Learning and Using Multimodal Stochastic Models: A Unified Approach

by

Mark Edgington

M.S. Electrical Engineering
Illinois Institute of Technology, 2003

Submitted to the Department of Math and Computer Science in
Partial Fulfillment of the Requirements for the Degree of

Doktor der Ingenieurwissenschaften (Dr.-Ing.)
(Doctor of Philosophy in Engineering and Computer Science)
at the
UNIVERSITY OF BREMEN

September 2016

Thesis Supervisor: Prof. Dr. Frank Kirchner
External Reviewer: Prof. Michael Beetz, Ph.D.

Datum des Promotionskolloquiums: 9. September 2016

Gutachter: Prof. Dr. Frank Kirchner (Universität Bremen)
Prof. Michael Beetz, Ph.D. (Universität Bremen)

Acknowledgements

I dedicate this work first and foremost to my loving wife Rhonda. Without her patience and support, I would not have had the wherewithall to continue the long (and at times arduous) process of completing it. In addition, I thank my family for their support even when they may have thought at times "will he ever finish?"

There are many friends and colleagues who also deserve my thanks. From the University of Bremen and the German Research Center for Artificial Intelligence (DFKI), I thank Frank Kirchner for his faith in my ability to complete the work, and his support along the way. Yohannes Kassahun has been a wonderful colleague and friend and I value our literally hundreds of hours of discussion and thinking together on a wide range of subjects, many related to the work I present here. I also appreciate Bertold Bongardt's persistent friendship and helpful feedback. José de Gea Fernandez has also been a colleague and friend I could talk with about anything, and with whom I shared an unbelievable number of coffees. Thanks also to Thijs Jeffry de Haas, Daniel Beßler, and Tchando Kongue, students I appreciated being able to advise and work with for several years. On the administrative end, I wish to thank Petra Gerdes and Silke Völkers, whose efforts kept the process on track, despite the complexity of communicating and arranging things from across the Atlantic. I must also thank the German Research Foundation (DFG) for the years of support I received in the context of the SFB/TR 8 Spatial Cognition collaborative research center. I have always appreciated the wise and gentle leadership of Christian Freksa, and the thoughtful suggestions and feedback I have received from him and others in his work-group including among others Mehul Bhatt and Diedrich Wolter. Thanks also to Wolfram Burgard, Cyrill Stachniss, and Kai Wurm from the University of Freiburg for the various collaborations and discussions we've had.

More recently, I'd like to thank my colleagues at Hope College for their support and feedback on various parts of this work. I have nothing but gratitude for Herb Dershem for his willingness to give his time and energy to provide wise guidance and feedback – his patient and persistent encouragement was instrumental in helping me get back on track with finishing the work. Others at Hope I'd like to thank are Paul Pearson and Brian Yurk, who provided valuable feedback and discussion of ideas. They would often bear with my rambling on an idea until I found the clarity I sought.

Contents

Abstract

This dissertation presents a principled approach to representing and using instance-based knowledge. Perceptions and actions are probabilistically modelled in a unified structure which allows for simultaneous perception modelling and reasoning about desired actions. In particular, a new method for online instance-based learning of such models is presented and analyzed. This method, called Dynamic Gaussian Mixture Estimation (DGME), adapts a model's complexity to the process being modelled. The models produced by DGME are evaluated on several classification, prediction, and control applications, and its characteristics are compared with other state-of-the-art methods. In the context of control applications, an additional novel method, Gaussian Mixture Control (GMC), is introduced for precisely controlling systems that exhibit multimodality.

A Note On Notation

This document uses the following notation: vectors are written in lower-case bold (e. g. $\mathbf{a}$), components of a vector are indicated with subscripts in parentheses (e. g. $\mathbf{a}_{(1)}$), a specific instance of a vector from a set of vectors (e. g. a training set of observation vectors) is indexed using an index in parentheses as a superscript (e. g. $\mathbf{a}^{(1)}$), and matrices are uppercase and bold (e. g. $\mathbf{A}$).

All vectors are column-vectors unless transposed: $\mathbf{a}$ is a column vector, while $\mathbf{a}^T$ is a row vector.

A matrix component is indicated using a (row index, column index) subscript in parentheses (e. g. $\mathbf{A}_{(r,c)}$), row-vectors in a matrix are referred to by (row index, $\cdot$) subscripts ($\mathbf{A}_{(r,\cdot)}$), and column-vectors in a matrix use ($\cdot$, column index) subscripts ($\mathbf{A}_{(\cdot,c)}$). Block submatrices of a matrix are indicated with a 4-tuple (r_1, c_1, r_2, c_2) subscript in parentheses (e. g. $\mathbf{A}_{(1,1,2,2)}$ represents a the upper-left 2×2 block submatrix of $\mathbf{A}$, and $\mathbf{A}_{(\cdot,1,\cdot,2)}$ represents a matrix containing the first and second columns of $\mathbf{A}$).

All indices start at 1.

A matrix or vector may also have its subscript begin with a label or index. For example, $\mathbf{v}_{DC,(1)}$ represents the first component of the vector $\mathbf{v}_{DC}$. This can also be used, for example, in the case of a matrix $\mathbf{X} = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3\end{bmatrix}$, where $\mathbf{v}_i$ are column vectors. In this case, $\mathbf{v}_{2,(3)}$ would refer to the third component of vector $\mathbf{v}_2$. Likewise, for matrices, one could write $\mathbf{M}_{Q,(5,5)}$ to represent the (5,5) element of $\mathbf{M}_Q$.

Scalars and functions are *not* written in bold, and are indexed using subscripts.

Notation examples that include the conventions described above are summarized in the following table:

Table 1: Examples of Notation used in this Thesis

$\mathbf{a}$	a column-vector
$\mathbf{a}^T$	a row vector
$\mathbf{a}^{(7)}$	the 7^{th} vector in a set of vectors $\{\mathbf{a}^{(i)}\}$
$\mathbf{a}_{(1)}$	the 1^{st} component of vector $\mathbf{a}$
$\mathbf{v}_{5,(3)}$	the 3^{rd} component of vector $\mathbf{v}_5$
$\mathbf{A}_{(r,\cdot)}$	the r^{th} row-vector in matrix $\mathbf{A}$
$\mathbf{A}_{(\cdot,c)}$	the c^{th} column-vector in matrix $\mathbf{A}$
$\mathbf{A}_{(1,1,2,2)}$	the upper-left 2×2 block submatrix of matrix $\mathbf{A}$
$\mathbf{A}_{(\cdot,2,\cdot,4)}$	a submatrix composed of column-vectors 2,3,4 of matrix $\mathbf{A}$
$\mathbf{M}_{Q,(5,5)}$	the (5,5) element of matrix $\mathbf{M}_Q$
$\mathbf{I}_3$	the 3×3 identity matrix
$a := f(a, b)$	the function f operates on a and b, and then overwrites a with f's output.
$\left[\mathbf{v}_a \parallel \mathbf{v}_b\right]$	a vector formed by concatenating $\mathbf{v}_a$ with $\mathbf{v}_b$
$\mathbb{E}[\mathbf{x}]$	the expected value of vector $\mathbf{x}$

Summary of Contributions

The contributions of this thesis can be broken down into two main areas: density estimation and control. Contributions in each area are considered in what follows.

Density Estimation A density estimation method, *Dynamic Gaussian Mixture Estimation (DGME)*, was developed that exhibits the properties summarized in Table 2. It distinguishes itself from the state of the art methods in that it (1) does not assume stationarity of the modelled process, (2) adapts the model complexity in a data-driven fashion, (3) has reduced computational complexity due to its semi-parametric nature, and (4) locally adapts the bandwidth (i. e. covariance) of each individual mixture component. The state-of-the-art kernel-based methods chosen for comparison are *Adaptive Mixtures (AM), Time-evolving Adaptive Mixtures (TEAM), and Online Kernel Density Estimation (oKDE)* (described in Priebe and Marchette, 1993, Szewczyk, 2005, and Kristan, 2010, respectively).

Property	AM	TEAM	oKDE	DGME	PF Based
Possible to work with non-stationary processes	–	✓	–	✓	✓
Model complexity adaptation	✓	–*	✓	✓	✓
Semi-parametric (vs. non-parametric)	✓	✓	(hybrid)	✓	–
Per-component bandwidth adaptation	✓	✓	–	✓	N/A

* An upper bound must be chosen on the number of components in order for the garbage-collection stage to work.

Table 2: Characteristics of state-of-the-art online density-estimation methods.

The AM and oKDE methods assume that a stationary process is being modelled, which raises the question of whether there is a significant advantage in using these methods over traditional batch density-estimation methods (which also rest on the assumption of stationarity). In contrast, the TEAM and DGME methods can both be configured so that the model can gradually "forget" information that is deemed less important, or that hasn't been observed for a long time. The AM and oKDE methods share with DGME the property that the model complexity is adapted to fit data as it is observed, with no assumptions made on the maximum number of components that any model might have.

The TEAM method, on the other hand, suffers from the need to identify an upper bound on the number of components assumed to be in the "true" distribution being modelled. If this bound is set too high, then the number of components in the model will always be much higher than the necessary number of components. Furthermore, the claim that TEAM is suitable for modelling non-stationary processes is incompatible with specifying a mixture component limit, since a non-stationary process may change over time not only in terms of its parameters, but also in terms of its complexity (which is correlated with the number of components the model has).

Control A method of controlling systems which have been identified with DGME has been developed, called *Gaussian Mixture Control (GMC)*. The GMC Method distinguishes itself from state of the art control methods primarily in the sense that it is able to control systems exhibiting *multimodal* models. In other words, a system whose behavior may fall under multiple distinct modes of operation can be controlled in order to maximize the likelihood of achieving a desired outcome *without needing to manually identify and characterize each mode of operation.* A further advantage of GMC is it's ability to handle complex and nonlinear[1] systems. A majority of modern control methods assume that the system to be controlled will either be intrinsically linear, or linearized using an inverse function when the system is non-linear. While several physical systems can be made to behave as a linear system, many physical (and non-physical[2]) systems exhibit nonlinear properties which are difficult to linearize. To control these kinds of systems requires control methods which are able to handle these nonlinearities. GMC takes advantage of its ability to use several *locally* linear models to approximate nonlinear system behavior. Table 3 compares features of four different control methods alongside GMC.

Property	PID	MPC	SS	GMC	Switching SS
Handles linear / linearized systems	✓	✓	✓	✓	✓
Handles non-linear systems		✓		✓	
Handles multi-modal systems				✓	✓
Uses uncertainty information		✓		✓	

Table 3: Characteristics of state-of-the-art control methods.

[1] A nonlinear system is characterized by a transfer function h where relations like $y_1 = h(x_1)$ and $y_2 = h(x_2)$ *don't* imply $ay_1 + by_2 = h(ax_1 + bx_2)$.

[2] An example of a non-physical system is a business or economic process.

Chapter 1

Introduction

1.1 Unified Prediction and Perception

In the past couple of decades there has been an increasing amount of research done which suggests that it may be likely that the actions (and predictions that are used in selecting actions) performed by humans are directly connected with perceptions, insofar as they may stem from the same structures in the brain (see Hommel et al., 2001; Hesslow, 2012). One could therefore postulate that this is an important characteristic behind the flexibility and power of the human mind. In particular, there is much research supporting the importance of prediction in cognitive processes. As mentioned by Bubic, Von Cramon, and Schubotz (2010), "predictive processing represents one of the fundamental principles of neural computations [and] errors of prediction may be crucial for driving neural and cognitive processes as well as behavior". One of the motivations of the work in this thesis is to find and exploit such a representation, where the representational structure can be used for storing perceptions (having a memory-like role), and can simultaneously be used for prediction, reasoning, and acting.

