1} \mathbf{W}_{r',t}\boldsymbol{\ell}_{r',t} = \mathbf{1}^\top(\mathbf{W}\odot\boldsymbol{\ell})\mathbf{1}\,.\end{aligned}$$

Adding the contribution from the hazard we see that

$$\begin{aligned}&\sum_{i=1}^{N} w_i \log p(\tau_i, \mathbf{y}|\theta_m, \theta_h)\\ &= \mathbf{1}^\top(\mathbf{W}\odot\boldsymbol{\ell})\mathbf{1} + \log(1 - H(1{:}T))\mathbf{W}_0\mathbf{1} + \log(H(1{:}T))\mathbf{W}_1\mathbf{1}\,.\end{aligned} \tag{5.91}$$

Additionally, the weight matrices will typically be very sparse. The matrices and $\boldsymbol{\ell}$ only need to be evaluated at the nonzero elements of the weight matrices. The computational complexity of (5.91) is $\mathcal{O}(\|\mathbf{W}\|_0)$, where $\|\mathbf{W}\|_0$ is the "zero norm" or cardinality of $\mathbf{W}$, compared to $\mathcal{O}(NT)$ for naive implementation.

5.4.2.2 Test Set Prediction

So far we have explained how to use a DL noise model efficiently in training. The second step is to get a run length distribution in test. After finding θ in training using SEM, the run length distribution found using ordinary BOCPD will be the predictive distribution on r_t. However, since we will be evaluated on the ability to predict run length induced by the real noisy labels, we must convert the predictive distribution on r_t to one on $\tilde{r}_t$.

We approximate the true posterior in $\tilde{r}_t$ by a distribution conditional on the previous latent change point being the previous labeled change point:

$$P(\tilde{r}_t|r_t) \approx P(\epsilon = r_t - \tilde{r}_t|\epsilon \in [-r_t, t - r_t]) \,. \tag{5.92}$$

In other words, we assume that given the sorted list of change point times $\boldsymbol{\tau}$, the observed change points $\tilde{\boldsymbol{\tau}}$ will be sorted despite applying iid noise to the latent change points $\boldsymbol{\tau}$. The accuracy of approximation (5.92) is investigated in Figure 5.3(c). We next marginalize out r_t to find $\mathbf{R}^*(\tilde{r}_t, t) := P(\tilde{r}_t|\mathbf{y}_{1:t})$:

$$P(\tilde{r}_t|\mathbf{y}_{1:t}) = \sum_{r'=0}^{t} \underbrace{P(r'|\mathbf{y}_{1:t})}_{\mathbf{R}(r',t)} \underbrace{P(\epsilon = r' - \tilde{r}_t)}_{=:\mathbf{K}(r',\tilde{r})} \underbrace{/P(\epsilon \in [-r', t - r'])}_{=:\mathbf{Z}(r',t)} \,. \tag{5.93}$$

Equation (5.93) effectively applies a kernel smoother, which we treat as the noise distribution, to $P(r'|\mathbf{y}_{1:t})$ to find $P(\tilde{r}_t|\mathbf{y}_{1:t})$. In the case of DL noise:

$$\begin{aligned} \mathbf{K}(i,j) &= \mathrm{DL}(i - j|\gamma) \,, \\ \mathbf{Z}(i,j) &= (\mathrm{DLCDF}(j - i|\gamma) - \mathrm{DLCDF}(-i|\gamma))^{-1} \,. \end{aligned} \tag{5.94}$$

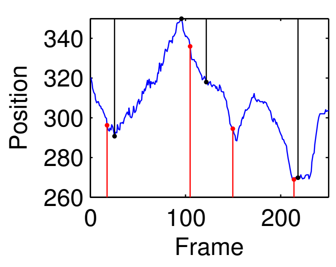
Using (A.21) we rewrite (5.93) as

$$\mathbf{R}^*(\tilde{r}_t, t) = \sum_{r'=0}^{t} \mathbf{R}(r', t)\mathbf{K}(r', \tilde{r}_t)\mathbf{Z}(r', t) \tag{5.95}$$

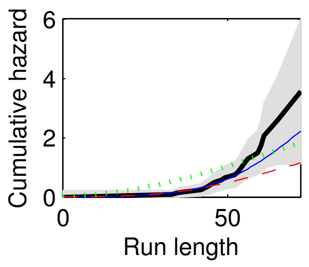
$$\implies \mathbf{R}^* = \mathbf{K}^\top(\mathbf{R} \odot \mathbf{Z}) \in [0, 1]^{T \times T} \,. \tag{5.96}$$

The $\mathbf{K} \in [0, 1]^{T\times T}$ term performs the smoothing while $\mathbf{Z} \in (\mathbb{R}^+)^{T\times T}$ re-weights the elements of $\mathbf{R}$ to control for edge effects resulting from conditioning on $\tilde{r}_t \in [0, t]$. Equation (5.96) can be implemented very efficiently since $\mathbf{K}$ will be (symmetric) Toeplitz, which allows for FFT based methods. Alternatively, we can use pruning and take advantage of sparsity imposed on $\mathbf{R}$. Therefore, the total complexity is $\mathcal{O}(\min(T^2 \log T, T\|\mathbf{R}\|_0))$.

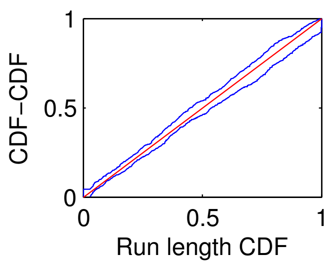
We also use (5.96) in the discriminative learning framework. We substitute the $\mathbf{R}^*$ matrix for the $\mathbf{R}$ matrix in the training objective specified in (5.79). The derivatives $\partial_\theta \mathbf{R}$ can be propagated forward with the chain rule to get $\partial_\theta \mathbf{R}^*$ allowing for gradient based optimization. Like generative SEM training, the noisy



(a) Importance sampler (bee)



(b) Hazard estimation (bee)



(c) Calibration test (synthetic)

Figure 5.3: **Importance sampler**: We show some of the latent change points (red lines below the curve) from the importance sample with the greatest weight w_i after $N = 10^5$ samples. The black lines above the curve are the labeled change points. The blue line is the x position in sequence two of the bee data. **Hazard estimation**: We show Greenwood's nonparametric cumulative hazard estimate from the change points in the entire bee sequence two (19 change points) in black with the 95% confidence region. The solid blue line is the estimate of the cumulative hazard from the SEM method. The red dashed line is estimated from noise free generative training and the green dotted line is from noise free discriminative training. **Calibration test**: A Monte Carlo calibration test [Fearnhead and Liu, 2011] of approximation (5.92) at time $t = 50$ with $N = 10^3$ samples. The CDF ($\leq$) and the CDF off-by-one ($<$) are both in solid blue, which should straddle the diagonal for exact inference.

discriminative setup also allows for joint training of θ_n and $\{\theta_h, \theta_m\}$.

Synthetic data We performed Monte Carlo experiments to validate the accuracy of approximation (5.92) used in finding the run length distribution in test after accounting for jitter. Figure 5.3(c) shows the output of a *calibration test* on the run length distribution at $t = 50$ using synthetic data. We sampled $N = 10^3$ synthetic data sets of $T = 100$ from BOCPD and applied change point noise from the DL with $\gamma = 0.5$. We then looked at the CDF C of the run length in the ith sample at t under the run length distribution from BOCPD, $C_i := P(\tilde{r}_t \leq \tilde{r}_{it}|\mathbf{y}_{1:t})$, using the smoothing method from (5.96). For continuous variables the empirical CDF of the CDF C should follow the diagonal, a uniform distribution, if the inference method is exact. Since the run length distribution is discrete we must look at the CDF ($\leq$) and the CDF off-by-one ($<$), which should straddle the diagonal for exact inference. The calibration test shows that the approximation is accurate in general. This test has to be done with synthetic data since with real data there is no way to determine if the inaccuracies are a result of modeling error or approximation error. We quantify the calibration by doing the appropriate one-sided Kolmogorov-Smirnov (KS) test against a uniform for the upper and lower curves. The curves in Figure 5.3(c) have p-values of $p = 0.9995$ and $p = 0.9977$. This means there is no detectable approximation error under this calibration test. However, for t close to zero the approximation error is detectable due to the edge effects of the jitter sending a change point prior to $t = 0$.

5.5 Results

We tested BOCPD with three different UPMs: IFM, GPTS, ARGP. We compared it against the vanilla GPTS and ARGP as well as the GPIL. For the classical methods, we compared against linear AR, MA, ARMA, and Kalman filtering. We also compare them to the time independent model (TIM), modeling the data as iid normal, as a baseline. The methods were compared on all six real world data sets described in Chapter 2: Nile, well log, Whistler snowfall, bee dance, portfolios, and fish killer. To quantify the performance of each of these models we evaluate them on test data using the one-step-ahead negative log predictive likelihood (NLL) in nats per obs., mean-square-error (MSE), and mean-absolute-error (MAE).[1] The predictive mean was used for the predictions under MSE while the predictive median was used for MAE. We initially evaluate the predictive accuracy of these methods; in Section 5.6 we evaluate run length estimation capability on the data sets where we have ground truth labels of the change points.

All of the change point methods used the logistic hazard: $H(t) = h\sigma(at + b)$, where $\theta_h = \{h, a, b\}$ where learned during training. In GPTS and ARGP we used output scale uncertainty. In the change point methods we learned the scale shape prior α_0 as hyper-parameter, while we fixed $\beta_0 = 1$ as it is unidentifiable when learning the output scale of the covariance functions. For vanilla ARGP and GPTS we fixed $\alpha_0 = 1$ and $\beta_0 = 1$, attempting to learn those hyper-parameters

[1] MSE is given by $\sum_t \|\boldsymbol{\mu}_t - \mathbf{y}_t\|_2^2$ where μ_t is the predictive mean. MAE is given by $\sum_t \|\boldsymbol{\mu}_t - \mathbf{y}_t\|_1$ where μ_t is the predictive median.

by MLE would be inappropriate since we only observe one draw from prior, unlike in the change point context. In the GPTS based methods we found the rational quadratic (RQ) covariance function worked the best. We also included a constant covariance component, which is especially important in the change point context since it allows for a changing mean function as is the case in an IFM. In the ARGP setups we used the squared exponential with automatic relevance detection (SE-ARD) covariance function. In the ARGP we learned orders one through five and selected the order with the best Bayesian information criterion (BIC), see (4.8), on the training set. In GPTS learning we initialized the hyper-parameters with (3.115) using five restarts, although this was often "overkill." For ARGP learning we initialized the hyper-parameters for each order with the hyper-parameters of the previous order with a length scale of $\ell = 1.0$ for the new order. Although the length scale is not unitless an initializer of 1.0 is not arbitrary given the data was standardized. The number of hyper-parameters was used for the parameter count in BIC.

For the Kalman filter (linear dynamical system) we tried latent dimensions of order one through five, selecting the one with the best BIC. For each latent dimensionality we learned the parameters using EM learning initialized at the N4SID solution.

For the AR, MA, and ARMA models we tried order zero through five (36 possible settings for ARMA) selecting by BIC. For each of these models we used the package of Sheppard [2009].[1] All the autoregressive models, AR, ARMA, and ARGP, used an initial state of the time series of zero. We threw out a particular order if the optimal parameters at that order did not give a stationary model. A nonstationary model might have predictions that become unstable, i.e. "blow up" during the test set. One of the advantages of the GP based time series methods is that there is no worry the inferred covariance will ever become nonstationary/unstable.

For the GPIL we used the SE-ARD covariance function with a linear mean component, it was initialized as described in Section 4.3.4. GPIL used a pseudo training set of size $N = 15$.

Each of the data sets was standardized, with the mean and scale determined

[1] We used the package available at `http://www.kevinsheppard.com/wiki/MFE_Toolbox`.

only from the training data. We reported the results on unsalted time series; we also ran experiments on the salted data and found that the performance changed negligibly, i.e. much less than the reported confidence intervals. After dividing each of the data sets into training and test, we trained each of the methods on the training set. The optimal prediction, according to each loss function, was supplied for each one-step-ahead prediction on the training set. We report the mean test set error for each combination of method, data set, and loss metric along with 95% error bars as determined by a t-test. The results are provided with the p-value testing the null hypothesis that methods are equivalent to BOCPD with an ARGP UPM (ARGP-CP) using a two-sided paired t-test, as recommended by Bar-Shalom et al. [2001, Sec. 1.5]. Since the NLL for continuous variables has an arbitrary additive constant determined by the parameterization of the space, we shift the NLL scores so the best performing method has mean NLL of zero.

5.5.1 Nile Data

We first consider the Nile data set. We trained the (hyper) parameters of all the models on data from the first $T = 200$ years (622–821). The predictive performance was evaluated on the following $T' = 463$ years, 822–1284.

The run length posterior of GPTS-CP on the Nile can be seen in Figure 5.4. The installation of the nilometer is the most visually noticeable change in the time series. We also see that the only change point the GPTS-CP is completely confident in occurs shortly after the nilometer installation.

In Figure 5.4 we also see that the smoother is successful at sharpening the run length distribution and median run length. In the filter there are times when it infers a change point and then reverses itself upon realizing the mistake, as shown by the "spiky" nature of the median run length. This behavior is absent in the smoothed distribution. We see that in years 900–1100 even the smoothed distribution maintains ≈ 5 different change point hypotheses. In other areas the smoothed, and to some extent, the filtered distribution is quite confident in the identified change point.

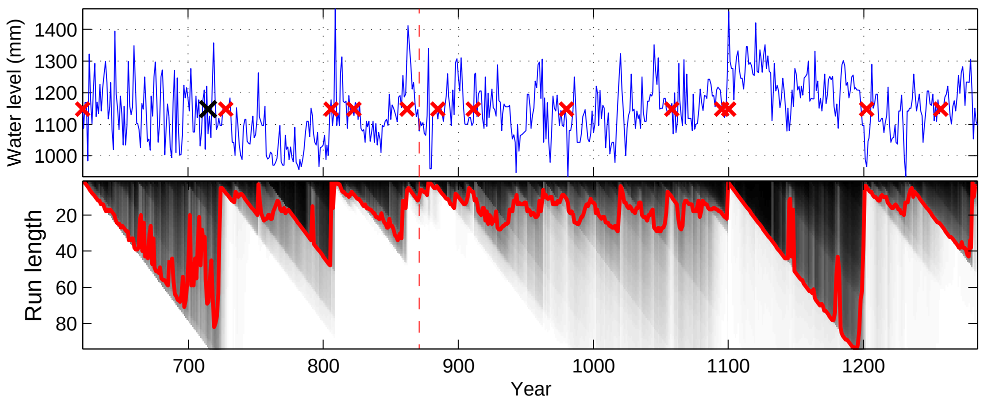
Quantitative results in predictive performance are shown in Table 5.2. We see that the GPTS UPM (GPTS-CP) outperforms all the other methods on all metrics. It even outperforms the ARGP UPM (ARGP-CP) which appears to be more flexible. Also, the Kalman filter does reasonably well on this data set. By analyzing the scatter plots of the Nile data it is apparent that the system is close-to-linear, satisfying the assumption of the Kalman filter, but possibly the noise level in the system is changing over time. It seems the extra ability to model nonlinearity in the ARGP UPM is not as useful as the GPTS UPM ability to aggregate information over a longer time span.

Table 5.2: **Nile**: Results of all methods considered on next step prediction error for Nile data during the test set of 822–1284. We compare: autoregressive (AR) model, autoregressive Gaussian process (ARGP), autoregressive Gaussian process in a change point model (ARGPCP), autoregressive moving average (ARMA), Gaussian process time series in a change point model (GPTSCP), Gaussian process inference and learning (GPIL) state space model, independent factor model (IFM) with change point detection, Kalman filter, moving average (MA), and time independent model (TIM).

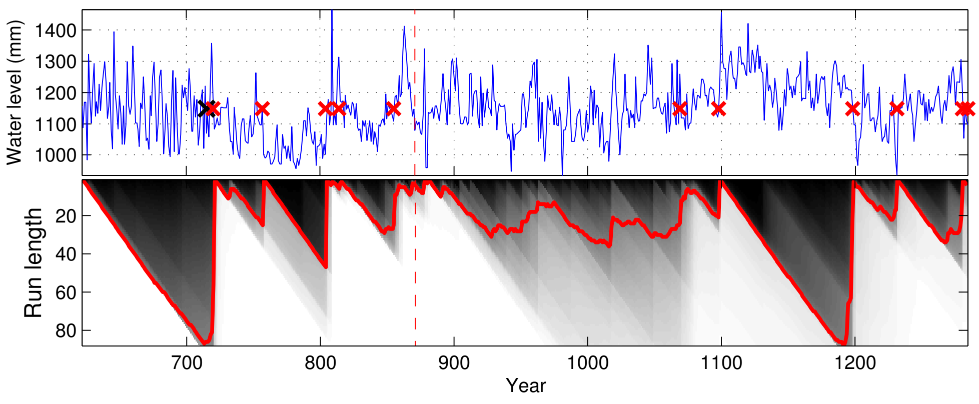
Method	NLL $\times 10^1$	p-value	MSE $\times 10^1$	p-value	MAE $\times 10^1$	p-value
AR	0.81±0.58	0.0002	5.20±0.86	0.1734	5.42±0.44	0.2341
ARGP	0.49±0.64	0.0582	5.06±0.86	0.4536	5.31±0.44	0.8640
ARGPCP	0.21±0.72	N/A	4.94±0.87	N/A	5.29±0.44	N/A
ARMA	1.12±0.67	<0.0001	5.77±0.93	0.0014	5.78±0.46	0.0015
GPTSCP	$\mathbf{0.00 \pm 0.77}$	0.0376	$\mathbf{4.73 \pm 0.82}$	0.0723	$\mathbf{5.12 \pm 0.43}$	0.0267
GPIL	1.28±0.66	<0.0001	5.91±0.93	0.0001	5.87±0.46	<0.0001
GPTS	0.29±0.64	0.5043	4.78±0.81	0.2353	5.18±0.42	0.1870
IFM	0.50±0.86	0.0766	5.20±0.88	0.0609	5.41±0.44	0.1942
Kalman	0.83±0.56	<0.0001	5.19±0.86	0.1382	5.44±0.44	0.1505
MA	2.17±0.61	<0.0001	7.1±1.1	<0.0001	6.53±0.49	<0.0001
TIM	3.94±0.72	<0.0001	10.3±1.5	<0.0001	7.89±0.59	<0.0001

5.5.2 Well Log Data

We also show the results of the well log data after training on the first $T = 1000$ points and testing on the remaining $T' = 3500$ using the same methods as with the Nile data. The results are show in Table 5.3. The well log time series is approximately piece-wise constant, corresponding with the IFM's assumptions. However, the slight nonlinear temporal correlations within each regime give the

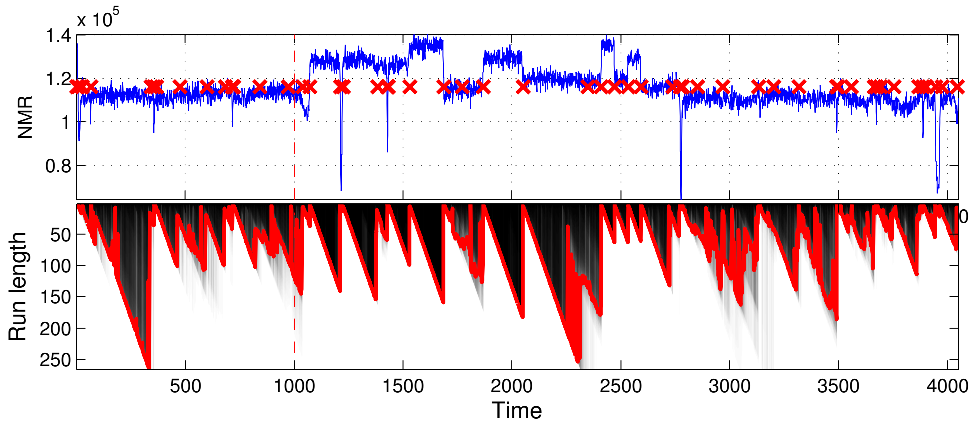


(a) Filtering

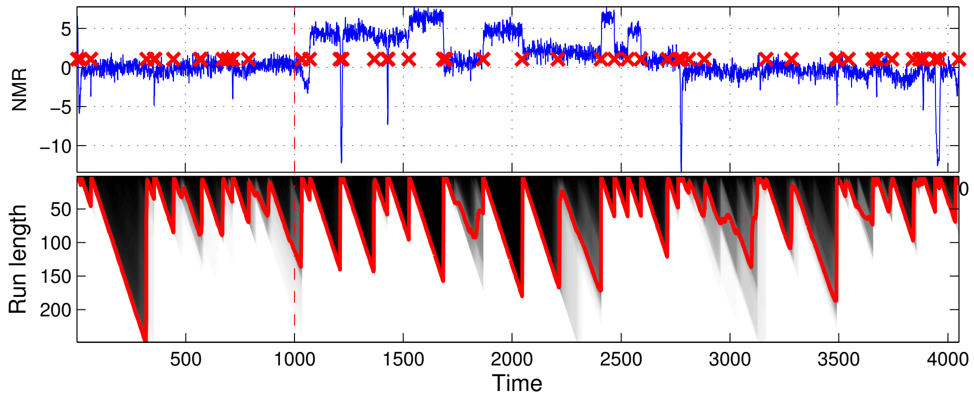


(b) Smoothing

Figure 5.4: **Nile Record**: The output BOCPD with the GPTS UPM, 622–1284. We show the results of filtering (top) and smoothing (bottom). The large black cross marks the installation of the nilometer in 715. The portion before the red dashed line was used to train the hyper-parameters. **Filtering**: The small red crosses mark alert locations where the probability of a change point under the BOCPD posterior since the last alert exceeds 0.95. We call this naive segmentation. On the bottom panel is the run length CDF and its median (solid red). **Smoothing**: The top panel shows the time series segmented by the output of the Viterbi segmentation after smoothing with red crosses. On the lower panel we show the smoothed run length CDF and its median.



(a) Filtering



(b) Smoothing

Figure 5.5: **Well Log**: The output BOCPD with the ARGP UPM. We show the results of filtering (top) and smoothing (bottom). The portion before the red dashed line was used to train the hyper-parameters. **Filtering**: The small red crosses mark alert locations where the probability of a change point under the BOCPD posterior since the last alert exceeds 0.95. We call this naive segmentation. On the bottom panel is the run length CDF and its median (solid red). **Smoothing**: The top panel shows the time series segmented by the output of the Viterbi segmentation after smoothing with red crosses. On the lower panel we show the smoothed run length CDF and its median.

ARGP UPM a slight advantage over the GPTS UPM and the IFM. The ARGP-CP performs much better than the ARGP since the ARGP gets confused by the steplike nature of the time series. We show that the filtered and smoothed segmentations are quite reasonable in Figure 5.5. The obvious discontinuities are flagged as change points. We also see that changes in variance, cusps, and unexpected troughs in the data are flagged as change points.

Table 5.3: **Well log**: Results of all methods (acronyms defined in Table 5.2) considered on next step prediction error for well log data during the test set of the final 3500 points.

Method	NLL $\times 10^1$	p-value	MSE $\times 10^1$	p-value	MAE $\times 10^1$	p-value
AR	14.3±1.3	<0.0001	22.4±1.5	<0.0001	11.24±0.36	<0.0001
ARGP	7.70±0.65	<0.0001	28.2±4.2	<0.0001	11.16±0.45	<0.0001
ARGPCP	**0.00 ± 0.31**	N/A	**6.94 ± 0.81**	N/A	**5.98 ± 0.21**	N/A
ARMA	15.8±2.1	<0.0001	22.0±2.2	<0.0001	10.60±0.37	<0.0001
GPTSCP	0.41±0.30	<0.0001	9.8±1.4	<0.0001	6.42±0.27	<0.0001
GPIL	15.28±0.97	<0.0001	91.0±5.7	<0.0001	21.70±0.75	<0.0001
GPTS	5.43±0.49	<0.0001	15.0±1.2	<0.0001	9.21±0.29	<0.0001
IFM	0.48±0.30	<0.0001	9.8±1.4	<0.0001	6.55±0.27	<0.0001
Kalman	9.8±1.2	<0.0001	17.3±1.4	<0.0001	9.79±0.32	<0.0001
MA	12.9±1.1	<0.0001	22.6±1.5	<0.0001	11.35±0.35	<0.0001
TIM	46.7±3.0	<0.0001	97.4±6.0	<0.0001	22.80±0.76	<0.0001

5.5.3 Snowfall Data

We also used historical daily snowfall data in Whistler, BC, Canada, to evaluate our methodology. The models were trained on three years of data, $T = 1000$ points. We evaluated the models' ability to predict next day snowfall using 35 years of test data, $T' = 12,880$ points. A probabilistic model of the next day snowfall is of great interest to local skiers. In this data set, being able to adapt to different noise levels is key: there may be highly volatile snowfall during a storm and then no snow in between storms. Hence, the data is compatible with the IFM's model assumptions. The GPIL is also competitive. Results are shown in Table 5.4.

GPIL learned a GP model for a scalar close-to-linear stochastic latent transition function. A possible interpretation of the results is that the daily precipitation is nearly linear. Note that for temperatures above freezing no snow occurs, which resulted in a hinge measurement model. GPIL learned a hinge-like function for the measurement model, Figure 5.6, which allowed for predicting no snowfall the next day with high probability. The Kalman filter was incapable of such predictions since it assumes linear functions f and g.

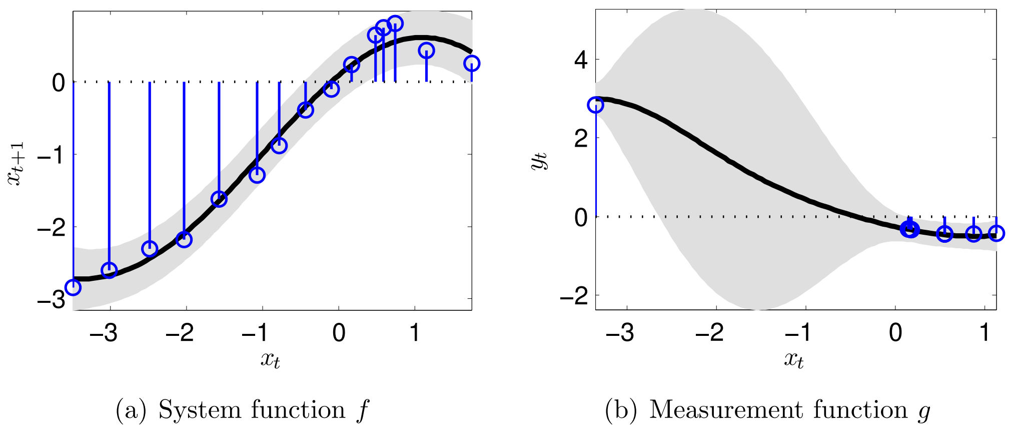


Figure 5.6: Learned measurement GPIL functions f (left) and g (right). The gray area is the 95% error bars that include model uncertainty and measurement noise. Pseudo targets are represented by the blue stem lines.

5.5.4 Fish Killer Data

We also work with the fish killer data. The data was sub-sampled to one data point per hour. We then trained on the first $T = 1000$ points (hours), August 2000–October 2000, and tested on the remaining $T' = 10,294$ ($\approx$ 14 months), October 2000–November 2001.

The GPTS UPM outperforms the ARGP UPM in Table 5.5. The fish killer time series is the smooth time series, making the long-range smoothing of the GPTS advantageous. However, the change point framework gives a very dramatic improvement in performance for the ARGP. The rapid changes in the time series near the fish kills likely confuse the standard ARGP.

Table 5.4: **Snowfall**: Results of all methods (acronyms defined in Table 5.2) considered on next step prediction error for the Whistler snowfall data during the test set of the final 35 years.