1.2 Lifelong Learning

In real-life scenarios, humans and animals tend to improve their models of the world in a highly adaptive manner. An interesting question to consider is "how do humans model

their environment?" It is clear that whatever the modelling process used by human beings is, it is dynamic in its ability to model new situations and reshape existing models as new information becomes available. In contrast, mathematical modelling techniques currently used in the scientific community are quite inflexible when it comes to reshaping due to new information. Typically the modelling process goes something like this:

1. A human makes a decision about which domain s/he wishes to model, and considers what kinds of mathematical models might best fit the chosen domain (e. g. a linear model).

2. What is deemed to be a sufficient amount of data is collected in the chosen domain, and the different selected models are fit to this data.

3. Each model is evaluated (e. g. using a technique such as cross-validation) in order to assess the likelihood of a particular model given the collected data.

4. The highest-likelihood model is chosen and used for analysis and prediction.

While for many applications this is a perfectly legitimate approach to modelling, it is not sufficient for "lifelong-learning" agents, which attempt to adapt their models of the world to their observations over time. For such a system, we wish to approximate human cognition, in which prior knowledge is combined with new observations to form new knowledge – in other words, a Bayesian approach is suitable (see Austerweil et al., 2015). Furthermore, it must be possible to reshape existing models without losing the information that has been stored in them. This assertion is based on the assumption that intelligent agents are physically incapable of storing individually every single observation which they make during their lifetime. These observations must, therefore, be summarized and compressed in some manner. The so-called "paradigm shift" which humans are known to experience at different points in time supports the idea that a human can hold multiple hypotheses simultaneously in his/her mind, and at some point decide to switch from one to another. On the other hand, it is not uncommon for a person to start out with one concept about the world, and later to slightly refine that concept (model).

In summary, intelligent agents need both the ability to simultaneously entertain multiple hypotheses about the world, as well as the ability to refine and reshape (possibly combine) existing hypotheses. Though it is not entirely clear how to best make use of multiple

hypotheses for any given domain, one traditional approach is to use maximum likelihood. Using a maximum likelihood criterion would result in a similar decision policy to the one used in the previously enumerated modelling process, where the most likely model is selected. Depending on a single model to understand one's environment, however, can have serious limitations – it is often beneficial, therefore to be able to reason by using information from several models simultaneously. This property of simultaneously maintaining and utilizing several hypotheses is already present in Gaussian Mixture Models, in which each Gaussian distribution can be considered as a single (locally linear) hypothesis. Most GMM based modelling approaches are limited in that:

1. the number of Gaussians (i. e. hypotheses) which can be entertained in a given model is usually fixed a priori, preventing the model from expanding, should new evidence become available that is unexplained by any of the existing Gaussians.

2. the dimensionality of the Gaussians in the model is fixed a priori, preventing the model from becoming more expressive in terms of the breadth of perceptual data that can be used.

The first of these limitations is addressed by the methods presented in this dissertation, while the second limitation may be partially addressed by incorporating probability distributions such as the Indian Buffet Process (Griffiths and Ghahramani, 2006).

1.3 Cognitive Framework used in this Thesis

Inspired by the previous observations about the requirements of a cognitive framework for perception and prediction that approximates certain aspects of how the human brain operates, this thesis focuses on a unified scheme by which perceptions can be directly integrated into a sort of 'working memory', which can simultaneously be used for predictive activities. Cognitive frameworks which have been proposed up to this point typically depend on separate representations for memory and predictive tasks, meaning that there must be a conversion between these representations. When the predictive facilities of a cognitive agent are used for actively controlling the agent's perception, separate representations become a bottleneck, requiring conversion between

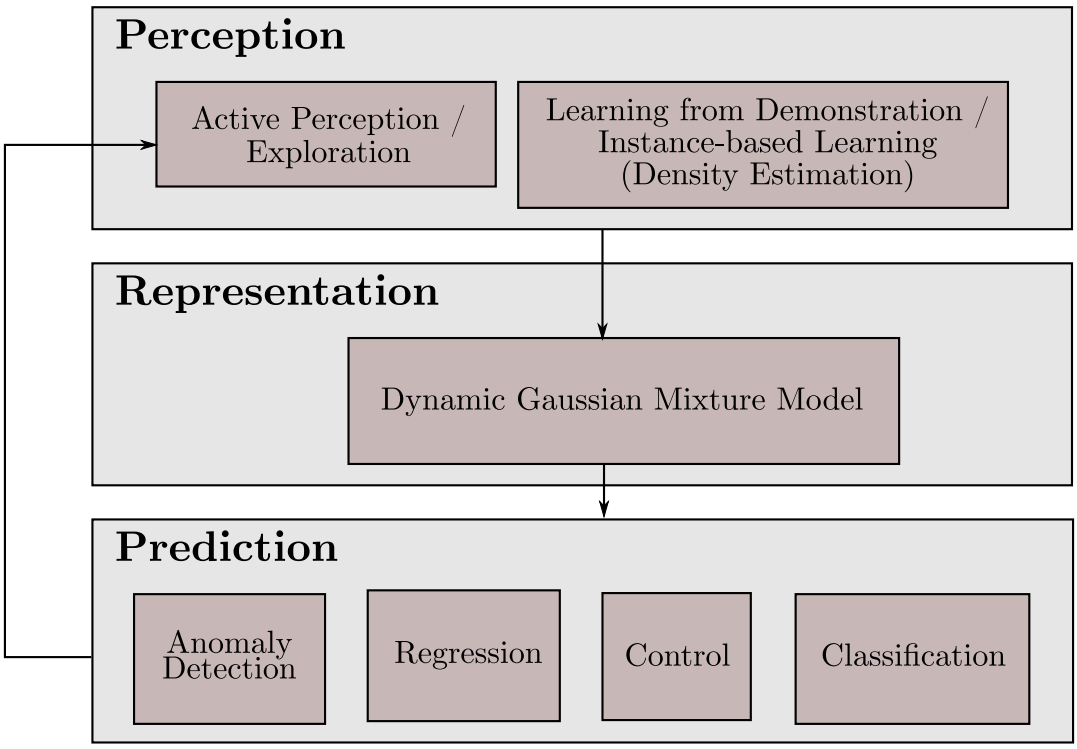

Figure 1.1: Overview of concepts relevant in this thesis.

representations after each new perception. For a constant stream of perceptions, this overhead is unacceptable.

Figure 1.1 shows a graphical depiction of some of the important concepts used in this dissertation. One of the core concepts that is integral to the new methods introduced is the *knowledge representation* into which *perceptions* can be directly integrated. Simultaneously, this representation can be used for *predicting* future events based on current observations of the world, or simulating events based on hypothetical scenarios. The integration of perceptions is achieved using a novel density-estimation technique introduced in this thesis named *Dynamic Gaussian Mixture Estimation* (DGME), which is discussed in more detail in Chapter 3. Various predictive tasks are investigated in the thesis, including regression and classification (see Chapter 4), anomaly-detection and control (see Section 5.4).

The knowledge representation chosen for this framework is probabilistic, using a continuous joint probability density function. The next section will elaborate on the rationale for choosing such a representation.

1.4 Probabilistic Modelling and Control

Probabilistic models are becoming increasingly important in the task of modelling complex processes. In the past, modelling efforts frequently made use of simple deterministic models, typically in the form of a mathematical function approximating the relationship between the input and output of a process. While this is sufficient for simple systems having few components, the required effort to model a system in this way increases exponentially as the number of components increases, due to the need to model not only the individual components, but also interactions among the system's different components. Most probabilistic modelling techniques today are still largely based on this classical modelling paradigm, where a fixed functional form is chosen a priori to represent a system's behavior, and in addition to learning/choosing this function's parameters, noise parameters are also chosen/learned which model the uncertainty around the points represented by the function.

Despite the improvement over deterministic models, these models which consist only of functions with noise parameters remain limited in the range of systems that they are capable of accurately modelling. While a large majority of systems are able to be modelled using such techniques, using these models in most modern-day control applications requires strong assumptions about the needed characteristics of these systems, such as the following:

- The modelled system must be linear and time-invariant.
- The system must be representable with a bijective mapping.

As a result, much effort is spent designing systems which meet these assumptions, and working on methods for linearizing non-linear systems. However, nonlinear systems that can be represented using multimodal probability distributions can offer certain efficiency or performance benefits which are sacrificed when designing a system to be linear and bijective. Furthermore, as a system's complexity increases, it becomes increasingly difficult to guarantee that these assumptions can be kept, due to nonlinear and multimodal effects that arise as a result of inter-component interactions. It is therefore desirable to have a control method that does not require a system to meet the above assumptions.

This thesis tries to address this problem by dealing with the particular class of stochastic

(e. g. probabilistic) models that can be represented as *multimodal* probability distributions[1]. A multimodal distribution is characterized by having a probability density function (pdf) which exhibits multiple *modes* – that is, instead of the pdf representing one single maximum over the distribution's support, several distinct maxima (i. e. modes) are present, which are sufficiently "far"[2] from each other to conclude that they cannot be reasonably replaced by a unimodal distribution.

This thesis addresses two central topics related to multimodal models:

1. How to effectively *learn* multimodal models.

2. How to effectively *use* multimodal models (e. g. for prediction, pattern-discovery, classification, and control).

Both of these topics will be discussed in detail.

The first topic, *learning* multimodal models, is discussed in Chapter 3, where two new methods for building such models are presented.

The second topic, *using* multimodal models, is discussed in Chapter 4, which covers the application of such models for pattern-discovery and classification, as well as in Section 5.4, where a new control method is presented which allows a multimodal pdf of a process to be used in order to accurately and predictably control the system.

This method makes a break from traditional control methods, in that it takes advantage of the multimodal information contained in a model, and doesn't depend on the assumption that the behavior at any point in the process' state-space can be modelled with a unimodal distribution. The Mixture of Gaussians (MoG) model, also known as a Gaussian Mixture Model (GMM), has been chosen as the basis of the knowledge representation structure used by the new methods presented in this dissertation. The reasons for choosing this instead of other representations such as a more general mixture model are threefold:

1. Using Gaussian mixture-components for the entire model allows for a principled

[1] Throughout this thesis, the terms "model" and "probability distribution" will be used interchangeably, when the meaning of "probability distribution" is clear from the context.

[2] Whether a distribution's modes are far enough from each other depends on the characteristics of the process being modelled. These can be determined either manually or by applying a suitable heuristic.

decision making policy regarding whether or not to update an existing component with new observations, and when to create new components in the model.

2. Gaussians are mathematically convenient, providing simple closed-form solutions to calculating conditional and marginal probability densities. This allows for efficient and accurate calculations.

3. Arbitrary density functions can be represented as a weighted sum of Gaussians (i. e. a GMM). Thus, if a method supports adapting the number of mixture components to the observations, the assumption that the data must be Gaussian-distributed is not required.

Section 2.2.2 serves as an introduction to the basics of GMMs, and discusses the factors involved in using them for modelling.

1.5 Summary

In this chapter, I have motivated the need for a unified and flexible framework for knowledge representation, capture, and use. Additionally, a case has been made for using a Gaussian Mixture Model as the basis for such a representation. The remaining chapters provide a foundation for understanding the techniques used in this thesis, and present a detailed explanation, analysis, and evaluation of the new methods contributed by this thesis.

Chapter 2 reviews the most important topics and works related to this thesis, including model representations, as well as modelling, control, and regression techniques. The first major contribution of this thesis, the *Dynamic Gaussian Mixture Estimation* (DGME) algorithm, is presented in Chapter 3. This algorithm is used to perform online "black-box" modelling of processes.

After a model has been acquired, it is helpful to be able to do something with it. Chapter 4 and sections 5.3 and 5.4 present different ways of making use of a model learned with DGME.

Prediction and classification techniques and examples are presented in Chapter 4. On

the other hand, Section 5.3 addresses the topics of robot-localization control. In contrast to the control method presented in this chapter, a further contribution of this thesis is presented in Section 5.4: a novel and robust control technique, called *Gaussian Mixture Control* (GMC), which makes intelligent use of a model's multimodality.