Method	NLL $\times 10^1$	p-value	MSE $\times 10^1$	p-value	MAE $\times 10^1$	p-value
AR	17.19±0.30	<0.0001	6.82±0.29	<0.0001	4.76±0.12	0.0129
ARGP	16.28±0.65	<0.0001	10.43±0.42	<0.0001	5.67±0.15	<0.0001
ARGPCP	0.74±0.41	N/A	7.36±0.29	N/A	4.68±0.13	N/A
ARMA	17.19±0.30	<0.0001	6.82±0.29	<0.0001	4.76±0.12	0.0129
GPTSCP	0.54±0.40	<0.0001	7.20±0.29	<0.0001	4.55±0.13	<0.0001
GPIL	16.85±0.41	<0.0001	7.66±0.33	0.0055	4.92±0.13	<0.0001
GPTS	16.71±0.21	<0.0001	6.59±0.27	<0.0001	4.81±0.12	<0.0001
IFM	**0.00 ± 0.43**	<0.0001	7.07±0.29	<0.0001	**4.45 ± 0.13**	<0.0001
Kalman	16.92±0.27	<0.0001	**6.55 ± 0.26**	<0.0001	4.75±0.12	0.0087
MA	17.36±0.29	<0.0001	7.10±0.30	0.0049	5.15±0.12	<0.0001
TIM	19.25±0.21	<0.0001	10.94±0.41	<0.0001	7.53±0.13	<0.0001

Table 5.5: **Fish killer**: Results of all methods (acronyms defined in Table 5.2) considered on next step prediction error for the fish killer data during the test set of the final 10,294 hours.

Method	NLL $\times 10^1$	p-value	MSE $\times 10^1$	p-value	MAE $\times 10^2$	p-value
AR	67.6±1.9	<0.0001	88.1±3.8	<0.0001	239.5±3.4	<0.0001
ARGP	50.0±2.1	<0.0001	85.8±3.8	<0.0001	208.1±4.0	<0.0001
ARGPCP	1.59±0.37	N/A	8.2±2.5	N/A	18.3±1.8	N/A
ARMA	35.3±2.4	<0.0001	2.22±0.19	<0.0001	34.45±0.63	<0.0001
GPTSCP	**0.00 ± 0.35**	<0.0001	0.93±0.12	<0.0001	**8.47 ± 0.49**	<0.0001
GPIL	30.3±2.3	<0.0001	29.8±3.6	<0.0001	85.0±3.0	<0.0001
GPTS	15.4±1.1	<0.0001	**0.69 ± 0.12**	<0.0001	9.88±0.47	<0.0001
IFM	12.30±0.30	<0.0001	4.63±0.70	0.0003	39.3±1.1	<0.0001
Kalman	23.7±3.8	<0.0001	0.78±0.15	<0.0001	11.44±0.50	<0.0001
MA	35.3±2.4	<0.0001	2.22±0.19	<0.0001	34.45±0.63	<0.0001
TIM	67.6±1.9	<0.0001	88.1±3.8	<0.0001	239.5±3.4	<0.0001

5.5.5 Bee Waggle Dance Data

Honey bees perform what is known as a waggle dance on honeycombs. The three stage dance is used to communicate with other honey bees about the location of pollen and water. Ethologists are interested in identifying the change point from one stage to another to further decode the signals bees send to one another. The bee data set contains six videos of sequences of bee waggle dances. The video files have been preprocessed to extract the bee's position and head-angle at each frame. While many in the literature have looked at the cosine and sine of the angle, we chose to analyze angle *differences*. Since this data set is multivariate we included vector autoregression (VAR) and an ARGP extension (VARGP).

In the quantitative results, Table 5.6, we find the ARGP-CP does significantly better than the other methods. The GPTS-CP does not improve much upon the GPTS because the x and y positions over time are explained well by a stationary GPTS model.

The GPTS-CP does identify some change points; since BOCPD is likelihood based no change points would be found if GPTS consistently had the best likelihood. As will be shown in Section 5.6 the performance of change point detection on ground truth change points will be much better if some supervision signal is included during training.

5.5.6 Industry Portfolio Data

We also tried a multivariate data set: the "30 industry portfolios" data set, which was also used in the context of change point detection by Xuan and Murphy [2007]. The data consists of daily returns of 30 different industry specific portfolios from 1963 to 2009. The portfolios consist of NYSE, AMEX, and NASDAQ stocks from industries such as food, oil, telecoms, etc. We trained on the first $T = 1000$ trading days and tested on the remaining $T' = 10,455$ trading days.

The data is heavy tailed and our change point model assumes the data has normal errors. Change point models can justify the assumption of normal errors with a change point model by claiming every heavy tail event is merely a change in variance. This will lead to very frequent change points. So, we transform the data set by whitening and then using the normal quantile function $\mathcal{N}\text{-CDF}^{-1}$ and

Table 5.6: **Bee dance**: Results of all methods (acronyms defined in Table 5.2) considered on next step prediction error for bee dance video sequence one during the test set of the final 807 frames.

Method	NLL $\times 10^0$	p-value	MSE $\times 10^0$	p-value	MAE $\times 10^1$	p-value
AR	10.22±0.34	<0.0001	14.37±0.67	<0.0001	53.2±1.4	<0.0001
ARGP	0.20±0.12	<0.0001	$\mathbf{1.45 \pm 0.21}$	<0.0001	$\mathbf{9.63 \pm 0.63}$	0.0009
ARGPCP	$\mathbf{0.00 \pm 0.13}$	N/A	1.70±0.26	N/A	10.06±0.71	N/A
ARMA	> 1000	0.0002	> 1000	0.0002	> 1000	<0.0001
GPTSCP	0.31±0.13	<0.0001	1.76±0.27	<0.0001	10.44±0.72	<0.0001
GPIL	7.38±0.31	<0.0001	7.07±0.46	<0.0001	34.8±1.3	<0.0001
GPTS	0.38±0.14	<0.0001	1.76±0.27	0.0004	10.21±0.72	0.0009
IFM	1.59±0.15	<0.0001	2.17±0.28	<0.0001	16.00±0.78	<0.0001
Kalman	0.79±0.18	<0.0001	1.78±0.27	<0.0001	10.53±0.72	<0.0001
MA	3.89±0.24	<0.0001	1.68±0.22	0.5684	13.52±0.66	<0.0001
TIM	9.08±0.30	<0.0001	14.68±0.69	<0.0001	53.7±1.4	<0.0001
VAR	0.71±0.21	<0.0001	1.61±0.23	0.0411	10.35±0.66	0.0334
VARGP	0.40±0.15	<0.0001	1.48±0.21	<0.0001	9.85±0.63	0.1506

Student's t CDF with four degrees of freedom, $\tilde{y}_t := \mathcal{N}\text{-CDF}^{-1}(\text{St}_4\text{-CDF}(y_t))$, to implicitly build the heavy tail assumption into the model. The marginal heavy tail behavior of financial returns are known to be well-modeled by a Student's t with four degrees of freedom [Carnero et al., 2004]. We could directly model the data using Student's t distributions, but we would lose the tractability from conjugate priors.

In Figure 5.1, we show that the change points found coincide with significant events with regard to the stock market: the climax of the Internet bubble, the burst of the Internet bubble, and the 2004 presidential election. The methods used in Xuan and Murphy [2007] did not find any correspondence with historical events. For financial returns the iid assumption within a regime is reasonable, so as expected the GP finds very long covariance length scales, attributing the variability within each regime to noise.

We show the predictive performance of each model in Table 5.7. Economic theory suggests that none of the methods should perform better than TIM in terms of their point estimates. Unsurprisingly, all of the methods are clustered

Table 5.7: **Portfolios**: Results of all methods (acronyms defined in Table 5.2) considered on next step prediction error for the industry portfolio data (all 30 indices) during the test set of the final 8,455 trading days, July 3 1975 to December 31 2008.

Method	NLL $\times 10^0$	p-value	MSE $\times 10^0$	p-value	MAE $\times 10^0$	p-value
AR	5.70±0.42	<0.0001	30.21±0.51	<0.0001	23.25±0.19	<0.0001
ARGP	> 1000	<0.0001	30.14±0.50	<0.0001	23.25±0.19	<0.0001
ARGPCP	0.17±0.22	N/A	**29.95 ± 0.50**	N/A	23.35±0.19	N/A
ARMA	5.77±0.42	<0.0001	30.29±0.51	<0.0001	23.28±0.19	<0.0001
GPTSCP	**0.00 ± 0.22**	<0.0001	30.17±0.51	<0.0001	23.38±0.20	0.0075
GPIL	5.04±0.39	<0.0001	29.99±0.50	0.0085	23.25±0.19	<0.0001
GPTS	3.22±0.34	<0.0001	30.20±0.51	<0.0001	**23.24 ± 0.19**	<0.0001
IFM	0.27±0.21	<0.0001	30.43±0.51	<0.0001	23.36±0.20	0.2822
Kalman	5.82±0.43	<0.0001	30.14±0.51	<0.0001	23.25±0.19	<0.0001
MA	6.07±0.43	<0.0001	30.60±0.51	<0.0001	23.40±0.19	0.0002
TIM	5.12±0.40	<0.0001	30.02±0.50	<0.0001	23.26±0.19	<0.0001
VAR	7.60±0.48	<0.0001	31.35±0.53	<0.0001	23.61±0.20	<0.0001
VARGP	9.90±0.45	<0.0001	30.15±0.50	<0.0001	23.28±0.19	<0.0001

close together in terms of MAE and MSE. However, the NLL rewards good estimates of the variance. The GPTS, and in particular change point methods perform the best here. Likely, they are exploiting volatility clustering present in financial time series.

Discussion We see the most pronounced change point behavior in the well log, snowfall, and fish killer data sets. The GPTS UPM appears to work better than the ARGP UPM in data sets where there are long range correlations, even within a single regime, such as the Nile, bee, and fish killer data sets. The GPTS also has an advantage on data sets with close-to-linear correlations such as portfolio data, Nile data, and bee data. The ARGP UPM is more suited towards nonlinear relationships and short term trends as the order is usually short.

5.6 Supervised Results

We compare supervised BOCPD to other methods on its ability to predict the labeled run length and forecast in data space. We compare the generative, discriminative, and unsupervised methods on these tasks. In run length prediction we compare the noise-free models with the noisy models. The models are evaluated on negative log predictive likelihood (NLL), MAE (L_1), and MSE (L_2). The point estimates used in the predictions for MAE and MSE are the posterior median and mean, respectively.

In all of the experiments we compare our results to the data independent model (DIM) as a reference. In data space prediction the DIM model corresponds to modeling the data as iid Gaussian. In the run length space the DIM model corresponds to assuming a constant hazard and an uninformative UPM $p(y_t) \propto 1$, which gives a geometric run length posterior in large t.

We adapted the CUSUM [Page, 1955] method to our experimental setup as well. We used the criterion of Grigg and Spiegelhalter [2008] to set the parameters of the test statistic to optimize the power of the CUSUM, which used the typical change in mean between change points. The CUSUM only provides a test statistic (which can be transformed to a p-value) for the null hypothesis there have been no change points since the last detection. To evaluate on run length the output of CUSUM in training was fit to the true (labeled) run length using a linear model. This supervised CUSUM is included in the comparison.

We used the Gaussian process time series UPM for supervised BOCPD in both the bee data and fish killer data. The independent factor model (IFM) UPM also gave competitive results although not a good as a GP. The GPTS UPM covariance was constructed using a sum of squared-exponential (SE), constant, and noise terms. The constant covariance allowed for a different mean in each regime. We also used a scale parameter $\tau \sim \text{Gamma}(\alpha_0, 1)$ in the covariance. This gives a total of five model hyper-parameters per dimension. As in the unsupervised examples, we used the logistic hazard $H(t) = h\sigma(at + b)$, and the hazard parameters $\theta_h = \{h, a, b\}$ were learned during training. For SEM we used $N = 10^4$ samples in the E-step and did five EM iterations in all the examples shown. Using the methods in Section 5.4.2.1 the computational penalty for a large N is small allowing us to use many samples in the E-step. The results of SEM did not appear to change much after a few iterations.

The methods are compared on bee dance data and the fish killer data, both of which come with labels. The bee data is multivariate and the labels have a strong relationship to the time series. By contrast, the fish killer data is univariate and the labels only correspond to certain features in the time series. A time series may possess many features that appear as change points, but which we have no interest in detecting; for instance, there may be a rapid increase in water level after a storm.

Table 5.8: **Bee dance**: Results for sequence one of the bee data for 250 training points and 807 test points. The results are provided with 95% error bars and the p-value testing the null hypothesis that methods are equivalent to GP SEM using a two-sided paired t-test. The first three columns represent the loss in predicting the run length (using $P(r_t|\mathbf{y}_{1:t})$) while the final column shows the loss in data space of predicting the next data point (using $p(y_{t+1}|\mathbf{y}_{1:t})$). All the methods are generative except those labeled "Disc." We shift the NLL (in nats/observation) so the best method has NLL zero. The MAE and MSE have the units of t and t^2, respectively, in run length prediction. We omit the p-values on the NLL columns as they are all $p < 0.0001$ except a where $p = 0.3429$.

	Run length r					Data $\mathbf{y}$
Method	NLL $\times 10^1$	L1 $\times 10^0$	p-value	L2 $\times 10^{-2}$	p-value	NLL $\times 10^0$
GP Gen. (Eq. (5.4))	36.6±1.9	23.6±2.1	<0.0001	11.5±1.5	<0.0001	0.09±0.15
GP SEM	**0.00 ± 0.87**	**14.4 ± 1.2**	N/A	4.82±0.60	N/A	0.02±0.14
GP Uns. (Eq. (5.13))	34.9±2.2	15.9±1.5	0.0268	6.3±1.1	0.0033	**0.00 ± 0.14**^{a}
IFM Disc. (Eq. (5.79))	41.3±3.8	298±18	<0.0001	> 1000	<0.0001	6.72±0.12
IFM Gen.	125±13	16.38±0.92	0.0039	4.30±0.40	0.1187	2.63±0.11
IFM SEM	2.19±0.48	15.63±0.78	0.0541	**3.63 ± 0.31**	<0.0001	2.65±0.11
IFM Uns.	487±38	21.0±1.2	<0.0001	7.01±0.58	<0.0001	1.33±0.15
TIM	5.02±0.19	18.59±0.86	<0.0001	14.49±0.79	<0.0001	8.69±0.30

5.6.1 Bee Dance Data

As in the unsupervised case, we compare methods on the first sequence of six in the bee data using the first 250 frames (four change points) for training and the remaining 807 frames (15 change points) for test in Table 5.8. In terms of NLL, change point models that lack a label noise distribution produce predictive distributions that are overconfident (too sharp). Noise models help on MSE and

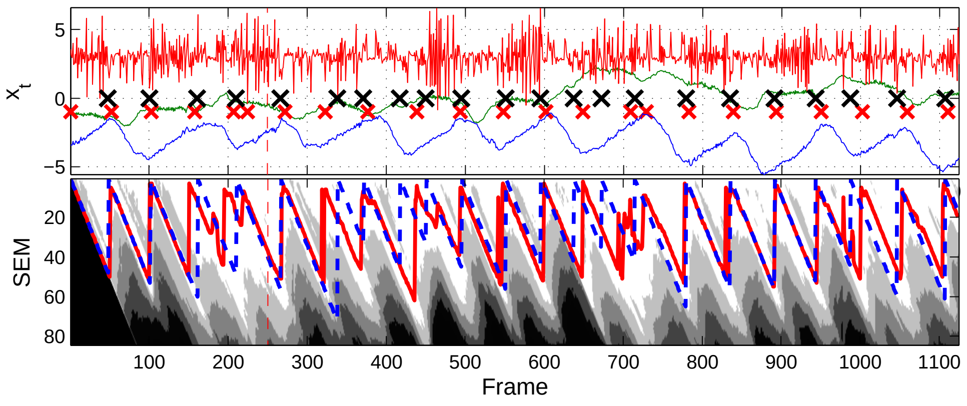


Figure 5.7: **Bee Dance Data**: The top panel shows: x coordinate (blue), y coordinate (green), and angle (red). The large black cross marks the labeled switches in the bee's dance. The red crosses mark alerts from SEM, which are placed when the probability of a change point since the last alert exceeds 95%. The lower panel shows the log run length distribution from SEM (lighter means higher probability). The posterior median in solid red and the labeled run length is marked in dashed blue. Note how the labeled run length falls into the white bands showing that the labeled run length is often near the posterior mode.

MAE as well, but those metrics do not allow for the extreme penalties NLL does. However, there is still a significant increase in performance in terms of MAE and MSE. In Figure 5.7, we see that SEM produces highly accurate run length despite a small training set. The estimated hazard functions from training are shown in Figure 5.3(b) along with a nonparametric hazard estimate from the entire sequence with error bars. Although SEM has the flexibility to de-jitter the change point locations, it still finds a more accurate hazard estimate, in terms of the actual labels, than standard generative training. The de-jittering of the change point labels is shown in Figure 5.3(a). By visual inspection of the bee data we see that the labeled change points are typically ± 10 frames away from the qualitatively optimal positions.

5.6.2 Fish Killer Data

We trained on the August to October period (1000 points) and tested on October to December (another 1000 points), sampled once every 75 minutes. Since water level is positive, we worked with log water level. The beginning and end of every

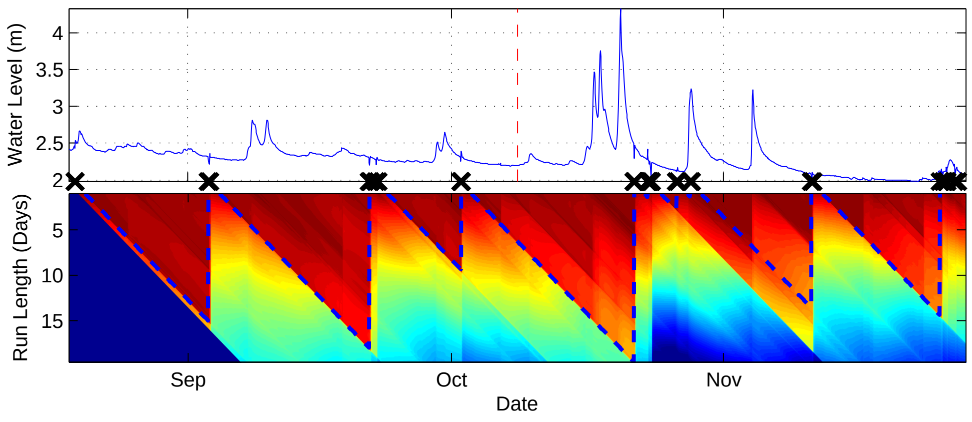


Figure 5.8: **Fish Killer Data**: The top panel shows the water level over a four month period. The lower panel shows the product of the run length distribution and the probability of being in a abnormal state. All other aspects of the plot match the graphical conventions of Figure 5.7.

water oscillation (fish kills) are treated as a change point for the purposes of supervision. Training period has five fish kills while there are another 10 in test. We learned the hyper-parameters θ using the generative training procedure.

We used a mixture of covariances for the UPM:

$$k_{\theta_m}(t,t') = k_{\xi_0}(t,t')(1-s) + k_{\xi_1}(t,t')s\,, \tag{5.97}$$

where $s \in \{0,1\}$ is a latent variable controlling which covariance function to use. Here, $\theta_m = \{\xi_0, \xi_1\}$. After each change point s is drawn from a fair Bernoulli distribution. Both covariance components are the same as the covariance used in the bee data (SE + constant + noise). Equation (5.97) allows for switching length scales at change points, which is helpful since shorter length scales fit the time series better during the fish kills. We plot the results in Figure 5.8. However, instead of plotting the run length matrix $\mathbf{R}$, we plot $\log(\mathbf{R} \odot \mathbf{A})$, $\mathbf{A}_{r',t} := P(s = 1|r_t = r')$, since we are interested in the run length since the last change point that selected the covariance function where the fish kill occurs. The method knows not to flag the water surges from large autumn storms as change points. By contrast, unsupervised training gives an average expected run length of 23 points ($\approx$ 1 day).

5.7 Conclusions

Change point methodology We have shown how to do probabilistic fully Bayesian change point detection in a general manner, i.e. little restriction on UPM or hazard function. We can compute several probabilistic quantities of interest: run length filtering (online operation), run length smoothing (retrospective operation), finding the most likely segmentation, finding the time of the ith change point, and the presence of change point in a segment. As opposed to previously assumed, all of these quantities are calculated exactly and without complex sampling schemes. We also have shown how to do supervised learning if there is a well-defined notion of change point.

Experimental results We have done a rigorous evaluation of all our methods against standard methods ($>$ 10 methods in total) on all data sets in Chapter 2. This included multivariate and univariate problems, the aspects of all the different data sets summarized in Table 2.1. We have compared every combination of over ten methods on six data sets with three loss functions. We can evaluate using a time series prediction task (along with unsupervised change point distribution). However, we can also do supervised evaluation of the run length distribution when labels are available. We have shown that change point methods, especially when combined with Gaussian processes, can lead to excellent performance in general. We have found that the ARGP UPM is arguably the most general and is helped by embedding in a BOCPD scheme. We also found it useful for supervised results to have a noise distribution on change point labels.

Chapter 6

Conclusions

In this thesis we have shown how to unify many different time series methods into a Gaussian process (GP) framework. With the appropriate covariance function (kernel) AR, MA, ARMA, and Kalman filter models (classical models) are equivalent to the GPTS model. We also have explained an arguably more general approach of the ARGP over the GPTS. The ARGP takes the linear relations in the AR model and places a nonparametric prior, the GP, over them. Likewise, the GPIL algorithm, with its filtering sub-routine the GP-ADF, takes the linear relations in a Kalman filter and places a GP prior on them. These nonlinear extensions to these linear models are further augmented by allowing for change points. We allow our models to throw away old data that is no longer useful, in an automatic and model based fashion.

AR models have a long history of researchers engineering extra ad-hoc terms and features to be placed in the model to create some qualitative effect seen in an observed time series. In GPTS and ARGP we can more directly specify our assumptions through the covariance function and then solve for the appropriate inferential/prediction mechanism. For instance, the periodic covariance function can be used to infer the periodic component of a time series. Similar examples exist for many of the challenges in time series data discussed in Section 1.3.1. Likewise, for the change point models we can automatically phase out old data in a model based way rather than contriving ad hoc schemes to throw away the old data.

The covariance building process has a model complexity advantage over the classical time series approach. With a sufficiently large latent state a Kalman filter can have the same features as a GPTS. However, learning the transition matrix of a Kalman filter with a large latent space contains many parameters and can lead to overfitting. The GPTS approach allows for similar qualitative effects while constraining the number of hyper-parameters, which limits the over-fitting potential.

All of these models can be implemented with tractable inference strategies (polynomial time). However, we have explained various strategies that can drastically improve the computational cost over naive implementation: Toeplitz methods, FFT methods, and rank-1 updating schemes. We can also reduce the more sophisticated models such as GPTS to a Kalman filter for doing linear time inference. Therefore, in addition to the modeling benefits of casting classical models in the GP framework, there are computational benefits to doing the reverse.

The model based approaches have benefits in the standard approximation methods used in nonlinear state space models, such as the UKF. They can use Gaussian processes for derivative free optimization of the parameters in the UKF by maximizing the evidence. Although the parameters are usually seen as approximation settings that are set using rough arguments about approximation accuracy, we can improve the performance by learning them as if they were model hyper-parameters.

We have united nonparametric Bayesian methodology with change point detection, both supervised and unsupervised, and state space modeling. The methods in this thesis have extended those used in the classical time series and control methodology and an additional interpretation of standard methods.

Future work This thesis provides many opportunities for future work. For instance, robustness against sigma point collapse in the UKF could be achieved by using the UKF as UPM within a change point framework. When the filter collapses the change point framework can automatically "restart" it. This would be the spirit of the Venn diagram in Figure 1.3 by providing yet another entry in it. There are also opportunities to unite the GP based UPMs with the latent force models of Álvarez et al. [2010], in which we observe the output of an ODE where the external forcing function is a GPTS UPM like process. A valuable direction would be to consider alternative UPMs such as Dirichlet processes. We could also explore combined UPMs where the predictions from GPTS or a Kalman filter are fed into an ARGP as features and trained jointly. We can alternatively combine ARGP and GPTS using a single GP with both time and the previous p points as a single input $(t, \mathbf{y}_{(p)})$. Finally, we allow for an observed process to be a linear transformation of multiple independent change point processes. This would be useful for instance, in the portfolio data, where there might be a change point in a particular sector which affects each dimension of the time to varying degrees.

Appendix A

Mathematical Background

A.1 Matrix Algebra

Traces and diagonals We use the following forms to put trace and diagonal operations in the form of matrix operations:

$$\operatorname{diag}(\mathbf{AB}) = (\mathbf{A} \odot \mathbf{B}^\top)\mathbf{1} = (\mathbf{1}^\top(\mathbf{A}^\top \odot \mathbf{B}))^\top , \tag{A.1}$$

$$\operatorname{tr}(\mathbf{AB}) = \mathbf{1}^\top(\mathbf{A} \odot \mathbf{B}^\top)\mathbf{1} = \mathbf{1}^\top(\mathbf{A}^\top \odot \mathbf{B})\mathbf{1} , \tag{A.2}$$

$$\operatorname{diag}(\mathbf{ab}^\top) = \mathbf{a} \odot \mathbf{b} . \tag{A.3}$$

These alternative forms can be cheaper computationally.

Special matrix operators The exchange matrix $\mathbf{E}$ is the mirror image of the identity matrix $\mathbf{I}$: It flips the elements of a vector $\mathbf{a} \in R^N$ such that

$$\mathbf{Ea} = \mathbf{a}(N{:}-1{:}1) . \tag{A.4}$$

If $N = 3$

$$\mathbf{E}_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} . \tag{A.5}$$

We can use the exchange matrix to define the anti-transpose on a matrix $\mathbf{X}$ as:

$$\mathbf{E}\mathbf{X}^\top\mathbf{E}\,. \tag{A.6}$$

Matrices that are persymmetric, such as Toeplitz matrices, are invariant to the anti-transpose. We can define a differencing operator by:

$$\mathbf{D}_{ij} = \mathbb{I}\{i = j\} - \mathbb{I}\{i = j + 1\} \tag{A.7}$$

$$\implies \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & \dots \\ -1 & 1 & 0 & \dots \\ 0 & -1 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{A.8}$$

Likewise, $\mathbf{D}^{-1}$ acts as the cumulative sum operator, the inverse operation to differencing:

$$\mathbf{D}^{-1}_{ij} = \mathbb{I}\{i \leq j\} \tag{A.9}$$

$$\implies \mathbf{D}^{-1} = \begin{bmatrix} 1 & 0 & 0 & \dots \\ 1 & 1 & 0 & \dots \\ 1 & 1 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{A.10}$$

We can also create a “repeat matrix.” For any $\mathbf{A} \in \mathbb{R}^{M\times N}$,

$$\mathbf{A} \otimes \mathbf{1}^\top = \mathbf{A}\begin{bmatrix}\mathbf{I} & \mathbf{I} & \dots & \mathbf{I}\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{A} & \dots & \mathbf{A}\end{bmatrix}, \tag{A.11}$$

$$\mathbf{1} \otimes \mathbf{A} = \begin{bmatrix}\mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I}\end{bmatrix}\mathbf{A} = \begin{bmatrix}\mathbf{A} \\ \mathbf{A} \\ \vdots \\ \mathbf{A}\end{bmatrix}. \tag{A.12}$$

Cholesky factorization The square root $\mathbf{B}$ of a matrix $\mathbf{A} \in \mathbb{S}^N$ is any matrix such that

$$\mathbf{B}^\top \mathbf{B} = \mathbf{A} \tag{A.13}$$

The matrix square root is not unique. However, the Cholesky factorization of $\mathbf{A}$ is the only square root that is upper triangular. We can find the determinant of $\mathbf{A}$ easily once we have computed the Cholesky factor $\mathbf{B}$:

$$\log |\mathbf{A}| = 2 \cdot \mathbf{1}^\top \log \operatorname{diag}(\mathbf{B})\,. \tag{A.14}$$