Chapter 2

Foundations

2.1 Model Representations

At the heart of any cognitive system is its ability to model (whether explicitly or implicitly) its world, so that predictions and intelligent decisions can be made based on this model. All such models depend on the existence of some underlying representation by which the model stores knowledge. This section reviews several of the most popular knowledge-representation schemes that are used with continuous-valued data[1], and compares their suitability for use in different kinds of modelling applications. The representations can roughly be categorized in terms of each of the following characteristics:

- Linear / Nonlinear
- Local / Global
- Probabilistic / Deterministic
- Parametric / Non-parametric / Semi-parametric

These characteristics, and the representations that can be associated with them, are discussed in the following subsections. Table 2.1 shows an overview of all the models discussed, and the characteristics each possesses.

[1]Because the methods in this thesis work with continuous-valued data, we only consider continuous-valued representations in this section.

Table 2.1: Model Representation Characteristics: a summary of various model representations and their characteristics.

Representation	Representation Characteristics			
	Linear (L) / Non-Linear (NL)	**Local (L) / Global (G)**	**Probabilistic (P) / Deterministic (D)**	**Parametric (P)/ Non-parametric (N)**
Single Gaussian	L	G	P	P
GMM	NL approx.	L	P	Semi-parametric
Unscented Kalman Filter	NL approx.	G	P	P
SVM	NL	G	D	P
RBF Network	NL approx.	L	P	Semi-parametric
Classical Physics Eqns	Both	G	D	P
Particle Filter	NL	L	P	N
Kalman Filter	L	G	P	P

2.1.1 Linear / Nonlinear

Linear model representations are characterized by the assumption that the process being modelled possesses a linear characteristic. In other words, the relationships between variables used to represent the observations (i. e. inputs, outputs, and environmental state) can be treated as linear. In contrast, nonlinear representations take account of nonlinearities in observations of the process.

An example of a linear model is a single-Gaussian model, whose covariance matrix models the linear dependencies (i. e. correlations) between the components of the observation-space (i. e. the *support* of the distribution). Despite the fact that individual Gaussians are linear, mixtures of Gaussians are capable of accurately approximating nonlinear processes. Another representation which captures the nonlinearities of a process is the unscented Kalman filter, which models nonlinearities in terms of sigma points (see (Sebastian Thrun, Burgard, and Fox, 2005)). Underlying this model, however, is the linear Kalman filter model, where the process nonlinearities have been linearized through the use of sigma points.

2.1.2 Local / Global

Global modelling techniques can roughly be identified as those that provide a single equation to represent the relationships between process-variables *over the entire space on which these variables are defined*. This equation describes a global trend that is assumed to remain consistent over a model's support. Such models differ from *local* models which, though they can be defined over the entire space of the process-variables, have fundamentally different equations describing the model behavior depending on the region of this variable space that is "active" (i. e. relevant for control or prediction because of the current state of the process) at a given point in time. Such models, whether they are defined over the entire variable space, or only some subregion of this space, are collectively referred to as local models, because they operate differently depending on the local region of variable space which is currently relevant.

Support vector machines are an example of a of global modelling technique. Though in general, they are used for classification, and not prediction, they represent a process

globally, and do not have region-specific classification behavior. Instead, they attempt to maximize the separation between class-boundaries for *all* training examples.

On the other hand, models such as GMMs and Radial Basis Functions (RBFs) are good examples of local models, due to the "local" nature of their underlying components, whose values taper off as an observation vector moves farther away from the Gaussian or basis-function's center. Likewise, the methods that use such models can be considered to be local.

When considering probabilistic representations, one can furthermore draw a connection between a model's globality / locality and a model's unimodality / multimodality. Multimodal models (in the probabilistic sense of multimodality) can typically be associated with locality, where each mode corresponds to a local region. Unimodal models, on the other hand, are more closely related to global models, even if they are most valid around the distribution's first-order statistical moment (i. e. mean).

2.1.3 Probabilistic / Deterministic

Probabilistic modelling techniques are characterized by the explicit modelling of a process' uncertainty and/or variance. When a process is stochastic in nature, it does not behave identically each time it finds itself in a certain (modelled) state. Because it is difficult – perhaps impossible – to fully observe the complete state of a (continuous) process, there will always be some (perceived) stochastic behavior of the process. Whether the process is considered stochastic or not, essentially depends on whether the perceived stochasticity is deemed to play an important role with respect to the goals behind the modelling.

The stochasticity of a system is typically interpreted in two ways (which can be seen as opposite sides of the same coin):

1. *Variance*: Different outcomes represent a "spread" of outcomes which the process exhibits (with respect to the *expected* value of an outcome).

2. *Uncertainty*: Different outcomes represent a measure of how certain (or likely) a given measurement is.

Though both of these interpretations are valid, in multimodal pdfs, there is an additional concept that has not received much treatment in the literature. While the occurrence of a set of observations that are "near" to each other in a metric sense represents the variance of this observation around the "center" of these observations, the existence of multiple centers introduces the concept of *confidence*. Confidence can be thought of as a measure of the overall likelihood that a measurement will belong to a given mode (i. e. that it is near[2] the center of a particular mode). The model can specify that it has higher confidence in some modes than in other modes. Similar to models that use transition-probabilities to describe transitions between discrete states, the concept of *confidence* also specifies a probability over a set of discrete modes. In contrast to probabilistic modelling techniques, deterministic techniques ignore the above-mentioned stochastic characteristics of a process. A simple example of a deterministic technique is a process that is modelled in terms of classical physics equations, where none of the variables are random-variables (i. e. variables modelled as having some probability distribution).

2.1.4 Parametric / Non-parametric

Contrary to intuition, parametric models are not distinguished from non-parametric models by the existence of parameters in the model. Instead, they are distinguished by the *number* of parameters in the model with respect to the model's complexity.

A parametric model generally possesses a *fixed* number of parameters, which (globally) represent the behavior of a process. In contrast, a non-parametric model typically adds new parameters as the number of observations increase. A particle-filter is an example of a nonparametric modelling technique, where particles are very closely connected with individual observations. Kalman filters, on the other hand, are parametric in the sense that the number of parameters required to specify the filter is fixed.

In addition to parametric and non-parametric methods, there exists a third classification of modelling methods with respect to the number of parameters they have: *semi-parametric*. The idea behind semi-parametric methods is that instead of generating parameters for *every* (recent) observation, the method *summarizes* the observations locally, and thus the

[2]Being "near" a center is relative to the variance around the center. A metric which encapsulates this concept is the Mahalanobis distance.

number of parameters in the model grows significantly more slowly than the number of parameters in a non-parametric method would grow. A Gaussian Mixture Model serves as a good example of a semi-parametric modelling method.

2.2 Density Estimation

In this section we consider a few methods that are commonly used for density estimation. A basic density estimation method familiar to most people is the histogram. In the case of histograms, the space spanned by the observation vectors is divided into 'bins'. Whenever an observation occurs which happens to fall into one of the bins, the value associated with the corresponding bin (initially zero) is incremented. After enough observations, the bin values begin to represent the probability density of an observation. A problem with histograms is that the bin boundaries are arbitrarily determined (they depend on the selected resolution of the histogram), and these boundaries may not allow the density to be properly represented. As an example, see Figure 2.1, which shows how a histogram differs significantly from the true density when a bin boundary lies in the center of a mode.

Fortunately there are several other density estimation methods available which avoid the problems associated with bin boundary locations. Some of the more common methods are discussed in the following subsections.

2.2.1 Kernel Density Estimation

A straightforward method known as Kernel Density Estimation (KDE) offers a significant improvement over histograms in terms of the accuracy with which a density can be estimated. The reason it is called *Kernel* Density Estimation is because it requires the selection of a kernel function K (also known as a Parzen Window (Parzen, 1962)). A kernel is any function that takes a single argument (typically representing a distance), and that returns a scalar (typically representing a weight). This function evaluates the probability density at a given point. Specifically, given a set of observation data vectors $D = \{\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_M\}$, the (unnormalized) probability density at a point $\mathbf{q}$ is the sum of

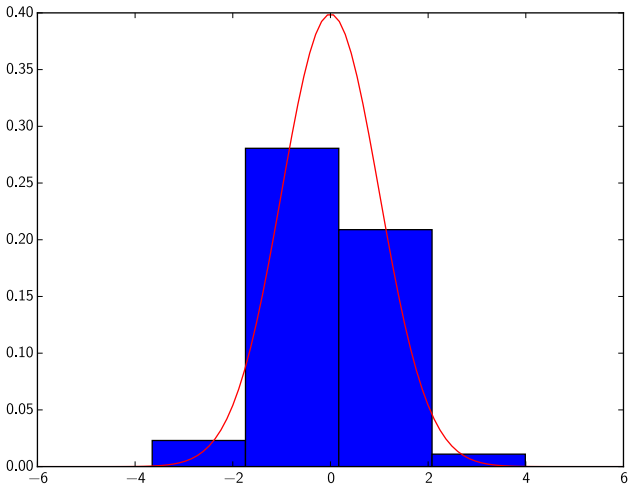

Figure 2.1: Illustration of one weakness of histograms: when a bin boundary divides a mode of the true density, the resulting histogram estimate (based on randomly sampled data from the red Gaussian curve) can give an incorrect indication of where the maximum occurs.

the kernel function evaluated over all distances between $\mathbf{q}$ and each observed point:

$$p(\mathbf{q}) \propto \sum_{i=1}^{M} K(\|\mathbf{q} - \mathbf{d}_i\|/h). \tag{2.2.1}$$

Here, h is a *bandwidth* parameter that influences the distance metric.

Note that if the chosen kernel function is a Gaussian density function, then

$$K(\mathbf{x}) = N(\mathbf{x} - \boldsymbol{\mu}; 0, \boldsymbol{\Sigma}), \tag{2.2.2}$$

and the pdf represented using this kernel with KDE has a form identical to an N-component GMM, under the constraint that all component weights are identical. In both cases (a KDE with Gaussian kernel, and a GMM), it is also possible to apply a k-Nearest-Neighbors (kNN) type approach to evaluating the density, in which case (2.2.1) becomes

$$p(\mathbf{q}) \propto \sum_{i \in \mathcal{N}} K(\|\mathbf{q} - \mathbf{d}_i\|/h), \tag{2.2.3}$$

where $\mathcal{N}$ is a set representing some neighborhood of points around $\mathbf{q}$. In practice, choosing an appropriately sized neighborhood will give nearly the same results as summing over all data points, because the contribution of any Gaussian function whose mean is far from the query point is approximately zero. The key to making this approach computationally feasible is indexing the points or Gaussians in such a way that the neighborhood of a certain query point can be quickly determined.

Note that in (2.2.2), the covariance-matrix parameter of the kernel function need not be constant, in general. Instead, the "shape" of the covariance matrix must be selected or computed according to some chosen covariance function. There are a few different approaches to selecting the covariance matrix: (1) use a fixed covariance matrix, (2) fit each covariance matrix to the local density around the observation, using a local density estimate from the k nearest neighbors, or (3) fit each covariance matrix to the local density around each particular observation, using a local density estimate from the nearest neighbors within a certain fixed-size neighborhood. These three methods are illustrated in Figure 2.2.

2.2.2 Estimating Densities with Gaussian Functions

The Gaussian probability distribution is commonly used to estimate densities. Depending on the dimensionality of the process being modelled, the univariate (i. e. 1-dimensional) or multivariate Gaussian pdf is utilized, and depending on the complexity of the process, either a single Gaussian function, or a mixture of Gaussian functions can be used.

2.2.2.1 1-dimensional Gaussian

The well known 1-dimensional Gaussian (a.k.a. Normal) probability density function (pdf) can look like one of several "bell curves", as shown in Figure 2.3. The defining statistics of a 1-dimensional Gaussian pdf are the mean μ, and the variance σ. We can think of the pdf as a way of describing the outcome of a particular event that is

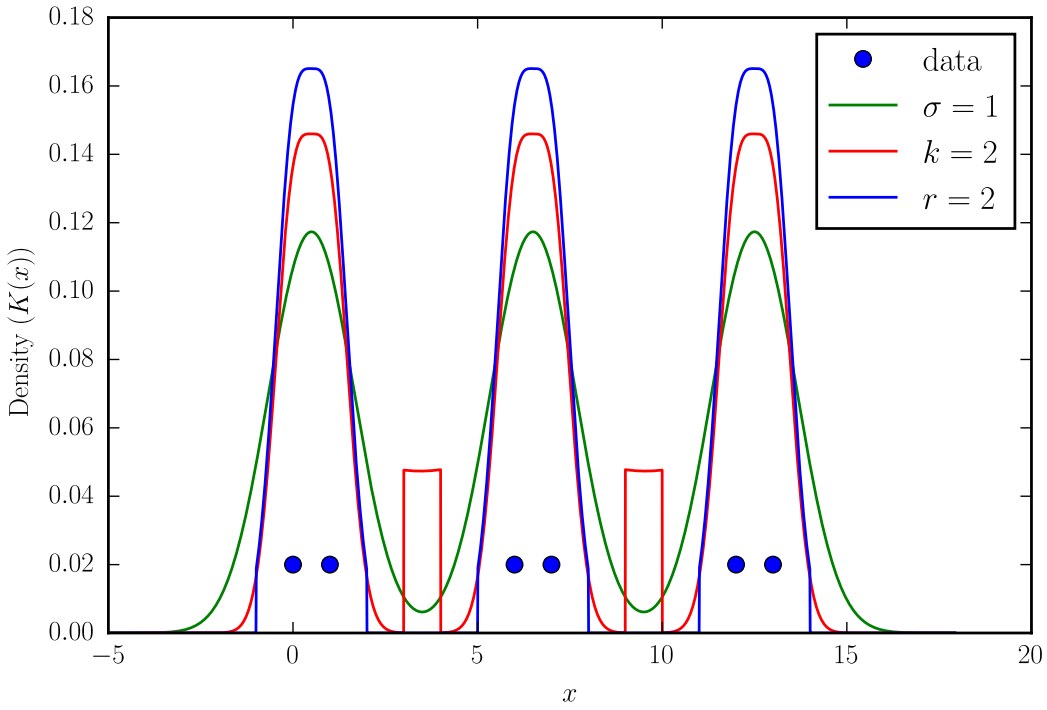

Figure 2.2: Different ways of choosing a covariance matrix for Gaussian-kernel KDE (shown in 1D for clarity). The 1D data is plotted along the $y = 0.02$ axis. Shown are the normalized KDE functions for the case where (1) the covariance is a fixed σ^2, (2) the covariance varies based on the k nearest neighbors, and (3) the covariance varies based on all neighbors within a fixed-neighborhood of $x \pm r$.

repeated multiple times. If the repeated event is, for example, the measurement of the temperature at some location under certain conditions, then the outcome will be a scalar value representing the measured temperature. The mean gives us a sense of the most commonly occurring outcome[3], and the variance helps describe the amount that a particular outcome will *vary* from the mean.

[3]In the case of a Gaussian function, the mean is always identical to the median and mode.

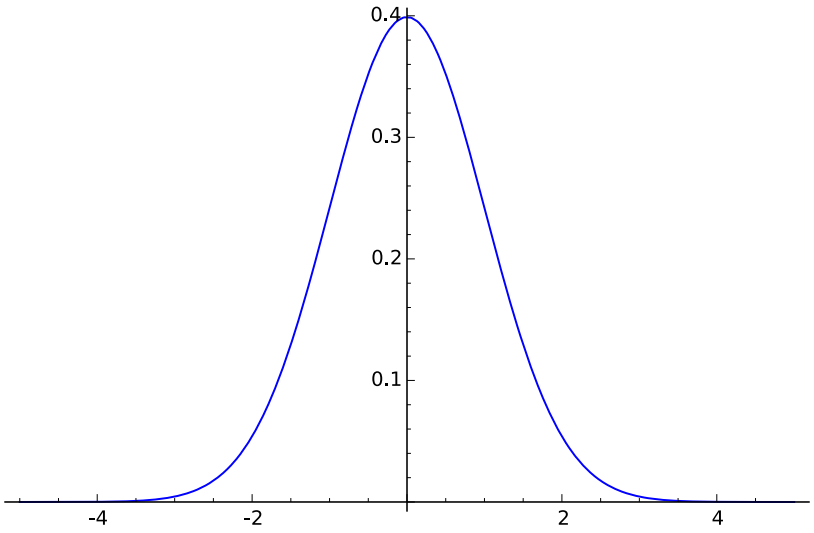

Figure 2.3: A 1D Gaussian pdf with $\mu = 0$ and $\sigma = 1$.

2.2.2.2 2-dimensional Gaussian

While a 1-dimensional Gaussian pdf is useful in modelling events that make a single measurement, if we wish to model a series of events, each involving 2 or more measurements (e. g. temperature and pressure), then a *multi-dimensional* Gaussian pdf can be used. An example of a 2-dimensional Gaussian pdf is shown in Figure 2.4, where the two measurements are represented by the two random variables X and Y. Notice the analog between the bell-curve in the 1-dimensional case, and the "bell-surface" in the 2-dimensional case. Just as the shape of the bell-curve in the 1-dimensional case depends on the value of σ, for the 2-dimensional case, the bell-surface shape depends on *three* values: σ_X, σ_Y, and σ_{XY}. The σ_X value represents the 1-dimensional σ value for the X measurement (as if we were only taking that measurement, and not measuring Y). Similarly, σ_Y represents the 1-dimensional σ value for the y measurement. The σ_{XY} value is what distinguishes a 1-dimensional Gaussian pdf from a 2-dimensional Gaussian pdf. It represents the *correlation*[4] between X and Y. Correlation can be understood in relation to the *mean* values of X and Y, which are μ_X and μ_Y, respectively.

If, after taking several measurements of X and Y, one observes that $Y - \mu_Y$ is often

[4]The term correlation is used loosely here, to mean the co-occurrence of random events. It does not refer to the Pearson Correlation Coefficient. There is, however, a simple relationship between the correlation matrix P and the covariance matrix Σ: $P = D^{-1}\Sigma D^{-1}$ where $D = (\text{diag}(\Sigma))^{1/2}$.

around *twice* the value of $X - \mu_X$, then an appropriate value for σ_{XY} is 2. If, on the other hand, one was to observe that $Y - \mu_Y$ is often around *one half* the value of $X - \mu_X$, *but having the opposite sign*, then an appropriate value for σ_{XY} is -0.5. In the first case, $\sigma_{XY} \approx 2$ means that X and Y are *positively correlated*. Likewise, when $\sigma_{XY} \approx -0.5$, X and Y are *negatively correlated*. If $\sigma_{XY} \approx 0$, there is (almost) no correlation between X and Y. Note that this *does not* imply that these two random variables can be considered to be statistically independent of one another (i. e. that the measurement of one variable has no bearing on the likelihood of what the other variable's measured value will be). Independent variables have zero correlation, but zero correlation does not require independence.

2.2.2.3 N-dimensional Gaussian

As the number of measurements involved in a random process' event increases, so does the dimension of the multivariate Gaussian required to model such a random process.

While the 1-dimensional Gaussian pdf equation has the form

$$f_X(x; \mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}, \tag{2.2.4}$$

the N-dimensional Gaussian pdf equation is written using the *N-dimensional column vectors* $\mathbf{x}$ and $\boldsymbol{\mu}$, and the $N \times N$ *matrix* $\boldsymbol{\Sigma}$ as

$$f_{\mathbf{X}}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{N/2}} \frac{1}{\|\boldsymbol{\Sigma}\|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}. \tag{2.2.5}$$

The column-vector $\boldsymbol{\mu}$ is called the *mean vector*, and the matrix $\boldsymbol{\Sigma}$ is called the *covariance matrix*.

For a 2-dimensional Gaussian pdf, $N = 2$, and the mean vector looks like

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}, \tag{2.2.6}$$

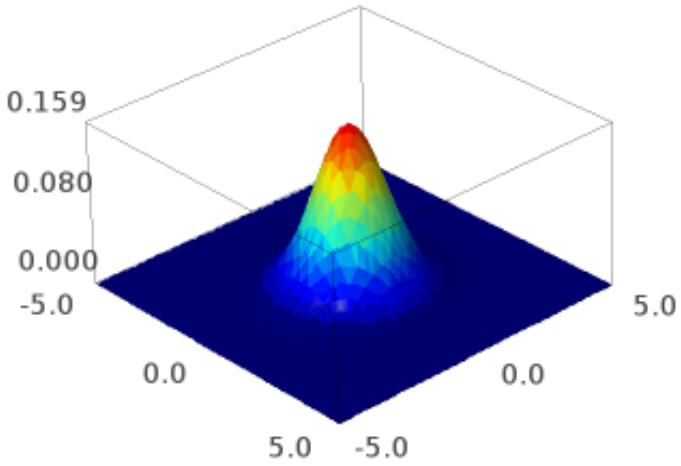

Figure 2.4: A 2D Gaussian pdf with $\mu = (0, 0)$ and $\Sigma = I_2$.

and the covariance matrix looks like

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_X & \sigma_{XY} \\ \sigma_{YX} & \sigma_Y \end{bmatrix}. \tag{2.2.7}$$

Notice that this matrix contains all of the values mentioned previously as needed for determining the bell-surface shape. In addition to those mentioned previously, there is the lower-left element of the matrix, σ_{YX}. Because correlation between two variables is a symmetric function, the value of σ_{YX} is the same as the value of σ_{XY}.

For an N-dimensional Gaussian pdf, we replace the two random variables X and Y of the 2-dimensional case with N random variables in the N-dimensional random vector $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & \cdots & X_N \end{bmatrix}^T$. The mean vector now looks like

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{bmatrix} \tag{2.2.8}$$

and the covariance matrix looks like

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1N} \\ \sigma_{21} & \sigma_{22} & & \vdots \\ \vdots & & \ddots & \\ \sigma_{N1} & \cdots & & \sigma_{NN} \end{bmatrix}. \tag{2.2.9}$$

Notice that the only correlations represented in this matrix are between *pairs* of random variables. Also, as with the 2-dimensional case, the entries below the diagonal are symmetric with those above the diagonal. If we map all of the non-redundant values in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ to a single parameter vector Θ, then the number of parameters, $\|\Theta\|$, needed to represent a unimodal multivariate Gaussian function is

$$\begin{aligned} \|\Theta\| &= \overbrace{N}^{\text{parameters in } \boldsymbol{\mu}} + \overbrace{\left[N + \frac{N(N-1)}{2}\right]}^{\text{parameters in } \boldsymbol{\Sigma}} \\ &= \frac{N(N+3)}{2}. \end{aligned} \tag{2.2.10}$$

Therefore, the storage complexity of a unimodal N-dimensional Gaussian pdf is $\mathcal{O}(N^2)$ if we place no constraints on the independence of the random variables being modelled.