Sub-sampling property Certain matrix operations have a sub-sampling property: The solution to smaller operations can found by sub-sampling the result to the larger operation. For instance, if $\mathbf{A} \in \mathbb{S}^N$ and $M \leq N$

$$\operatorname{chol}(\mathbf{A})(1{:}M, 1{:}M) = \operatorname{chol}(\mathbf{A}(1{:}M, 1{:}M))\,. \tag{A.15}$$

A consequence of this is that we can get the intermediate determinants using (A.14). We can get the intermediate results $\mathbf{a}$ of an inner product $\mathbf{x}^\top \mathbf{x}$ with a cumulative sum and Hadamard product:

$$\mathbf{a} = \mathbf{D}^{-1}(\mathbf{x} \odot \mathbf{x})\,. \tag{A.16}$$

The sub-sampling property is preserved when solving a linear system with a lower triangular matrix:

$$\mathbf{L}\mathbf{x} = \mathbf{b} \implies \mathbf{x} = \mathbf{L} \backslash \mathbf{b} \implies \mathbf{x}(1{:}m) = \mathbf{L}(1{:}m, 1{:}m) \backslash \mathbf{b}(1{:}m)\,. \tag{A.17}$$

Hadamard matrix forms The general definition of a matrix multiplication is

$$X = AB \implies X_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}\,, \tag{A.18}$$

We may be in a situation where we have multiple “like terms”

$$X_{ij} = \sum_{k=1}^{p} A_1(i,k)A_2(i,k)\cdots A_n(i,k) \tag{A.19}$$

$$\times B_1(k,j)B_2(k,j)\cdots B_m(k,j) \tag{A.20}$$

$$\times C_1(i,j)C_2(i,j)\cdots C_q(i,j)\,. \tag{A.21}$$

In that case we can group the like terms with a Hadamard product:

$$\begin{aligned} X &= C_1 \odot C_2 \odot \cdots \odot C_q \\ &\quad \odot ((A_1 \odot A_2 \odot \cdots \odot A_n)(B_1 \odot B_2 \odot \cdots \odot B_m))\,. \end{aligned} \tag{A.22}$$

Obviously, if any of these terms reverse the order, i,j instead of j,i, we can substitute the transpose in the final expression.

Geometric matrix series Suppose we have matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ and a series such that,

$$\mathbf{B}_n := \sum_{i=0}^{n-1} \mathbf{A}^i \in \mathbb{R}^{N\times N}\,. \tag{A.23}$$

The series converges if and only if each of the eigenvalues has a magnitude less than one: $|\lambda_i| < 1$. Then the series equals

$$\mathbf{B}_n = (\mathbf{I} - \mathbf{A})^{-1}(\mathbf{I} - \mathbf{A}^n) \in \mathbb{R}^{N\times N}\,, \tag{A.24}$$

and the series converges to

$$\lim_{n\to\infty} \mathbf{B}_n = (\mathbf{I} - \mathbf{A})^{-1}\,. \tag{A.25}$$

Rotation of trace Any trace of a matrix product is invariant under cyclic permutations:

$$\operatorname{tr}(\mathbf{ABCD}) = \operatorname{tr}(\mathbf{BCDA}) = \operatorname{tr}(\mathbf{CDAB}) = \operatorname{tr}(\mathbf{DABC})\,. \tag{A.26}$$

This combined with $x = \mathrm{tr}(x)$ for any scalar x is often used to re-arrange matrix products. For instance, suppose we have vectors $\mathbf{a} \in \mathcal{X} \to \mathbb{R}^{1\times n}$, $\mathbf{b} \in \mathbb{R}^{n\times 1}$, $\mathbf{c} \in \mathbb{R}^{1\times m}$, $\mathbf{d} \in \mathcal{X} \to \mathbb{R}^{m\times 1}$, we can simplify the following expression,

$$\begin{aligned}\int \mathbf{a}(x)\mathbf{b}\mathbf{c}\mathbf{d}(x)p(x)dx &= \int \mathrm{tr}(\mathbf{a}(x)\mathbf{b}\mathbf{c}\mathbf{d}(x))p(x)dx && \text{(A.27)}\\ &= \int \mathrm{tr}(\mathbf{c}\mathbf{d}(x)\mathbf{a}(x)\mathbf{b})p(x)dx && \text{(A.28)}\\ &= \int \mathbf{c}\mathbf{d}(x)\mathbf{a}(x)\mathbf{b}p(x)dx && \text{(A.29)}\\ &= \mathbf{c}\int \mathbf{d}(x)\mathbf{a}(x)p(x)dx\,\mathbf{b} \in \mathbb{R}\,. && \text{(A.30)}\end{aligned}$$

The Lag Matrix When using autoregressive models it will be convenient to work with the lag matrix of a time series vector $\mathbf{y}_{1:T}$. If we use order p we get:

$$\mathrm{ARsplit}_p(\mathbf{y}) = \begin{bmatrix} 0 & 0 & \dots & 0 \\ y_1 & 0 & \dots & 0 \\ y_2 & y_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ y_p & y_{p-1} & \dots & y_1 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \in \mathbb{R}^{T\times p}\,. \qquad \text{(A.31)}$$

A.2 Probability Operations

Gaussian properties In this section we demonstrate four useful properties of a multivariate Gaussian distribution. We can marginalize out variables of a multivariate Gaussian with:

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}^\top \\ \mathbf{C} & \mathbf{D}\end{bmatrix}\right) \implies \mathbf{x} \sim \mathcal{N}(\mathbf{a}, \mathbf{A})\,. \qquad \text{(A.32)}$$

Conditioning is done by:

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}^\top\\ \mathbf{C} & \mathbf{D}\end{bmatrix}\right) \tag{A.33}$$

$$\implies \mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a} - \mathbf{C}^\top\mathbf{D}^{-1}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{C}^\top\mathbf{D}^{-1}\mathbf{C})\,. \tag{A.34}$$

If the mean is also random we can use the *mixing property of Gaussians*:

$$\mathbf{x}|\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\,, \quad \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \implies \mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma} + \boldsymbol{\Sigma}_0)\,. \tag{A.35}$$

When doing a general linear transformation we see that:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{a}, \mathbf{A})\,, \quad \mathbf{y} = \mathbf{C}\mathbf{x} \implies \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{Ca}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{A}^\top\mathbf{C}^\top\\ \mathbf{CA} & \mathbf{CAC}^\top\end{bmatrix}\right)\,. \tag{A.36}$$

Moments of a mixture We can find the moments of any arbitrary mixture distribution provided that we can find the moments of the components. Suppose, we have a mixture distribution in the form

$$p(\mathbf{x}) = \sum_{i=1}^{N} w_i p_i(\mathbf{x})\,, \tag{A.37}$$

where $\mathbf{w} \in [0, 1]^N$ and $\|\mathbf{w}\|_1 = 1$. The expectation and covariance of the ith component is $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$. By the linearity of the expectation,

$$\mathbb{E}\left[\mathbf{x}\right] = \sum_{i=1}^{N} w_i \boldsymbol{\mu}_i\,. \tag{A.38}$$

To find the covariance we start by finding the second moment and subtracting off the outer product of the mean:

$$\mathrm{Cov}\left[\mathbf{x}\right] = \sum_{i=1}^{N} w_i(\boldsymbol{\Sigma}_i + \boldsymbol{\mu}_i\boldsymbol{\mu}_i^\top) - \mathbb{E}\left[\mathbf{x}\right]\mathbb{E}\left[\mathbf{x}\right]^\top\,. \tag{A.39}$$

These are referred to as the *mixture equations* in Bar-Shalom et al. [2001, Ch. 1].

Conditional expectation The law of iterated expectations states that

$$\mathbb{E}_x[x] = \mathbb{E}_y[\mathbb{E}_x[x|y]]\,, \tag{A.40}$$

which follows from Fubini's theorem. Likewise, there is the law of total variance which states that

$$\mathrm{Var}_x[x] = \mathbb{E}_y[\mathrm{Var}_x[x|y]] + \mathrm{Var}_y[\mathbb{E}_x[x|y]]\,. \tag{A.41}$$

Entropy Similarly, we get the following laws relating entropy and mutual information:

$$\mathrm{I}(x;y) = \mathrm{H}[x] - \mathrm{H}[x|y] = \mathrm{H}[y] - \mathrm{H}[y|x] \tag{A.42}$$
$$= \mathrm{H}[x] + \mathrm{H}[y] - \mathrm{H}[x,y]\,. \tag{A.43}$$

The entropy can be viewed as the NLL under the true model:

$$\mathrm{H}[x] := -\int p(x)\log p(x)dx \tag{A.44}$$
$$= \mathbb{E}_p[\text{NLL of } x \text{ under model } p]\,. \tag{A.45}$$

Under an incorrect model q we can express the NLL in terms of KL divergence:

$$\mathbb{E}_p[\text{NLL of } x \text{ under model } q] = \mathrm{KL}(p(x)\|q(x)) + \mathrm{H}[x]\,. \tag{A.46}$$

A.3 The Log-Sum-Exp Trick

We often put variables in algorithms in logarithmic scale for numerical stability; their magnitude varies too wildly to get sufficient precision otherwise. For brevity, we denote the logarithm of a variable $x \in \mathbb{R}^+$ as $x_\ell := \log x \in \mathbb{R}$. When an algorithm involves products moving to logarithmic scale is trivial,

$$x = abc \implies x_\ell = a_\ell + b_\ell + c_\ell\,. \tag{A.47}$$

When the algorithm involves a sum term we must first exponentiate and then sum

$$x = a + b + c \implies x_\ell = \log(\exp(a_\ell) + \exp(b_\ell) + \exp(c_\ell))\,, \tag{A.48}$$

which defeats the whole purpose of working in logarithmic scale in the first place. However, we can cleverly divide out the largest term, by shifting all the terms so that largest is zero:

$$\begin{aligned} m &:= \max(a_\ell, b_\ell, c_\ell)\,, && \text{(A.49)}\\ x_\ell &= \log(\exp(a_\ell) + \exp(b_\ell) + \exp(c_\ell)) + \log(\exp(-m)) + m && \text{(A.50)}\\ &= \log(\exp(a_\ell - m) + \exp(b_\ell - m) + \exp(c_\ell - m)) + m\,. && \text{(A.51)} \end{aligned}$$

This obeys two rules of thumb in numerical stability: Avoid the exponentiation of positive numbers and the logarithm of numbers less than one. Shifting the largest to zero guarantees we will only exponentiate non-positive numbers and since we are summing positive numbers including 1 we know we will take the logarithm of a number larger than one. Even if some of the logarithmic terms underflow by getting shifted to negative infinity, its exponent will be set to zero and the result will be valid.

Appendix B

Foundational Aspects

B.1 Measuring Performance

Given that machine learning is interested in predictive performance we need a way to measure the performance. Machine learning has developed a standard evaluation procedure; for simplicity, we first consider the case of iid data. For iid data, the evaluation procedure works as follows. Each data point is randomly assigned to be in either a training set of size N or a test set of size N'.[1] The models' parameters θ are set in a way specified by the model designer using the training set. Then on each point in the test set the model makes a prediction about the data point y_i. If the model outputs a probability distribution $p(y_i|x_i)$ we convert it to an action $a \in \mathcal{A}$ (or point estimate) by minimizing the expected loss $L \in \mathcal{Y} \times \mathcal{A} \to \mathbb{R}$,

$$a_i = \operatorname*{argmin}_a \mathbb{E}\left[L(y_i, a)\right] = \operatorname*{argmin}_a \int L(y_i, a) p(y_i|x_i)\, dy_i\,. \tag{B.1}$$

This optimization is Bayes' decision rule. In the TIM model, the actions taken a are the ones that minimize (B.1) under TIM's Student's t posterior predictive distribution. Certain methods do not provide a predictive distribution p, and

[1] A common novice mistake is to test a model on the same data set the model was trained on. This gives overly optimistic estimates of performance and does not tell us how well an algorithm will perform on new data. After all, a method could merely memorize the correct output for a training set. Generalization error bounds attempt to quantify how optimistic the training set error is.

directly give an action a.

We have an infinite number of choices for the loss function L (also known as negative utilities U in fields such as economics). If y is continuous ($y \in \mathbb{R}$) common loss functions are mean-square-error (MSE), $L(y_i, a) = (y_i - a)^2$, and mean-absolute-error (MAE), $L(y_i, a) = |y_i - a|$. The optimal actions for MSE and MAE are to supply the mean and median of p, respectively. MSE and MAE are both examples of loss functions for point estimates ($\mathcal{A} = \mathcal{Y}$); however, we can also evaluate the accuracy of a predictive probability distribution ($\mathcal{A} = \mathbf{M}(\mathcal{Y})$).[1] For the log loss, also called negative log likelihood (NLL), $L(y_i, p) = -\log p(y_i|x_i)$, the optimal action is to supply p itself; the log loss is therefore a *proper loss* function [Dawid, 2006]. We should keep track of the units of a loss function: For NLL the unit is nats (or bits if $\log_2$ is used) while for MSE and MAE we use the units of y^2 and y, respectively. For this reason, MSE and MAE are not well-defined for multivariate predictions if the variables have different units. For instance, consider the case of a joint prediction of wind speed and temperature.

In classification $y \in \mathcal{Y}$ (binary $|\mathcal{Y}| = 2$ or multiclass $|\mathcal{Y}| > 2$) it is common to define a *loss matrix* $\mathbf{L} \in \mathbb{R}^{|\mathcal{Y}| \times |\mathcal{A}|}$ where $\mathbf{L}_{ij}$ is the cost incurred by providing a hard label j when the true label is i. The versatility of the log loss allows it to be used in the classification context as well. The models' performance or empirical loss $\widehat{L}$ is the average loss over each test point:

$$\widehat{L} := \frac{1}{N'} \sum_{i=1}^{N'} L(y_i, a_i)\,. \tag{B.2}$$

The generalization error $\mathcal{L}$ is the (hypothetical) limiting case: $\mathcal{L} := \lim_{N' \to \infty} \widehat{L}$.