2.2.2.4 Gaussian Mixtures

While a unimodal Gaussian pdf can represent random processes that have only one neighborhood of commonly observed measurement values (i. e. the measured values tend to fall within some neighborhood around $\boldsymbol{\mu}$), there are several processes for which this assumption is too restrictive. Take as a simple example a 3-dimensional process exhibiting behavior approximating that of an *exclusive-or* function (which we represent with the $\oplus$ binary operator). In this case, the observation vector can take on (within some tolerance) the values

$$\mathbf{X} \in \{\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T, \begin{bmatrix}0 & 1 & 1\end{bmatrix}^T, \begin{bmatrix}1 & 0 & 1\end{bmatrix}^T, \begin{bmatrix}1 & 1 & 0\end{bmatrix}^T\}, \tag{2.2.11}$$

where each vector has the form $\begin{bmatrix}X_1 & X_2 & X_1 \oplus X_2\end{bmatrix}^T$. Each vector component takes on one of the Boolean values 1 or 0, indicating true or false, respectively. If we were to try to fit the parameters of a 3-dimensional Gaussian to represent this process, we would end up with something similar to the Gaussian depicted in Figure 2.5, which has the following parameters[5]:

$$\boldsymbol{\mu} = \begin{bmatrix}0.4676 & 0.4803 & 0.4417\end{bmatrix}^T \tag{2.2.12}$$

and

$$\boldsymbol{\Sigma} = \begin{bmatrix}0.2518 & 0.0306 & 0.0101 \\ 0.0306 & 0.2525 & 0.0175 \\ 0.0101 & 0.0175 & 0.249\end{bmatrix}. \tag{2.2.13}$$

If this fitted Gaussian were subsequently used in order to predict the value of $X_3 = X_1 \oplus X_2$ given the values of X_1 and X_2 (how this is done is discussed in detail in Section 4.2), the values predicted are shown in Table 2.2. This clearly show that the model has failed to

[5]These parameters were estimated using 100 vectors uniformly selected from among the four vectors in (2.2.11), with $0.05 \cdot U(0,1)$ noise added to each, where U is a uniform random variable distributed over the interval $[0,1)$.

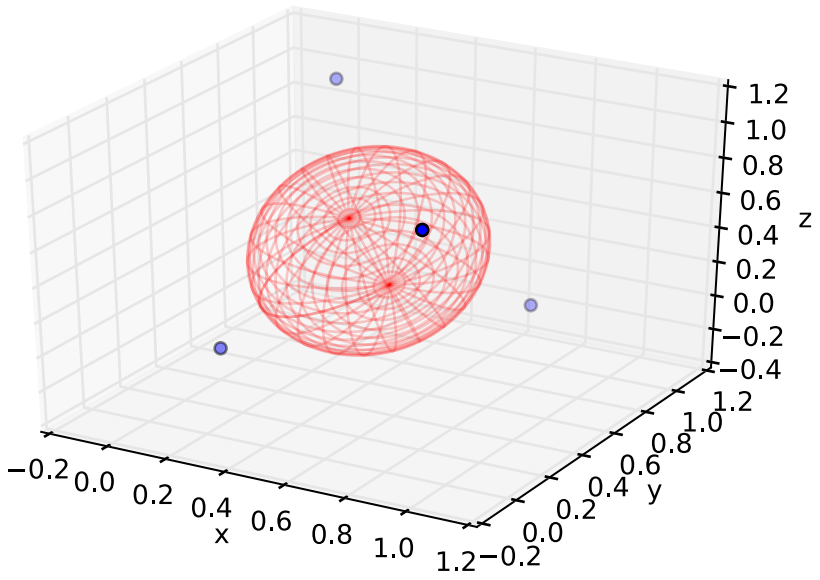

Figure 2.5: An attempt to model the XOR relationship using a single 3-dimensional Gaussian pdf. The smaller blue points represent the function $z = x \oplus y$, and the larger red ellipsoid represents the 1-σ equiprobability surface of the Gaussian which is estimating the density of noisy $[x\ y\ z]^T$ points.

adequately represent the relationship between the variables.

In order to handle more complex processes such as this, one can use a Gaussian Mixture Model (GMM), which consists of multiple Gaussian functions, each having their own set of parameters. Most complex processes can be decomposed into several linear subprocesses which are active in some subregion of the overall observation space. Each of these subprocesses can be modelled by one of the Gaussian functions in the mixture model.

For example, a 2-component GMM can be fit to the XOR data that failed to be represented with a single Gaussian. The resulting model is depicted in Figure 2.6, and the prediction results are in Table 2.3.

A GMM represents a probability density function $p(\mathbf{x})$, and can be thought of as a set of

Table 2.2: Results of predicting $X_1 \oplus X_2$ after it has been modelled using a single Gaussian whose components are fit to the noisy data $\begin{bmatrix} X_1 & X_2 & X_1 \oplus X_2 \end{bmatrix}^T$. Prediction results are poor since a single Gaussian is only able to model linear relationships, and $\oplus$ is a nonlinear function.

X_1	X_2	True $X_1 \oplus X_2$	Predicted $X_1 \oplus X_2$
0	0	0	0.3952
0	1	1	0.4608
1	0	1	0.4274
1	1	0	0.493

Table 2.3: Results of predicting $X_1 \oplus X_2$ after it has been modelled using a 2-component GMM whose components are fit to the noisy data $\begin{bmatrix} X_1 & X_2 & X_1 \oplus X_2 \end{bmatrix}^T$. Prediction results are good since the $\oplus$ function can be represented by two linear relationships, one for each state of the X_1 parameter.

X_1	X_2	True $X_1 \oplus X_2$	Predicted $X_1 \oplus X_2$
0	0	0	0.0051
0	1	1	0.9335
1	0	1	0.9481
1	1	0	0.005

"weighted Gaussian" pairs,

$$G \equiv \{(g_1(\mathbf{x}), w_1), (g_2(\mathbf{x}), w_2), \ldots, (g_m(\mathbf{x}), w_m)\}, \tag{2.2.14}$$

where each g_i is a distinct Gaussian pdf, and each w_i is an integer value that designates the weight that g_i has in the model. The GMM pdf function is calculated as

$$p(\mathbf{x}) = \sum_{i=1}^{m} \hat{w}_i \; g_i(\mathbf{x}), \tag{2.2.15}$$

where each function $g_i(\mathbf{x})$ is a multivariate Gaussian distribution:

$$g_i(\mathbf{x}) = f_{\mathbf{X}}^{(i)}(\mathbf{x}) \sim \mathcal{N}(\mu_i, \Sigma_i), \tag{2.2.16}$$

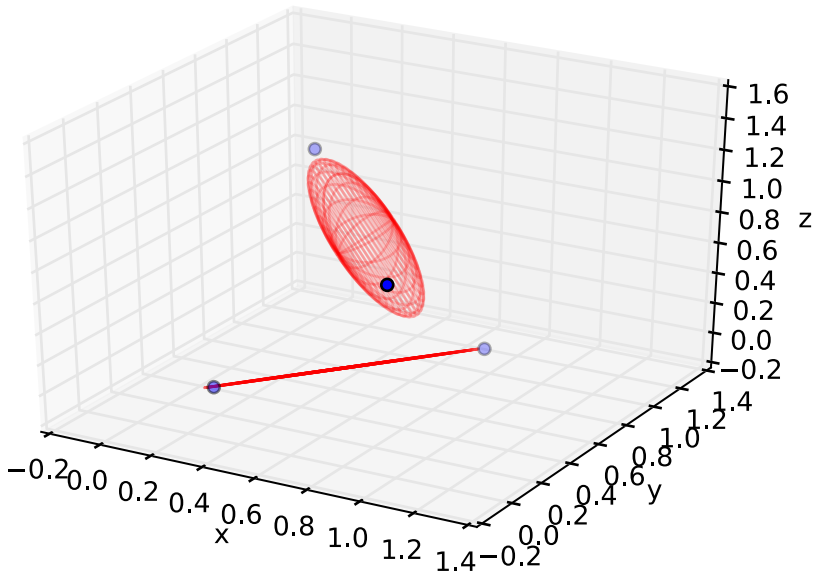

Figure 2.6: Modelling the XOR relationship using a 2-component mixture of 3-dimensional Gaussian pdfs. The smaller blue points represent the function $z = x \oplus y$, and the larger red ellipsoids (one is very narrow) represent the 1-σ equiprobability surface of each Gaussian.

and its corresponding *normalized* weighting factor is

$$\hat{w}_i = w_i \; / \; \sum_{k=1}^{m} w_k. \tag{2.2.17}$$

To properly estimate the density of a process using a GMM, it is necessary to make use of some kind of fitting algorithm that will provide good estimates of all of the GMM's parameters. One fairly popular fitting algorithm is known as Expectation Maximization. In Chapter 3, an alternative fitting algorithm for estimating GMM parameters, *Dynamic Gaussian Mixture Estimation* (DGME), is presented, and is a key contribution of this dissertation.

2.2.2.5 Expectation Maximization

The Expectation Maximization (EM) algorithm was first introduced by Dempster, Laird, and Rubin (1977). The EM technique is a general concept that can be applied in several different contexts. The technique involves two steps, not surprisingly called the *Expectation* step, and the *Maximization* step. The idea is that some fixed set of observation data $\mathbf{x}$ is available, and a chosen[6] family of probability distributions (or mixture of distributions), which is parameterized using a set of unknown parameters $\boldsymbol{\theta}$. The desired result of the EM algorithm is to determine an optimal[7] parameter vector, $\boldsymbol{\theta}^*$, which will cause the probability distribution to best explain the observation data. In order to determine this optimal parameter vector, an initial (probably non-optimal) parameter vector $\boldsymbol{\theta}^{(0)}$ is first chosen. Using this parameter vector, an equation is constructed that represents the **expected** likelihood of the data given some arbitrary set of parameter values $\hat{\boldsymbol{\theta}}$:

$$f(x, \theta) = E_Z[L(x \mid \theta)] = \int p(z|\theta)L(x|z, \theta)\, dz \tag{2.2.18}$$

The expectation is over a probability distribution that may include latent variables Z, and that may depend on a particular set of parameter values. The previously chosen parameter values will be used in this pdf if they play a role in it.

After the expected value function has been determined, it is **maximized** with respect to θ: a new θ value is found that maximizes the expected likelihoods determined by the previous 'best' θ value.

2.3 Why use Gaussian Mixture Models?

Depending upon the task at hand, GMMs may be very suitable for representing a model and producing inferences from this model. The reasons for this are threefold: representational power, locally distinct contributions to the density, and closed-form

[6] The selection of this probability distribution can be completely arbitrary, though it is clearly preferable to choose a distribution family which one believes will be able to adequately represent the data.

[7] The EM algorithm will not escape local optima, so the final parameter values which the EM algorithm converges to may only be locally optimal.

computations (i. e. no sampling required).

Despite the basis function for a GMM being the Gaussian (i. e. Normal) pdf, GMMs are flexible in the kinds of pdfs they can represent or approximate. As the number of components increases towards infinity, a GMM is capable of representing *any* arbitrary probability density surface (Dasgupta and Raftery, 1998; Fraley and Raftery, 1998).

In practice, however, the fit depends on the kind of distribution one wishes to approximate. For example, the Weibull distribution is a pdf commonly used for modelling failure rates. It's mathematical form is

$$f(x) = \begin{cases} \frac{a}{\lambda}\left(\frac{x}{\lambda}\right)^{a-1} \exp\{-(x/\lambda)^a\} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.3.1}$$

where a is the *shape* parameter, and λ is the *scale* parameter.

In Figure 2.7, a 7-component GMM is used to approximate a Weibull distribution where $a = 1.5$ and $\lambda = 1$. The GMM follows the general trend of the Weibull distribution, but suffers from having a "wavy" shape which is not as smooth as the approximated function.

Furthermore, the Gaussian function is a *local* function, in that its values rapidly approach zero as the point it is evaluated at moves further from the mean. This property means that when Gaussians are used as mixture components, each mixture component becomes 'responsible' for a given neighborhood of the pdf's support. Because of this, it is possible to model systems which have extremely different behavior under different conditions, so long as these conditions are treated as random variables that are part of the model.