A wide-spread misperception is that actions must be point estimates: Class probabilities must be converted to a hard cost label or a regression prediction must be converted to a best estimate (albeit with a variable cost function). More complex actions are possible, for instance, a classifier could rank each class according to its likelihood ($\mathcal{A} = \Pi(\mathcal{Y})$) or a regression method could provide a credible interval ($\mathcal{A} = \mathcal{Y}^2$). Other examples include, portfolio allocation in in-

[1] We use the notation $\mathbf{M}(\mathcal{X})$ to describe the set of all probability distributions $p(x)$ on $x \in \mathcal{X}$. More technically, $\mathbf{M}(\mathcal{X})$ is the set of all positive normalized measures on $\mathcal{X}$.

vestment, ordering search results in information retrieval or face recognition, or power to the motor in feedback control. In all of these cases, the actions test many aspects of the predictive distribution. For instance, the portfolio allocation algorithm must balance risk, which utilizes many aspects of the predictive distribution. See Bishop [2007, Ch. 1] for more details.

Another common misperception is that focusing on the expected loss (and not the variance) leads to unnecessary risk taking. Indeed if an agent focuses only on the expected monetary payoff (say expected profit in dollars) unnecessary risks will result: Making a roulette bet with an expected payoff of \$5 will be equally as advisable as receiving \$5 as a sure outcome. This type of situation leads to the St. Petersburg paradox [Bernoulli, 1738]. However, the wide family of loss functions L, such as $-\log(\$)$ or $-\sqrt{\$}$, takes the variance of the payoff into account. In fact, Savage [1954] showed rationality requires an agent pick a loss function L and stay within the expected (not median or maximum) loss framework [Koller and Friedman, 2009, Ch. 22]. A good historical account of these developments is given by Ortega [2010].

Ideally a loss function should not be arbitrary and defined in terms of amount of money made or lost by a particular action a. For instance, in ranking the commonly used normalized discounted cumulative gain (NDCG) [Valizadegan et al., 2009] is a combination of several ad hoc terms that attempt to give "something reasonable." If a human user is taking an action based on the predictive probability or the problem has not been defined in a way that makes the loss function non-arbitrary we would like to evaluate the predictive probabilities on a proper loss function such as the NLL. The NLL is invariant to nonlinear transformations in the data and has other properties general loss functions do not. For instance, in the case of predicting a continuous variable y the mean-square-error (MSE) or mean-absolute-error (MAE) are different for y, y^3, and $\log y$. However, the difference in NLL between two methods is the same in all these cases, which makes the NLL a natural choice for cases when the loss function is arbitrary. Transformation invariance is closely related to *locality* in loss functions: A loss function is local if the loss when the true value is y if the loss only depends on $p(y|x)$ in an infinitesimal neighborhood of y. The NLL was believed to be the only local and invariant proper loss function [Bernardo, 1979]; however, Dawid

[2006] showed that under mild regularity conditions the Fisher loss is local and invariant as well.

Unlike the misclassification rate in hard classification, it is often charged that the log loss is not interpretable. Some rules of thumb for interpreting the log loss have been developed (see Kass and Raftery [1995] and Murray [2007, Ch. 4]). If we attempt to choose between two models $\mathcal{M}_A$ and $\mathcal{M}_B$ based only on N' test set predictions with average loss L_A and L_B in nats per observation (or nats per obs. for short), Bayes' rule says that the posterior of probability of $\mathcal{M}_A$ being correct is:

$$P(\mathcal{M}_A|\text{Test set performance}) = \sigma(-N'(L_A - L_B))\,, \tag{B.3}$$

where $\sigma(\cdot)$ is the logistic sigmoid and the prior on $\mathcal{M}_A$ is $P(\mathcal{M}_A) = 0.5$. Therefore, if we want to go from a prior probability on $\mathcal{M}_A$ of 50% to 95%, we need around $N' = 300$ if the difference in loss is 0.01 nats per obs., $N' = 30$ if the difference in loss is 0.1 nats per obs., and $N' = 3$ if the difference in loss is 1.0 nats per obs. Therefore, a difference less than 0.01 nats per obs. is often said to be of insignificant performance, 0.1 nats per obs. is good, and at 1.0 nats per obs. $\mathcal{M}_A$ is "blowing $\mathcal{M}_B$ out of the water." For continuous variables the log loss is only invariant up to an additive constant and may be negative. Therefore, expressions such as L_A is twice L_B make no sense: We need to think in terms of $L_A - L_B$.

If we measure the NLL in bits there are two other interpretations: 1) If we designed optimal lossless compression codes C_A and C_B based on $\mathcal{M}_A$ and $\mathcal{M}_B$, respectively, then the difference in the compressed file sizes C_A and C_B will be $N'(L_A - L_B)$. 2) Likewise, if a bookie set the odds of a gambling system based on $\mathcal{M}_B$ and the player based his on $\mathcal{M}_A$, with one bet made per test point, then the expected number of rounds for the player to double his money would be $(L_B - L_A)^{-1}$ [Cover and Thomas, 2006, Ch. 6]. This property is invariant to the game played subject to fairness-like regularity conditions.

In many applications algorithms are used in an interactive way with a human user. For instance, in topic modeling documents are clustered into logical categories; Chang et al. [2009] introduced to machine learning the notion of evaluating algorithms by human users — on Mechanical Turk — as opposed to using a contrived loss function.

Benchmarking a time series method is not as straightforward as an iid method. Because the predictions are independent in the iid case, we average the loss over the data points. Suppose we have a training time series of length T and and test set of length T'. Unlike in the iid case, we do not split test and train randomly, we select the later portion of the time series for test and the earlier part for training to maintain causality and a more realistic evaluation. Doing otherwise would allow for *information leakage*; using information from the future to predict the past. We illustrate the difference in data splitting methodology between iid data and time series data in Table B.1. For a time series method we report the average error for a rolling forecast over a forecast horizon of length w:

$$\widehat{L} := \frac{1}{T'} \sum_{t=T+1}^{T+T'} L(y_t, a_t)\,, \tag{B.4}$$

$$a_t = \operatorname*{argmin}_a \mathbb{E}\left[L(y_t, a)\right] \tag{B.5}$$

$$= \operatorname*{argmin}_a \int L(y_t, a) p(y_t | y_{1:t-w})\, dy_t\,. \tag{B.6}$$

We are left with an arbitrary prediction horizon parameter w. If we do not have a reason to favor a particular w, the least arbitrary option is to use $w = 1$: the one-step-ahead predictive. If we use NLL as the loss function and $w = 1$ we find that $\widehat{L} = -\log p(y_{T+1:T'}|y_{1:T})$. The *prequential* framework described in Dawid [1986] is largely based on $w = 1$, most of which was inspired by evaluating probabilities in weather prediction.

iid data	$\diamond$	$\diamond$	$\star$	$\star$	$\star$	$\diamond$	$\star$	$\diamond$	$\star$	$\diamond$	$\star$	$\star$	$\star$	$\diamond$	$\star$
time series data	$\diamond$	$\diamond$	$\diamond$	$\diamond$	$\diamond$	$\diamond$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

Table B.1: Illustration of difference of train ($\diamond$) and test ($\star$) split in time series versus iid data scenarios. We show $N = 6$ training points and $N' = 9$ test points. In the iid case we randomly picked the location of the test points. In the time series case, we have all the training points followed by the test points.

Some exceptions to the standard iid testing procedure shown in Table B.1 exist. There is data set shift, where the test set input distribution $p(x)$ may be different than in training [Quiñonero-Candela et al., 2009]. Transduction, where the algorithm had full knowledge of the test set inputs [Gammerman et al., 1998]. Finally, there is active learning, where the algorithm is allowed to query the training inputs x it believes are most informative [Settles, 2010]. We do not emphasize any three of these cases in the iid case. However, we do extensively work with the time series analog of data set shift, change point detection.

B.2 Bayesian Machine Learning

A significant cohort of machine learning researchers borrows ideas from Bayesian statistics. Bayesians are willing to use probability to express uncertainty over any *operationally defined* quantity; sometimes these probabilities are interpreted as states of belief or knowledge. Although used much earlier by Bayes [1763]; Laplace [1774], Bayesian probability was operationally defined in terms of betting odds by de Finetti [1931b]:

Definition 1. *An agent gives an event A probability $P(A)$ if it would bet \$$P(A)$ when they receive \$1 if A happens and nothing if it does not.*

It has also been indirectly defined by common sense axioms of how someone should rationally calculate states of knowledge by Cox [1946]. Quoting MacKay [2003, Ch. 2] the Cox axioms are summarized by:

Definition 2. *Axiom 1. Degrees of belief can be ordered; if $P(A)$ is "greater" than $P(B)$, and $P(B)$ is "greater" than $P(C)$, then $P(A)$ is "greater" than $P(C)$. [Consequence: Beliefs can be mapped onto real numbers.]*

Axiom 2. The degree of belief in a proposition A and its negation $\bar{A}$ are related. There is a function f such that

$$P(A) = f[P(\bar{A})]\,. \tag{B.7}$$

Axiom 3. The degree of belief in a conjunction of propositions A, B (A and B) is related to the degree of belief in the conditional proposition $A|B$ and the degree of belief in the proposition B. There is a function g such that

$$P(A, B) = g[P(A|B); P(B)]\,. \tag{B.8}$$

This is in contrast to the frequentist alternative where only samples from "repeatable" processes are treated as random and under the jurisdiction of probability theory. In this context, infinite trials form the *definition* of probability.

Definition 3. *The probability of A, $P(A)$, is the proportion of times event A happens over an infinite number of identical trials:*

$$P(A) := \lim_{N\to\infty} \frac{N_A}{N} \,. \tag{B.9}$$

Bayesians charge that this is a metaphysical notion and it is meaningless because (B.9) is not a real mathematical limit; it also conditions on untestable notions of identical trials and independence.

In the Bayesian setting a probability distribution $p(\mathbf{y}_{1:N}) \in \mathbf{M}(\mathcal{Y}^N)$ on a set of variables $\mathbf{y}_{1:N} \in \mathcal{Y}^N$ merely represents a state of knowledge about the variables. Important terminology in Bayesian methods is the prior and posterior: The prior is a probability distribution that expresses a knowledge before seeing any data while the posterior is the updated state of knowledge after observing the data. The prior on parameters are parameterized by hyper-parameters.

At first glance it seems as if we update our beliefs in a completely unconstrained way and there is no calculus to work with such quantities. However, de Finetti [1931b] proved that unless we reason about a set of variables $\mathbf{y}_{1:N}$, e.g. computing $p(\mathbf{y}_a|\mathbf{y}_b)$, $a, b \subseteq 1{:}N$ and use p to make bets, using the laws of probability we could be *Dutch booked*, meaning we will lose money for any possible outcome. We can guarantee we will not be Dutch booked if and only if we make bets using the rules of probability to represent our state of knowledge. No notion of an infinite sequence of experiments is required. Leaving no room for Dutch books is known as being coherent. Bayesian updating leaves a great deal of flexibility in what prior is used, but in order to avoid Dutch books some prior must be used. Note that although this was the first provable justification for Bayesian probability this is by no means the only one (see Bernardo and Smith [2000] for an in-depth summary). Foundational issues for Bayesian probability are found throughout Jaynes [2003].

The work of de Finetti [1931a] and later Hewitt and Savage [1955] provided a bridge between the two notions or probability. If we believe that our observations are *exchangeable*: $p(\mathbf{y}_{1:N}) = p(\mathbf{y}_{\pi(1:N)})$ for any permutation $\pi \in \Pi(1{:}N)$, then the distribution on the data is mathematically equivalent to placing a distribution on a hypothetical limiting distribution on y, $q \in \mathbf{M}(\mathcal{Y})$, and then observing the data y independently from q. We do not believe we are more likely to see any particular data set more than an arbitrary recording of it. In other words, we can represent any exchangeable joint as

$$p(\mathbf{y}_{1:N}) = p(\mathbf{y}_{\pi(1:N)}),\ \forall_{\pi,N} \quad \Leftrightarrow \quad \exists_{\mu(dq)} \text{ s.t. } p(\mathbf{y}_{1:\infty}) = \int \prod_{i=1}^{\infty} q(y_i)\mu(dq)\,, \tag{B.10}$$

where $\mu(dq)$ represents a prior distribution over the sampling distribution q. We can show via a *well-defined* mathematical limit that the empirical distribution on an infinite sequence on y's will approach q:

$$\lim_{N\to\infty} p(\mathbb{P}(\mathbf{y}_{1:N})) = \mu(dq)\,, \tag{B.11}$$

where $\mathbb{P}(\mathbf{y}_{1:N})$ represents the empirical distribution. This notion of an infinite sequence of experiments is legal within the Bayesian framework because it is the mathematical limit of a probability distribution: We are not defining probability in terms of physical limit on an infinite number of events. However, for exchangeable observations the two formulations are mathematically equivalent. Note that the distribution must be infinitely exchangeable; meaning, exchangeability applies for any N with joint distributions consistent under marginalization. Finitely exchangeable versions are derived in Diaconis and Freedman [1980].

An often overlooked aspect of the de Finetti theorem (B.10) is that it requires us to consider distributions on infinite dimensional θ. If we restricted ourselves to finite dimensional θ (parametric models) the de Finetti theorem would be false.

This division in statistics has consequences in the fMRI example from Chapter 1. If we are building a celery detector, the hypothesis testing (frequentist) classifier would be designed such that the probability of classifying the image as a celery stimulus *given it is really from an airplane* is less than α, usually 5% (this is known as a false positive rate). In the Bayesian celery detector, we say the probability that it is a celery, *given the image we observed*, is x%. The substantive difference between the two statements is often puzzling to those not well-versed in probability.[1] The difference is made most clear by extreme case reasoning. No unique procedure is defined for the frequentist celery detector, a celery detector that completely ignored the data and alerted randomly 5% of the time would satisfy the frequentist statement. No unique procedure is defined for the Bayesian change point detector either; in the Bayesian change point detector the designer must first specify a prior probability distribution of the mapping from an image to the probability of the stimulus being celery. Once the prior assumptions are specified, there then exists a uniquely optimal procedure for classifying the fMRI images. Also note that the hypothesis testing statement is asymmetric, we would get different answers depending on if we consider a false positive to be saying it is a celery image when it is an airplane or vice-versa.

The division has a slightly different context in machine learning. Machine learning is concerned about test set performance, so we would like to specify algorithms that make the best predictions on new data. However, it is not possible to say before being given a data set what the best algorithm will be. The "no free lunch theorem" says that algorithms that perform better on one data set will lose some performance on others. Bayesian machine learning cares about achieving optimal performance on average under our prior assumptions. By contrast, frequentist analysis is typically concerned with asymptotic or minimax performance: How well does the algorithm perform in large training sets assuming nature is adversarial. In other words, how well does an algorithm perform in the worst-case; the true generating distribution has been selected to trick the algorithm.

In machine learning practitioners from domain areas often ask "if it works" why do we care about such philosophical arguments. Indeed the predictive performance centric nature of machine learning has led to a more pragmatic stance. All properties of an algorithm should somehow be translatable to how well it performs on a test set; any other considerations lead to unresolvable philosophical wars that preoccupied statistics for decades. Even non-Bayesian theories of machine learning such as probably approximately correct (PAC) [Valiant, 1984]

[1] Benjamin Franklin used the Bayesian formulation as he once said "it is better [one hundred] guilty Persons should escape than that one innocent Person should suffer." [Smyth, 1905, March 14, 1785] Implying that $P(\text{Innocent}|\text{Evidence}) \leq 0.01$ for conviction. The hypothesis testing statement would have been: For every hundred innocent persons tried the number convicted should be less than or equal to one. Or equivalently, $P(\text{Evidence}|\text{Innocent}) \leq 0.01$.

and Vapnik-Chervonenkis (VC) [Vapnik and Chervonenkis, 1971] analysis state their results in terms of how well the algorithm will perform in the test set for a finite (real) training set. The only difference being that the non-Bayesian analysis assumes the "true" data generating mechanism has been chosen in an adversarial manner. A summary of the various optimality criterion are shown in Table B.2.

	Machine Learning	Statistics
Bayesian	Bayes' risk in data space $\mathbb{E}_{\theta,\mathcal{D}}[L(y_\star, a)]$	Bayes' risk in parameter space $\mathbb{E}_{\theta,\mathcal{D}}[L(\theta, a)]$
frequentist	minimax risk in data space $\max_\theta \mathbb{E}_{\mathcal{D}}[L(y_\star, a)]$	minimax risk in parameter space $\max_\theta \mathbb{E}_{\mathcal{D}}[L(\theta, a)]$

Table B.2: Summary of optimality criteria in Machine Learning and Statistics. The divisions between machine learning and statistics are somewhat fuzzy and this table is one way to summarize the difference. However, too boldly promulgating this table as the distinction would invite controversy. Prediction of the population quantity θ can typically be operationally defined as the limiting case predicting value of consistent estimator for large data set: $\lim_{N\to\infty} \widehat{\theta}(\mathcal{D})$. We define $\mathcal{D} := \{X, Y, x_\star, y_\star\}$ as the combination of the training and test set, inputs and outputs. The Bayesian methods will determine a from predictive probabilities using (B.1). In the frequentist criterion it is entirely up to the method on how to predict a.

References

Adams, R. P. and MacKay, D. J. (2007). Bayesian online changepoint detection. Technical report, University of Cambridge, Cambridge, UK. 7, 30, 124

Akaike, H. (1974). A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723. 90

Álvarez, M. A., Peters, J., Schölkopf, B., and Lawrence, N. D. (2010). Switched latent force models for movement segmentation. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 55–63, Cambridge, MA, USA. The MIT Press. 125, 178

Anagnostopoulos, C., Tasoulis, D., Hand, D. J., and Adams, N. M. (2008). Online optimization for variable selection in data streams. In *Proceedings of the 18th European Conference on Artificial Intelligence*, pages 132–136, Patras, Greece. IOS Press. 19

Auer, P. (2000). Using upper confidence bounds for online learning. In *41st Annual Symposium on Foundations of Computer Science*, pages 270–293, Redondo Beach, CA, USA. IEEE Computer Society. 97

Baker, J. K. (1975). The DRAGON system—an overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):24–29. 22

Bar-Shalom, Y., Li, X. R., and Kirubarajan, T. (2001). *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons. 161, 186

Barry, D. and Hartigan, J. A. (1993). A Bayesian analysis of change point problems. *Journal of the Americal Statistical Association*, 88(421):309–319. 124, 133

Baxter, M. and King, R. G. (1999). Measuring business cycles: approximate band-pass filters for economic time series. *Review of Economics and Statistics*, 81(4):575–593. 17

Bayes, T. (1763). Essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418. 194

Beran, J. (1994). *Statistics for Long-Memory Processes*. Monographs on Statistics and Applied Probability 61. Chapman & Hall, New York, NY, USA. 28

Bernardo, J. M. (1979). Expected information as expected utility. *The Annals of Statistics*, 7(3):686–690. 191

Bernardo, J. M. and Smith, A. F. (2000). *Bayesian Theory*. Probability and Statistics. John Wiley & Sons, New York, NY, USA. 195

Bernoulli, D. (1738). Specimen theoriae novae de mensura sortis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 5(2):175–192. In Latin. 191

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer. iii, 17, 151, 191

Boyle, P. and Frean, M. (2005). Multiple output Gaussian process regression. Technical Report CS-TR-05/2, School of Mathematical and Computing Sciences, Victoria University of Wellington, Wellington, New Zealand. 80

Carnero, M. A., Peña, D., and Ruiz, E. (2004). Persistence and kurtosis in GARCH and stochastic volatility models. *Journal of Financial Econometrics*, 2(2):319–342. 169

Chan, T. F. and Lewis, J. G. (1979). Computing standard deviations: accuracy. *Communications of the ACM*, 22(9):526–531. 131

Chang, J., Boyd-Graber, J., Gerrish, S., Wang, C., and Blei, D. (2009). Reading tea leaves: How humans interpret topic models. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 288–296, Cambridge, MA, USA. The MIT Press. 192

Cooke, M., Green, P., Josifovski, L., and Vizinho, A. (2001). Robust automatic speech recognition with missing and unreliable acoustic data. *Speech communication*, 34(3):267–285. 22

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. John Wiley & Sons, 2nd edition. 192

Cox, R. (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14(1):1–13. 194

Crouhy, M., Galai, D., and Mark, R. (2005). *The Essentials of Risk Management*. McGraw-Hill. 17

Csató, L. and Opper, M. (2002). Sparse on-line Gaussian processes. *Neural Computation*, 14(3):641–668. 43, 102

Cunningham, J. P., Shenoy, K. V., and Sahani, M. (2008). Fast Gaussian process methods for point process intensity estimation. In McCallum, A. and Roweis, S., editors, *25th International Conference on Machine Learning*, pages 192–199, Helsinki, Finland. ACM. 62

Darnieder, W. F. (2011). *Bayesian Methods for Data-Dependent Priors*. PhD thesis, The Ohio State University, Columbus, OH, USA. 18

Dawid, A. P. (1981). Some matrix-variate distribution theory: Notational considerations and a Bayesian application. *Biometrika*, 68(1):265–274. 80

Dawid, A. P. (1986). Probability forecasting. *Encyclopedia of Statistical Sciences*, 7:210–218. 193

Dawid, A. P. (2006). The geometry of proper scoring rules. Technical Report 268, Department of Statistical Science, University College London. 190, 191

de Finetti, B. (1931a). Funzione caratteristica di un fenomeno aleatorio. *Atti della R. Academia Nazionale dei Lincei, Serie 6*, 4:251–299. In Italian. 195

de Finetti, B. (1931b). Sul significato soggettivo della probabilità. *Fundamenta Mathematicae*, 17:298–329. In Italian. 9, 10, 194, 195

Deisenroth, M. P. (2009). *Efficient Reinforcement Learning using Gaussian Processes*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany. 45, 48

Deisenroth, M. P., Huber, M. F., and Hanebeck, U. D. (2009). Analytic moment-based Gaussian process filtering. In *26th International Conference on Machine Learning*, pages 225–232, Montreal, QC, Canada. Omnipress. 100, 101, 105, 109, 112, 113, 115, 121, 137

Deisenroth, M. P. and Ohlsson, H. (2011). A general perspective on Gaussian filtering and smoothing: Explaining current and deriving new algorithms. In *American Control Conference*, San Francisco, CA, USA. 86

del Moral, P. (1996). Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2(4):555–580. 85

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38. 153

Diaconis, P. and Freedman, D. (1980). Finite exchangeable sequences. *The Annals of Probability*, 8(4):745–764. 196

Doucet, A., de Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo methods in practice*. Springer Verlag. 119

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222. 144

Eltahir, E. A. B. and Wang, G. (1999). Nilometers, El Niño and climate variability. *Geophysical Research Letters*, 26(4):489–492. 28

Fearnhead, P. (2006). Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and Computing*, 16(2):203–213. 7, 133, 145

Fearnhead, P. and Liu, Z. (2007). Online inference for multiple changepoint problems. *Journal of the Royal Statistical Society, Series B*, 69(4):589–605. 7, 124

Fearnhead, P. and Liu, Z. (2011). Efficient Bayesian analysis of multiple changepoint models with dependence across segments. *Statistics and Computing*, 21(2):217–229. 150, 158

Ferguson, J. (1980). Variable duration models for speech. *Proceedings of the Symposium on the Application of Hidden Markov models to Text and Speech*, 1:143–179. 134

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230. 11, 12

Francis, N., Ghysels, E., and Owyang, M. (2011). The low-frequency impact of daily monetary policy shocks. Working Paper Series 009B, Federal Reserve Bank of St. Louis, St. Louis, MO, USA. 14

Friedman, J. H. (1997). Data mining and statistics. What's the connection? In *Proceedings of the 29th Symposium on the Interface: Computing Science and Statistics*, Houston, TX, USA. 1

Gammerman, A., Vovk, V., and Vapnik, V. (1998). Learning by transduction. In *14th Conference on Uncertainty in Artificial Intelligence*, pages 148–155, Madison, WI, USA. Morgan Kaufmann. 194

Garnett, R., Osborne, M. A., and Roberts, S. J. (2009). Sequential Bayesian prediction in the presence of changepoints. In *26th International Conference on Machine Learning*, pages 345–352, Montreal, QC, Canada. ACM. 28, 125, 136

Geier, G. J. (1996). Odometer assisted GPS navigation method. 4

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. Chapman & Hall, 2nd edition. 132

Ghahramani, Z. and Beal, M. J. (2001). Propagation algorithms for variational Bayesian learning. In *Advances in Neural Information Processing Systems 13*, pages 507–513, Cambridge, MA, USA. The MIT Press. 90

Ghahramani, Z. and Roweis, S. (1999). Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems 11*, pages 599–605, Cambridge, MA, USA. The MIT Press. 53, 85, 100, 101

Gibbs, M. N. (1997). *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, Cambridge, UK. 102

Girard, A., Rasmussen, C. E., Quiñonero-Candela, J., and Murray-Smith, R. (2003). Gaussian process priors with uncertain inputs – application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems 15*, pages 545–552, Cambridge, MA, USA. The MIT Press. 45

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, 3rd edition. 57

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732. 90

Grigg, O. A. and Spiegelhalter, D. J. (2008). An empirical approximation to the null unbounded steady-state distribution of the cumulative sum statistic. *Technometrics*, 4:501–511. 171

Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal Gaussian process regression models. In *Machine Learning for Signal Processing (MLSP)*, pages 379–384, Kittilä, Finland. IEEE. 57, 65

Harvey, A. C. (1991). *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press. 83

Herbrich, R. (2005). Minimising the Kullback-Leibler divergence. Technical Report 74555, Microsoft Research, Cambridge, UK. 46

Hewitt, E. and Savage, L. J. (1955). Symmetric measures on Cartesian products. *Transactions of the American Mathematical Society*, 80:470–501. 195

Hipp, C. (1974). Sufficient statistics and exponential families. *The Annals of Statistics*, 2(6):1283–1292. 11

Honkela, A. and Valpola, H. (2005). Unsupervised variational Bayesian learning of nonlinear models. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 593–600, Cambridge, MA, USA. The MIT Press. 100, 101

Hsu, D., Kakade, S. M., and Zhang, T. (2009). A spectral algorithm for learning hidden Markov models. In *The 22nd Conference on Learning Theory*, Montreal, QC, Canada. 63

Hutchison, K. (1993). Is classical mechanics really time-reversible and deterministic? *The British journal for the philosophy of science*, 44(2):307–323. 76

Inusah, S. and Kozubowski, T. J. (2006). A discrete analogue of the Laplace distribution. *Journal of Statistical Planning and Inference*, 136(3):1090–1102. 152

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. 195

Johnson, M. T. (2005). Capacity and complexity of HMM duration modeling techniques. *IEEE Signal Processing Letters*, 12(5):407–410. 134

Jones, S. (2009). The formula that felled Wall Street. In *Financial Times*. 123

Julier, S. J. and Uhlmann, J. K. (1997). A new extension of the Kalman filter to nonlinear systems. In *Proceedings of AeroSense: 11th Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pages 182–193, Orlando, FL, USA. 85

Julier, S. J. and Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422. 85

Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F. (1995). A new approach for filtering nonlinear systems. In *American Control Conference*, volume 3, pages 1628–1632, Seattle, WA, USA. IEEE. 92

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME — Journal of Basic Engineering*, 82(Series D):35–45. 4, 85

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the Americal Statistical Association*, 90(430):773–795. 192

Kassam, S. and Poor, H. (1985). Robust techniques for signal processing: A survey. *Proceedings of the IEEE*, 73(3):433–481. 22

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25. 112

Ko, J. and Fox, D. (2009). GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90. 100, 101, 121

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models*. The MIT Press, Cambridge, MA, USA. 191

Kuss, M. (2006). *Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning*. PhD thesis, Technische Universität Darmstadt, Germany, Darmstadt, Germany. 45