But one might still ask, "What is so special about the Gaussian function? Couldn't one use, for example, a Poisson distribution?" The answer is that the Gaussian pdf belongs to a family of distributions known to be *self-conjugate* distributions, which have the unique property that when a conditional probability is calculated, the resulting conditional (i. e. posterior) pdf also has the same form (in this case, Gaussian) as the unconditional (i. e. prior) distribution, only with different parameters. This also holds true for marginalization, and as a result, when determining a conditional pdf, numerical integration can be avoided, reducing the overall computational effort required. Instead of

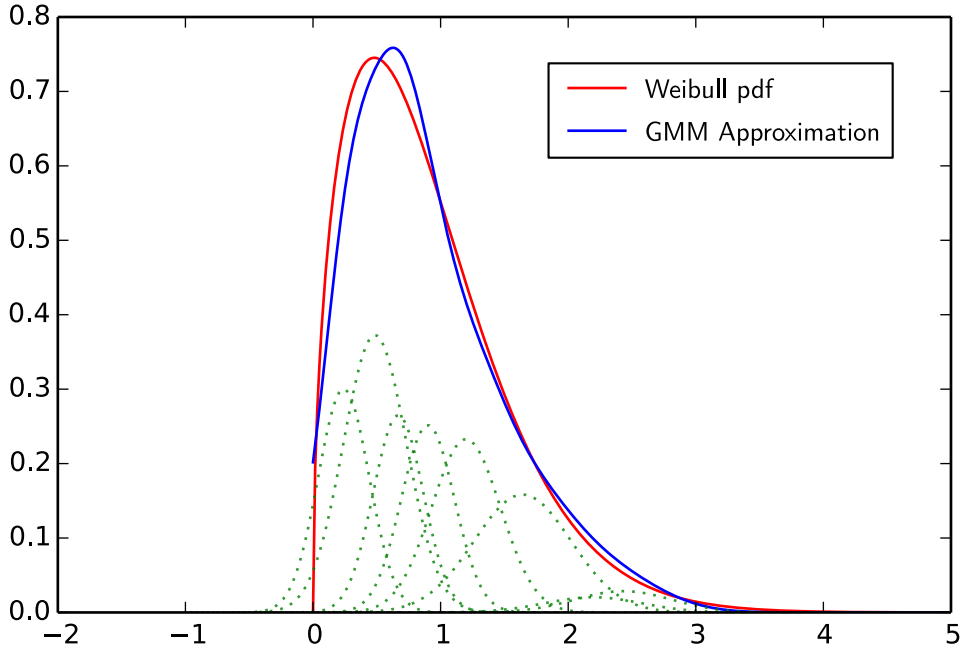

Figure 2.7: An example in which a GMM has approximated a Weibull distribution with a=1.5, based on 10000 randomly drawn samples. Individual Gaussian components, scaled by their weights, are drawn as dotted lines.

needing to use monte-carlo based numerical integration in order to calculate the posterior with Bayes theorem as

$$\overbrace{p(y \mid x)}^{\text{posterior}} = \frac{\overbrace{p(x \mid y)}^{\text{likelihood}}\overbrace{p(y)}^{\text{prior}}}{\underbrace{\int p(x \mid y)p(y)\,dy}_{\text{evidence}}}, \tag{2.3.2}$$

the posterior pdf can be calculated in one step using closed-form equations for the posterior pdf parameters in terms of the parameters of the likelihood, prior, and evidence (see (4.2.2) for an example of this).

Chapter 3

Dynamic Gaussian Mixture Estimation

This chapter introduces one of the main contributions of this thesis: a density-estimation technique named *Dynamic Gaussian Mixture Estimation* (DGME). First, a motivation for the technique will be provided, after which DGME will be formally presented in terms of its operation, and an explanation of the different parameters that affect the way it behaves when building a model. Finally, the method's performance will be demonstrated by comparing it to the performance of other density estimation methods.

3.1 Challenges with Existing Density Estimation Methods

When modelling a process' density from a very large number of observations, the density estimation methods discussed in Section 2.2 require either a large amount of storage (in the case of KDE), or a large amount of processing (in the case of the EM). Furthermore, both methods require the a priori selection of parameters, requiring knowledge of the process being modelled (KDE requires the selection of a bandwidth parameter, and EM requires the selection of the number of Gaussian components in the model, as well as regularization parameters).

The difficulty in requiring these a priori parameters is that one of the goals

of density estimation involves *finding out* what number of components would be appropriate to model a particular process, which precludes knowing this information in advance. One way to handle this problem is to run a method several times with varying numbers of components, and based on the results to select an optimal number of components that minimizes some goodness-of-fit cost function. Unfortunately, running a method several times requires even more processing time. It would be preferable to have a method that can provide reasonable parameter estimates without extensive storage and/or processing requirements.

3.2 Main Concept Behind DGME

The density estimation algorithm presented in this dissertation, Dynamic Gaussian Mixture Estimation (DGME), seeks to reduce the need for significant amounts of prior knowledge about a process being modelled, and to be efficient in terms of the amount of processing required for the algorithm to run and the amount of storage required to model the observed data.

One of the fundamental assumptions made by the DGME algorithm is that observations are not stored in memory, and must be assimilated into a model at the time they are observed. Traditional batch algorithms like *Expectation Maximization* are unable to function under this assumption, and require that all the observed data be cached.

At its core, the basic design behind of the DGME algorithm is elegant and intuitive. As each new observation is encountered, an assessment is made as to whether it fits well into the existing model or not. This assessment serves as the basis for making the ***do-merge decision***: if the observation does fit well into the existing model, then the parameters of one or more of the existing components of the model are adjusted to better explain the observation. If it does not fit well, then a new component is added to the model to account for the observation. Mathematically speaking, we can define a *likelihood function* $L_i(\mathbf{x})$ for the i^{th} Gaussian component of the current model which measures how well an observation fits into this component. By additionally defining a *likelihood threshold*, L_{th}, we can write the *do-merge decision* in terms of these likelihoods as

The *do-merge decision* criterion

$$\underbrace{\max_i L_i(\mathbf{x})}_{L_{max}} \geq L_{th}, \tag{3.2.1}$$

in which we've defined L_{max} as the maximum value of $L_i(\mathbf{x})$ for any i.

This general idea is summarized in Algorithm 1. The main algorithm is comprised of four important subroutines, whose behaviors affect the overall performance of the method when applied in a given scenario. The `GET_LIKELIHOOD_THRESHOLD` subroutine and the `GET_MAX_PENALIZED_LIKELIHOOD` subroutine compute L_{th} and L_{max}, respectively, and these values are compared as in (3.2.1) to decide whether merging the observations is required. After this decision has been made, either the `MERGE_OBS` or the `ADD_NEW_COMPONENT` subroutine will be executed in order to update the model with the observation. Each of these supporting subroutines is described in more detail in Section 3.8.

Algorithm 1: Pseudocode for Overall DGME Algorithm

input : a single observation vector $\mathbf{x}$
initial covariance magnitude σ_0^2
maximum covariance magnitude σ_{max}^2
Mahalanobis radius r_M

Result: Parameters of model are updated so that it estimates the density of $\mathbf{x}$ and all previously observed data.

```
Function DGME_UPDATE_FROM_OBS(x, σ0², σmax², rM)
    if x is first observation for model then
        ADD_NEW_COMPONENT(x, σ0² I) ;                    // I is an identity matrix
        nf ← the number of features in x;
        Lth ← GET_LIKELIHOOD_THRESHOLD(nf, σmax²);
    else
        // get the maximum likelihood and the index i of the component having
           this maximum.
        i, Lmax ← GET_MAX_PENALIZED_LIKELIHOOD(x, σmax², rM);
        if Lmax ≥ Lth then
            MERGE_OBS(i, x);
        else
            ADD_NEW_COMPONENT(x, σ0² I);
        end
    end
end
```

3.3 Distance-based Likelihood Functions

We will now consider some specific forms that the `GET_LIKELIHOOD_THRESHOLD` and `GET_MAX_PENALIZED_LIKELIHOOD` subroutines can take on. We will begin by considering what a sensible merging behavior defined by the *do-merge decision* criterion should look like. One of the dangers of a density-estimation method that can repeatedly merge new observations into existing mixture components is that the size of the neighborhood over which a component is active can become extremely large. In this case, the benefits of having a mixture of *local* components are diminished. Therefore, it is desirable to place some kind of guarantee that each component has "jurisdiction" over a limited region of the support space. In other words, we would like to place an upper bound on the extent to which a component's "shape"[1] can stretch in any given direction.

The question we will now consider is, *how can we define* L_{th} *and* $L_i(\mathbf{x})$ *in order to limit a component's "shape"?* To work towards an answer to this question, we will first explore what happens when we define the *do-merge decision* criterion so that new observations are merged when they lie within a fixed Mahalanobis distance[2] from at least one of the existing model components. Such a *do-merge decision* can be defined mathematically in terms of L_{th} and $L_i(\mathbf{x})$ as:

The *do-merge decision* likelihood functions for merging when an observation is within a Mahalanobis distance of r_M

$$L_{th} = -r_M \quad \text{and} \tag{3.3.1}$$

$$L_i(\mathbf{x}) = -\sqrt{(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)} \tag{3.3.2}$$

Figure 3.1 provides a graphical explanation of this criterion. Shown in the figure is a 2D two-component model, where each ellipse represents an equiprobability contour of a component. The points along each ellipse have a Mahalanobis distance of 1 from the

[1] When a component's "shape" is mentioned in this dissertation, or its "size" growing or shrinking, this is an informal way of referring to the shape of a 1-σ equiprobability ellipse associated with the component's covariance matrix. Likewise, any time a covariance's "shape" or "size" is mentioned in this dissertation, it is referring to the shape or size of the 1-σ equiprobability ellipse.

[2] The Mahalanobis distance of a point $\mathbf{x}$ with respect to a Gaussian component is defined as $\sqrt{(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$, which can also be written as $\sqrt{-2 \ln p(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}$

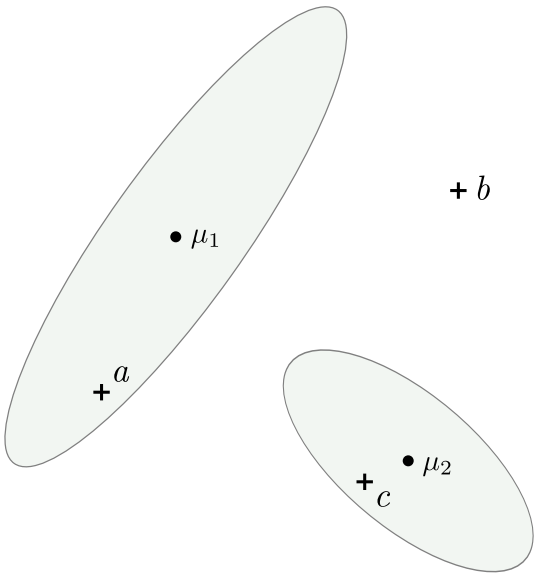

Figure 3.1: Depiction of the *do-merge decision* criterion defined by (3.3.1) and (3.3.2) when $r_M = 1$. Each ellipse represents an equiprobability contour of a 2D GMM component, where the points along the ellipse have a Mahalanobis distance of 1. If an observation occurs at a or c, it will be merged into components 1 or 2, respectively. If an observation occurs at b, a new component will be added to the model.

corresponding component's mean. If we let $r_M = 1$ in (3.3.1), then the model-update action taken will differ depending on which region of the figure an observation point is contained in. The point labeled a would be merged into the component with mean μ_1, while the point labeled b would cause a new component to be created in the model, and the point labeled c would be merged into the component with mean μ_2. In summary, any point within a Mahalanobis distance of r_M from a mean μ_i will be merged into component i, whereas if a point is not within a Mahalanobis distance of r_M from any of the component means, a new component will be created that represents the point.

The Mahalanobis-distance based likelihood function defined in (3.3.2) is intuitive, ensuring that $L_j(\mathbf{x}) > L_k(\mathbf{x})$ when component j explains the observation $\mathbf{x}$ better than component k does. Though intuitive, a Mahalanobis-based likelihood also has some issues that we must handle. The first problem is that though the intent is to provide a fixed limit to the shape of each component, the limit will *decrease* as the number of merged observations increases, resulting in potentially more components than desired in the model.