Laplace, P. S. (1774). Mémoires sur la probabilité des causes par les évènements. *Mémoires de mathématique et des physiques presenteés à l'Académie royale des sciences, par divers savans, & lûs dans ses assemblées*, 6:621–656. In French. 194

Lawrence, N. D. (2004). Gaussian process models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems 16*, pages 329–336, Cambridge, MA, USA. The MIT Press. 102

Lázaro-Gredilla, M. (2010). *Sparse Gaussian Processes for Large-Scale Machine Learning*. PhD thesis, Universidad Carlos III de Madrid, Madrid, Spain. 55, 104

Lefebvre, T., Bruyninckx, H., and De Schutter, J. (2004). *Nonlinear Kalman filtering for force-controlled robot tasks*. Springer-Verlag Berlin. 83

Li, D. X. (2000). On default correlation: A copula function approach. Working paper 99-07, The RiskMetrics Group, 44 Wall St. New York, NY 10005. 123

Little, R. J. A. and Rubin, D. B. (1987). *Statistical analysis with missing data*. John Wiley & Sons, New York, NY, USA. 19

Lux, T. and Marchesi, M. (2000). Volatility clustering in financial markets: A microsimulation of interacting agents. *International Journal of Theoretical and Applied Finance*, 3(4):675–702. 4

MacKay, D. J. (1998). Introduction to Gaussian processes. *Neural Networks and Machine Learning*, 168:133–166. 38

MacKay, D. J. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. 90, 194

Mann, T. P. (2006). Numerically stable hidden Markov model implementation. Technical report, University of Washington, Seattle, WA, USA. 133

Marlin, B. M., Zemel, R. S., Roweis, S., and Slaney, M. (2007). Collaborative filtering and the missing at random assumption. In *23rd Conference on Uncertainty in Artificial Intelligence*, pages 50–54, Vancouver, BC, Canada. 19

Martins, J. R., Sturdza, P., and Alonso, J. J. (2003). The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):245–262. 111

Marvasti, F. A., editor (2001). *Nonuniform Sampling: Theory and Practice*, volume 1. Springer. 22

Matheron, G. (1973). The intrinsic random functions and their applications. *Advances in Applied Probability*, 5(3):439–468. 34

Maybeck, P. S. (1979). *Stochastic Models, Estimation, and Control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, Inc. 85

Minka, T. (2001). Bayesian linear regression. Technical report, MIT Media Lab. 135, 136

Mitchell, T. M. (2006). The discipline of machine learning. Technical Report CMU-ML-06-108, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. 2

Mitchell, T. M., Shinkareva, S. V., Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. A., and Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195. 2

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2008). Junior: The Stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597. 3

Murray, I. (2007). *Advances in Markov chain Monte Carlo methods*. PhD thesis, Gatsby computational neuroscience unit, University College London, London, UK. 90, 192

Murray-Smith, R. and Girard, A. (2001). Gaussian process priors with ARMA noise models. In *Irish Signals and Systems Conference*, pages 147–152, Maynooth, Ireland. 56, 67

Neal, R. M. (1992). Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Working paper CRG-TR-92-1, University of Toronto, Connectionist Research Group, ON, Canada. 144

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. Springer, New York, NY, USA. 54, 109

Neal, R. M. (1997). Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report 9702, Dept. of statistics and Dept. of Computer Science, University of Toronto, Toronto, ON, Canada. 45

Neal, R. M. (1998). Assessing relevance determination methods using DELVE. *NATO ASI Series F Computer and Systems Sciences*, 168:97–132. 37

Neal, R. M. and Hinton, G. E. (1998). *Learning in Graphical Models*, chapter A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants, pages 355–368. Kluwer Academic Publishers. 89

Nelson, C. R. and Kang, H. (1981). Spurious periodicity in inappropriately detrended time series. *Econometrica: Journal of the Econometric Society*, 49(3):741–751. 17

Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. In *21st International Conference on Machine Learning*, volume 69, page 78, Banff, AB, Canada. ACM. 37

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2010). Real-time local GP model learning. *From Motor Learning to Interaction Learning in Robots*, 264:193–207. 77

Oh, S. M., Rehg, J. M., Balch, T., and Dellaert, F. (2008). Learning and inferring motion patterns using parametric segmental switching linear dynamic systems.

International Journal of Computer Vision (IJCV) Special Issue on Learning for Vision, 77(1–3):103–124. 28

Olofsson, T. (2005). Deconvolution and model-based restoration of clipped ultrasonic signals. *Instrumentation and Measurement, IEEE Transactions on*, 54(3):1235–1240. 21

Oppenheim, A. V. and Schafer, R. W. (1989). *Discrete-time signal processing*. Prentice Hall, 3rd edition. 22

Orbanz, P. (2009). Construction of nonparametric Bayesian models from parametric Bayes equations. In *Advances in Neural Information Processing Systems 22*, pages 1392–1400, Cambridge, MA, USA. The MIT Press. 34

Ortega, P. A. (2010). *A Unified Framework for Resource-Bounded Agents Interacting with an Unknown Environment*. PhD thesis, University of Cambridge, Cambridge, UK. 191

Ó Ruanaidh, J. J., Fitzgerald, W. J., and Pope, K. J. (1994). Recursive Bayesian location of a discontinuity in time series. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 4, pages 513–516, Adelaide, SA, Australia. IEEE. 30

Osborne, M., Garnett, R., Swersky, K., and de Freitas, N. (2011). A machine learning approach to pattern detection and prediction for environmental monitoring and water sustainability. In *ICML Workshop on Machine Learning for Global Challenges*, Bellevue, WA, USA. 29

Osborne, M. A., Garnett, R., and Roberts, S. J. (2009). Gaussian processes for global optimization. In *3rd International Conference on Learning and Intelligent Optimization (LION3)*, Lecture Notes in Computer Science (LNCS), Trento, Italy. Springer. 96

Osborne, M. A., Garnett, R., and Roberts, S. J. (2010). Active data selection for sensor networks with faults and changepoints. In *Advanced Information Networking and Applications (AINA)*, pages 533–540, Perth, WA, Australia. 17

Page, E. S. (1955). A test for a change in a parameter occurring at an unknown point. *Biometrika*, 42(3–4):523–527. 171

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288. 85

Pollard, D. (2002). *A User's Guide to Measure Theoretic Probability*. Cambridge University Press. 10

Quiñonero-Candela, J., Girard, A., Larsen, J., and Rasmussen, C. E. (2003). Propagation of uncertainty in Bayesian kernel models—application to multiple-step ahead forecasting. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 2, pages 701–704, Hong Kong. IEEE. 45, 76, 105

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959. 102

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset shift in machine learning*. Neural Information Processing. The MIT Press. 193

Rasmussen, C. E. and Ghahramani, Z. (2001). Occam's razor. In *Advances in Neural Information Processing Systems 13*, pages 294–300, Cambridge, MA, USA. The MIT Press. 90

Rasmussen, C. E. and Ghahramani, Z. (2003). Bayesian Monte Carlo. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 489–496, Cambridge, MA, USA. The MIT Press. 97, 137

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press. iii, 3, 33, 38, 40, 135

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamical systems. *AIAA Journal*, 3(8):1445–1450. 89

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407. 19

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345. 63

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536. 54

Saatçi, Y., Turner, R., and Rasmussen, C. E. (2010). Gaussian process change point models. In *27th International Conference on Machine Learning*, pages 927–934, Haifa, Israel. Omnipress. iii

Salakhutdinov, R., Roweis, S., and Ghahramani, Z. (2003). Optimization with EM and expectation-conjugate-gradient. In *20th International Conference on Machine Learning*, pages 672–679, Washington, DC, USA. The AAAI Press. 89

Savage, L. J. (1954). *The Foundations of Statistics*. John Wiley & Sons, New York, NY, USA. 191

Schölkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, Cambridge, MA, USA. 54, 77

Schwarz, G. E. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464. 90

Seeger, M. (2008). Low rank updates for the Cholesky decomposition. Technical report, Department of EECS, University of California at Berkeley, Berkeley, CA, USA. 73, 74

Serra, O. (1984). *Fundamentals of Well-log Interpretation: The acquisition of logging data*. Developments in Petroleum Science. Elsevier Science Ltd, New York, NY, USA. 32

Settles, B. (2010). Active learning literature survey. Computer Sciences 1648, University of Wisconsin-Madison, Madison, WI, USA. 194

Sheppard, K. (2009). *MFE MATLAB Function Reference Financial Econometrics*. 160

Shumway, R. and Stoffer, D. (1982). An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264. 89

Smyth, A. H., editor (1905). *The Writings of Benjamin Franklin*, volume 9. Macmillan, New York, NY, USA. 197

Snelson, E. and Ghahramani, Z. (2005). Compact approximations to Bayesian predictive distributions. In *22nd International Conference on Machine Learning*, pages 840–847, Bonn, Germany. Omnipress. 96

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264, Cambridge, MA, USA. The MIT Press. 102

Snelson, E., Rasmussen, C. E., and Ghahramani, Z. (2004). Warped Gaussian processes. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, pages 337–344, Cambridge, MA, USA. The MIT Press. 72

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In Fürnkranz, J. and Joachims, T., editors, *27th International Conference on Machine Learning*, pages 1015–1022, Haifa, Israel. Omnipress. 96, 97

Storkey, A. J. (1999). Truncated covariance matrices and Toeplitz methods in Gaussian processes. In *9th International Conference on Artificial Neural Networks*, volume 1, pages 55–60, Edinburgh, UK. IET. 79

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA. 97

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press, Cambridge, MA, USA. 83, 93

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *12th International Conference on Artificial Intelligence and Statistics*, volume 5 of *W&CP*, pages 567–574, Clearwater Beach, FL, USA. Journal of Machine Learning Research. 104

Titsias, M. K., Rattray, M., and Lawrence, N. D. (2010). *Inference and Learning in Dynamic Models*, chapter Markov chain Monte Carlo algorithms for Gaussian processes, pages 1–25. Cambridge University Press. 42

Tolman, R. C. (1938). *The principles of statistical mechanics*. Dover Publications. 76

Trench, W. F. (1964). An algorithm for the inversion of finite Toeplitz matrices. *Journal of the Society for Industrial and Applied Mathematics*, 12(3):515–522. 58

Turner, R. (2008). Bayesian approaches to rare event prediction in multivariate time series. First year report, University of Cambridge, Cambridge, UK. iii

Turner, R. (2010). Bayesian change point detection for satellite fault prediction. In French, M., Jackson, S., and Jokisuu, E., editors, *Interdisciplinary Graduate Conference (IGC) 2010*, pages 213–221, Cambridge, UK. iii

Turner, R., Deisenroth, M. P., and Rasmussen, C. E. (2009a). System identification in Gaussian process dynamical systems. In Görür, D., editor, *Nonparametric Bayes Workshop at NIPS 2009*, Whistler, BC, Canada. iii

Turner, R., Deisenroth, M. P., and Rasmussen, C. E. (2010). State-space inference and learning with Gaussian processes. In *13th International Conference on Artificial Intelligence and Statistics*, volume 9 of *W&CP*, pages 868–875, Chia Laguna, Sardinia, Italy. Journal of Machine Learning Research. iii, 137

Turner, R. and Rasmussen, C. E. (2010). Model based learning of sigma points in unscented Kalman filtering. In Kaski, S., Miller, D. J., Oja, E., and Honkela, A., editors, *Machine Learning for Signal Processing (MLSP)*, pages 178–183, Kittilä, Finland. IEEE. iii

Turner, R., Saatçi, Y., and Rasmussen, C. E. (2009b). Adaptive sequential Bayesian change point detection. In Harchaoui, Z., editor, *Temporal Segmentation Workshop at NIPS 2009*, Whistler, BC, Canada. iii

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142. 197

Valizadegan, H., Jin, R., Zhang, R., and Mao, J. (2009). Learning to rank by optimizing NDCG measure. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 1883–1891, Cambridge, MA, USA. The MIT Press. 191

van der Merwe, R. and Wan, E. A. (2001). The square-root unscented Kalman filter for state and parameter-estimation. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 6, pages 3461–3464, Salt Lake City, UT, USA. IEEE. 88

van Overschee, P. and de Moor, B. (1994). N4SID: Subspace algorithms for the identification of comined deterministic-stochastic systems. *Automatica*, 30(1):75–93. 63, 110

Vanhatalo, J., Jylänki, P., and Vehtari, A. (2009). Gaussian process regression with Student-t likelihood. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 1910–1918, Cambridge, MA, USA. The MIT Press. 45

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280. 10, 198

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269. 148

Wan, E. A. and van der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control*, pages 153–158, Lake Louise, AB, Canada. IEEE. 93

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2008). Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298. 100, 101, 137

Wasserman, L. (2006). *All of nonparametric statistics.* Springer-Verlag New York Inc. 11

Wei, G. C. G. and Tanner, M. A. (1990). A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the Americal Statistical Association*, 85(411):699–704. 154

Weiss, Y. and Freeman, W. T. (2001). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200. 55

Wendland, H. (2005). *Scattered data approximation*, volume 17. Cambridge University Press. 102

Whitcher, B., Byers, S. D., Guttorp, P., and Percival, D. B. (1998). Testing for homogeneity of variance in time series: Long memory, wavelets and the Nile river. Technical report, Water Resources Research. 28

Wilson, A. and Ghahramani, Z. (2010). Copula processes. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2460–2468, Cambridge, MA, USA. The MIT Press. 72

Xuan, X. and Murphy, K. (2007). Modeling changing dependency structure in multivariate time series. In *24th International Conference on Machine Learning*, pages 1055–1062, Corvallis, OR, USA. ACM. 29, 124, 168, 169

Yan, F. and Qi, Y. (2010). Sparse Gaussian process regression via L1 penalization. In *27th International Conference on Machine Learning*, pages 1183–1190, Haifa, Israel. Omnipress. 104

The car image in Figure 1.2(b) was used with permission from Audi USA from Flickr image 4733495126. Likewise, the satellite image in Figure 1.2(b) was used with permission from Eric Hackathorn from Flickr image 2248808589.