The second problem is that without some kind of regularization, $\mathbf{\Sigma}$ can become singular, such that when $(\mathbf{x} - \boldsymbol{\mu})$ is in the null-space of $\mathbf{\Sigma}$, the likelihood $L_i(\mathbf{x})$ becomes $-\infty$. This makes it possible for observations that are truly "near"[3] a component's mean to appear "far" from it, and to not end up being merged into it when they should. Both of these problems will be addressed in the following subsections.

3.4 Effect of Observation Quantity on the Merge Boundary

Though the *do-merge decision* criterion defined in (3.3.1) and (3.3.2) correctly implements the merge behavior described in Figure 3.1, this does *not* guarantee that the upper-bound on how far a covariance matrix is allowed to "stretch" will remain constant. In other words, even if observation values which are as "far" away from a component as possible (given the *do-merge decision* with $r_M = 1$) are merged again and again into the component, the maximum "stretching" of the component will not remain constant. As an example, if we start with a single univariate component having $\mu = 0$ and $\sigma = 1$, and repeatedly choose a maximal observation[4] which still qualifies to be merged under this *do-merge decision* criterion, then the σ values decrease as the number of observations merged (N) increases. In other words, when $r_M = 1$ in (3.3.1), the tight upper-bound on the stretch will inevitably decrease as N increases. This behavior is shown in Figure 3.2.

Though σ decreases as N increases in Figure 3.2 (where $r_M = 1$), it is possible to influence this trend by modifying the value of r_M in (3.3.1). The longer-term σ trend resulting from r_M values of 0.9, 1, and 1.1 is shown in Figure 3.3. Note how initially σ will decrease regardless of the value of r_M. For increasing N values where $N > 5$, however, the σ values for $r_M = 1$ begin to flatten out, while for $r_M > 1$, σ increases, and for $r_M < 1$, σ decreases. These trends are explored in more detail in Figure 3.4. The fact that σ decreases initially, independent of the value of r_M or the initial σ value, guarantees that each component will occupy a restricted neighborhood of the observation space, and not spread uncontrollably. Because of this, the GMMs which result when applying DGME

[3] Here, "nearness" is measured in terms of the Mahalanobis distance that an observation has with respect to a Gaussian component.

[4] In these analyses, every "maximal observation" that is chosen lies in the same direction from the component's mean.

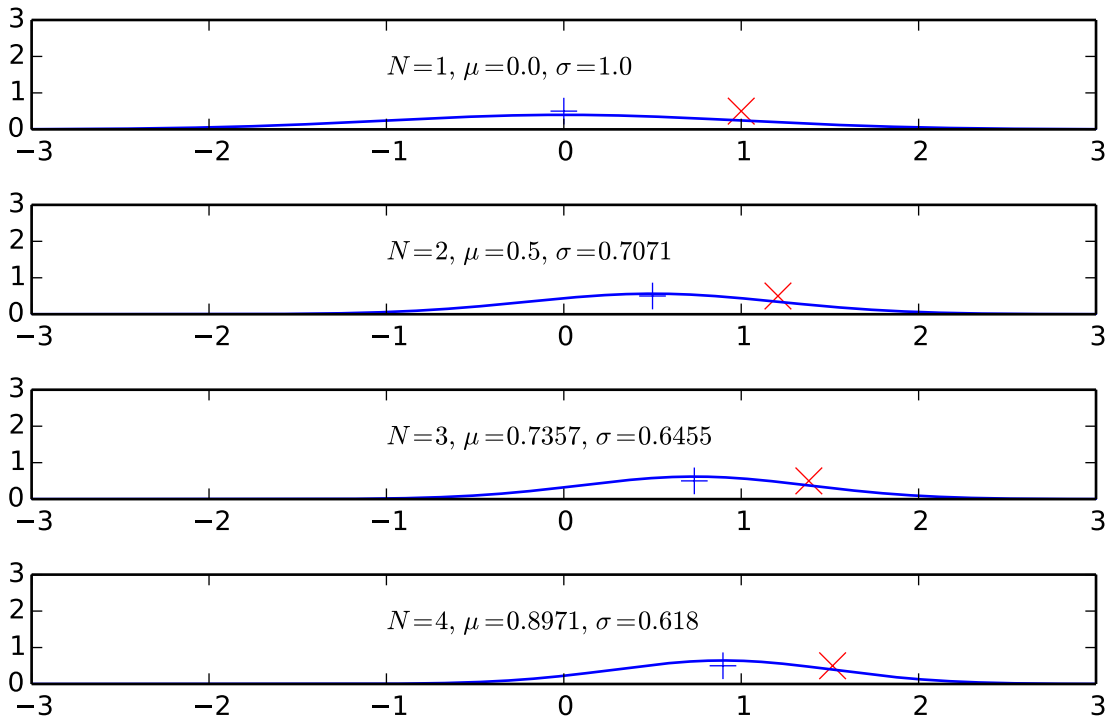

Figure 3.2: Result of repeatedly merging observations into a 1D Gaussian pdf with a mean marked with the blue $+$. Each new observation x_{next} (shown as a red $\times$) is chosen as a maximal value that doesn't exceed the *do-merge decision* threshold. Note that the quantity σ decreases as N, the number of observations represented by the model, increases. The *do-merge decision* criterion used here is $-\sqrt{(x-\mu)^2/\sigma^2} > -1$.

end up with a smaller amount of bias than would otherwise occur when one tries to model nonlinear processes with "more global" linear models.

3.4.1 Extension to Multidimensional Components

The results for the univariate case shown in Figures 3.3 and 3.4 can also be extended to the case when we have multivariate Gaussian components. To see how they apply, we must consider the relationship of the multidimensional Mahalanobis distance and the unidimensional Mahalanobis distances for each dimension. It is possible to view any arbitrary covariance matrix $\mathbf{\Sigma}$ as a rotated version of a diagonal covariance matrix $\mathbf{\Lambda}$ (e. g. by diagonalizing $\mathbf{\Sigma}$ so that $\mathbf{\Sigma} = \mathbf{U\Lambda U}^{-1}$ where $\mathbf{U}$ is a unitary matrix containing

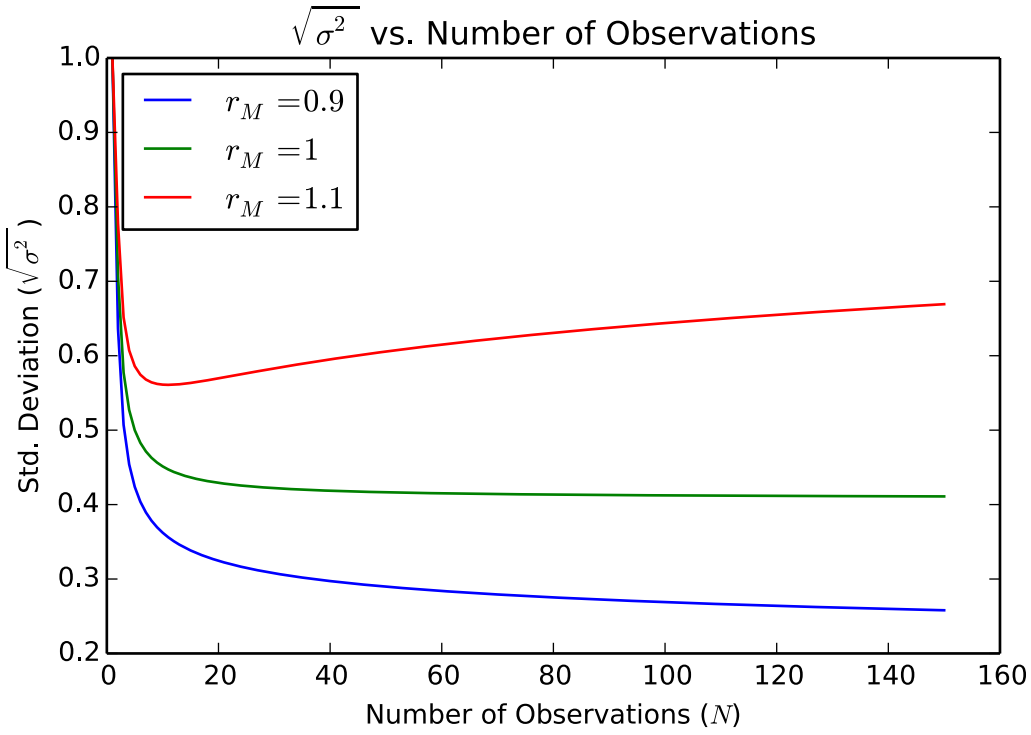

Figure 3.3: The standard-deviation ($\sqrt{\sigma^2}$) of a single 1D Gaussian pdf vs. the number of "maximal" observations (N) that have been incorporated into the Gaussian's parameter estimates. Each new observation is a maximal value that doesn't exceed the *do-merge decision* threshold. As the value of r_M used in (3.3.1) increases with respect to $r_M = 1$, so does the slope of the line approached by the $\sqrt{\sigma^2}$ vs. N curve. Likewise, when $r_M < 1$, the asymptotic slope decreases.

column eigenvectors of $\mathbf{\Sigma}$). Considering this, then any results related to the effect that merging "maximal" observations into a diagonal-covariance mixture component has on the component over time can be applied to a component with an arbitrary covariance matrix.

If we constrain our multidimensional component to have a diagonal covariance matrix,

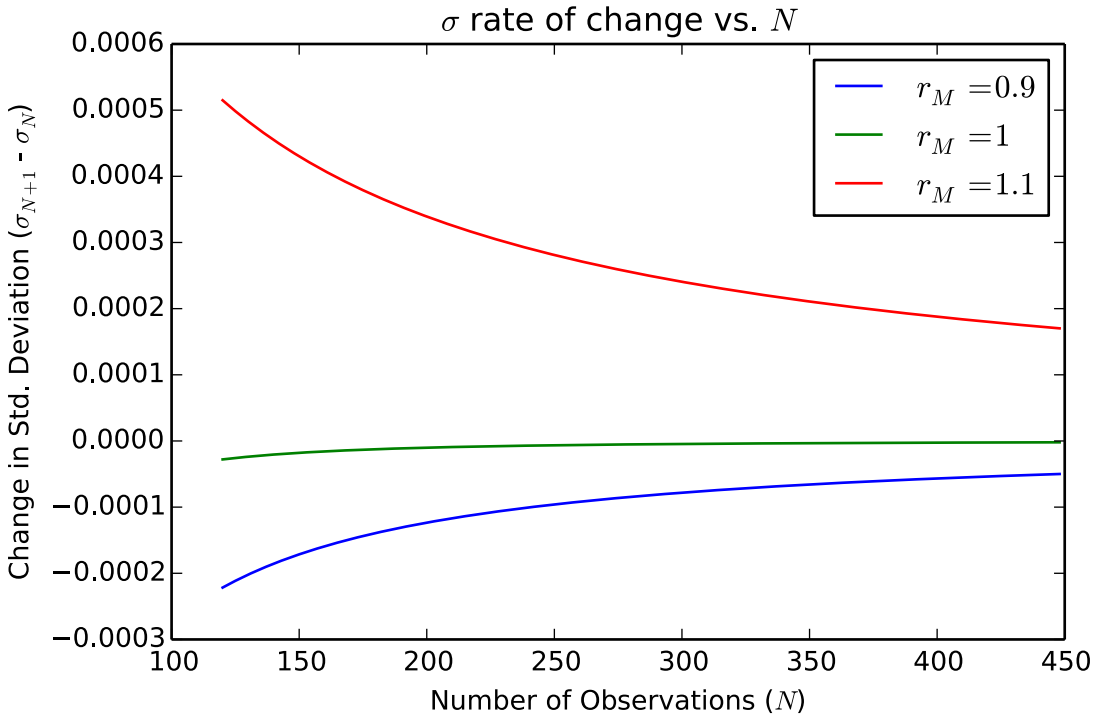

Figure 3.4: The rate-of-change of the standard-deviation (σ) values plotted in Figure 3.3 (note that a higher range of N values are plotted here). The asymptotic limit of σ is reached most quickly when $r_M = 1$.

we can write the *squared* Mahalanobis distance to a point $\mathbf{x}$ as

$$\begin{aligned} d_M^2 &= (\boldsymbol{\mu} - \mathbf{x})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \mathbf{x}) \\ &= \frac{(\mu_1 - x_1)^2}{\sigma_1^2} + \frac{(\mu_2 - x_2)^2}{\sigma_2^2} + \cdots + \frac{(\mu_N - x_N)^2}{\sigma_N^2} \\ &= d_{M,1}^2 + d_{M,2}^2 + \cdots + d_{M,N}^2. \end{aligned} \tag{3.4.1}$$

where $d_{M,i}^2$ is the squared Mahalanobis distance of the i^{th} element of $\mathbf{x}$ with respect to a pdf where all dimensions besides the i^{th} have been marginalized out of the full multidimensional pdf. In other words, the squared *multidimensional* Mahalanobis distance is the sum of the squared *unidimensional* Mahalanobis distances for each individual dimension.

Let us assume that the 1^{st} vector component has the largest individual covariance (i. e. $\sigma_1^2 \geq \sigma_i^2 : i \neq 1$). In this case, an $\mathbf{x}$ vector that is considered maximal (i. e. that

causes $d_M = d_{M,thresh}$) will have the form $(x_1, 0, \cdots, 0)$. When all $x_i : i \neq 1$ are set to 0, then based on (3.4.1), $d_M = d_{M,1}$. Therefore, choosing x_1 so that it satisfies the constraint $d_{M,1} = d_{M,thresh}$ also satisfies the constraint $d_M = d_{M,thresh}$. Because of this, we can directly use the results in Figures 3.3 and 3.4 to understand what happens to the shape of a multidimensional component as new observations are merged into it – for a particular dimension, its shape will be affected in the same way, being dependent on the value r_M (i. e. $d_{M,thresh}$) that is chosen.

3.5 Likelihood Function Singularities and Regularization

One issue with defining the likelihood as in (3.3.2) has to do with the fact that $\mathbf{\Sigma}_i$ will at some point be estimated from a very small sample size, so that the observations in the sample all lie in an affine subset of the full observation space. In this case, it is possible for the equiprobability ellipse of $\mathbf{\Sigma}_i$ to become very small (i. e. collapsing to a plane, line, or point having a lower dimension than the dimension of the observation space). When this happens, if there is a new observation $\mathbf{x}$ for which $\mathbf{x} - \boldsymbol{\mu}_i$ doesn't lie in the subspace spanned by the row or column vectors of $\mathbf{\Sigma}_i$, then the quantity $(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)$ will become infinite. For example, if a component's covariance is estimated on the basis of only two observations, $\mathbf{x}_1$ and $\mathbf{x}_2$, then the likelihood of this component would be $-\infty$ for all new observations except for those lying on the line collinear with $\mathbf{x}_1 - \mathbf{x}_2$.

The way that such singular covariance matrices are avoided in DGME is to use a *regularized* estimate of the covariance when calculating $L_i(\mathbf{x})$ in (3.3.2). Instead of naively using the $\mathbf{\Sigma}$ matrix, which is estimated using the standard sample covariance formula, we construct a regularized matrix, where we prevent any eigenvalues from becoming 0. To do so, we first perform an eigendecomposition of $\mathbf{\Sigma}$:

$$\mathbf{\Sigma} = \mathbf{Q} \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix} \mathbf{Q}^T$$

where $\mathbf{Q}$ is a matrix in which the columns are comprised of orthonormal eigenvectors,

and λ_k are the corresponding eigenvalues. The regularized matrix, $\mathbf{\Sigma}_R$, is then defined as

$$\mathbf{\Sigma}_R = \mathbf{Q} \begin{bmatrix} \lambda_{R,1} & & 0 \\ & \ddots & \\ 0 & & \lambda_{R,N} \end{bmatrix} \mathbf{Q}^T \tag{3.5.1}$$

where

$$\lambda_{R,k} = \max(\lambda_k, \lambda_{R,min}). \tag{3.5.2}$$

Here, $\lambda_{R,min}$ is a lower-bound on the eigenvalues of $\mathbf{\Sigma}_R$ which can be chosen as desired in order to adjust the amount of regularization. Typically a value should be chosen based on the expected inter-modal distance,[5] d_{im}, of the process being modelled. In particular, a good rule of thumb is for $\sqrt{\lambda_{R,min}}$ to be at least an order of magnitude smaller than d_{im}.

3.6 Population vs. Sample Covariance

Since DGME builds a pdf based on a sequence of observations, a clear assumption is that we don't already have a complete set of observations that constitutes the entire *population* of observations. As a result, it is appropriate for us to use the unbiased *sample* estimate of covariance when we estimate a component's $\mathbf{\Sigma}$ parameter. In other words, when estimating the covariance of component i, we use

$$\mathbf{\Sigma}_i = \frac{1}{N-1} \sum_{k=1}^{N} (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T. \tag{3.6.1}$$

While use of this estimator makes sense when we want to have an unbiased estimate of the density, we must consider whether it is appropriate to use as part of the *do-merge decision* criterion. In this case, the mean of the Gaussian component is regarded as the exact center of some neighborhood that defines the threshold of whether a new observation is allowed to be merged or not. If we use the population covariance, then the corresponding population standard-deviation will be the average of the distance from the

[5] The inter-modal distance is is the smallest distance that one expects to exist between the modes of the estimated density function, based on whatever process is being modelled.

mean to each point in the set of observations that the component represents. Assuming we have points arranged equidistant from each other on a circle, then the covariance will be spherical, where each diagonal entry of Σ is σ^2 and σ (i. e. the population standard-deviation) will be the radius of that circle. If we let $r_M = 1$ in (3.3.1), then this radius becomes the threshold for determining whether a new observation will get merged or not. The problem with this is that it assumes the first few observations are a perfect representation of the population. Because this is a bad assumption, it is important to initially increase the threshold radius to allow for more data (which may end up farther than a population standard-deviation away from the sample mean) to be gathered and a better estimate of the mean to be found. As more data are collected, it makes sense for this radius to approach the population standard-deviation. This is precisely how the sample covariance estimator behaves: initially it "inflates" the variance to account for uncertainty in the data, and as more data is collected (i. e. more observations are encountered), this uncertainty decreases, and the sample standard-deviation becomes closer to the population standard-deviation.

The conclusion we can draw from this is that it is appropriate to use the sample covariance estimator both for estimating the parameters of a Gaussian component and for computing the Mahalanobis distance used as part of the the *do-merge decision* . Although the difference between the sample and population estimates becomes negligible as N_i (the number of observations used to estimate the parameters of component i) increases, the N_i value for every newly-created mixture component will be small initially. It is therefore important to use the sample covariance estimator for the *do-merge decision* in order to maintain unbiased mixture component parameter estimates when N_i is small.

3.7 Updating the Model with an Observation

In Algorithm 1, if the *do-merge decision* is decided *yes*, then the new observation must be merged into an existing component of the model. We must therefore determine the best way to merge an observation into a model. A measure of optimality used in DGME pertaining to observation merging is the following: assuming that an observation $\mathbf{x}_{new}$ was merged into a GMM component G_a, then the likelihood of all observations represented by G_a should be maximized given G_a as the local model for these observations. In other words, if $\mathbf{x}_{new}$ is to be merged into G_a, and S_{new} is the set of observations that G_a

represents after the merge, then the updated parameters of G_a are given as:

$$(\boldsymbol{\mu}_{new}, \boldsymbol{\Sigma}_{new}) = \underset{(\boldsymbol{\mu},\boldsymbol{\Sigma})}{\operatorname{argmax}} \sum_{\mathbf{x}\in S_{new}} g(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.7.1}$$

where g is a normally distributed pdf with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, and

$$S_{new} = S_{old} \cup \{\mathbf{x}_{new}\}. \tag{3.7.2}$$

This is the maximum likelihood criterion, and the corresponding mean and covariance update equations are derived in Sections 3.7.1 and 3.7.2.

3.7.1 Merging Two Gaussian Components into One

Since merging a single observation into an existing Gaussian can be seen as a specialized case of merging two Gaussians together, we will first derive the equations relevant to merging two Gaussian components together, and subsequently consider the case where we are merging only a single observation.

Assume that we begin with two one-dimensional Gaussian components, g_1 and g_2 which were estimated, respectively, from the sets of points $\{a_1, \cdots, a_N\}$ and $\{b_1, \cdots, b_M\}$ using an unbiased estimator. In other words, these components are distributed as follows:

$$g_1 \sim \mathcal{N}\left(\underbrace{\frac{1}{N}\sum_{i=1}^{N} a_i}_{\mu_1}, \underbrace{\frac{1}{N-1}\sum_{i=1}^{N}(\mu_1 - a_i)^2}_{\sigma_1^2}\right) \tag{3.7.3}$$

$$g_2 \sim \mathcal{N}\left(\underbrace{\frac{1}{M}\sum_{i=1}^{M} b_i}_{\mu_2}, \underbrace{\frac{1}{M-1}\sum_{i=1}^{M}(\mu_2 - b_i)^2}_{\sigma_2^2}\right) \tag{3.7.4}$$

If we were to have estimated a single component g_3 from the set of points $\{a_1, \cdots, a_N, b_1, \cdots, b_M\}$, such a component would be distributed as follows:

$$g_3 = g_1 \curlyvee g_2 \sim \mathcal{N}\left(\underbrace{\frac{1}{N+M}(\sum_{i=1}^{N} a_i + \sum_{i=1}^{M} b_i)}_{\mu_3}, \underbrace{\frac{1}{N+M-1}(\sum_{i=1}^{N}(\mu_3 - a_i)^2 + \sum_{i=1}^{M}(\mu_3 - b_i)^2)}_{\sigma_3^2}\right) \tag{3.7.5}$$

Here, the $\curlyvee$ operator signifies the *merging* of two components into a single component.

Our objective is to write the mean and variance parameters of (3.7.5) in terms of N, M, and the parameters in (3.7.3) and (3.7.4). By inspection, and with a bit of algebraic manipulation, the merged mean parameter can be written as:

Two-Gaussian Merge Mean Equation

$$\mu_3 = \frac{1}{N+M}(N\mu_1 + M\mu_2) \tag{3.7.6}$$

To determine the variance parameter, we will first consider how to write the expression $(\mu_3 - a_i)^2$ in terms of other parameters. First, we will rewrite this expression as:

$$(\mu_3 - a_i)^2 = (\mu_1 - a_i + \underbrace{\Delta_{13}}_{\mu_3 - \mu_1})^2 = (\mu_1 - a_i)^2 + \Delta_{13}^2 + 2\Delta_{13}(\mu_1 - a_i). \tag{3.7.7}$$

Now we can write:

$$\sum_{i=1}^{N}(\mu_3 - a_i)^2 = \sum_{i=1}^{N}(\mu_1 - a_i)^2 + \sum_{i=1}^{N}\Delta_{13}^2 + 2\Delta_{13}(\sum_{i=1}^{N}\mu_1 - \sum_{i=1}^{N}a_i) \tag{3.7.8}$$

$$= (N-1)\sigma_1^2 + N\Delta_{13}^2 + 2\Delta_{13}(\cancelto{0}{N\mu_1 - N\mu_1}) \tag{3.7.9}$$

$$= (N-1)\sigma_1^2 + N(\mu_3 - \mu_1)^2 \tag{3.7.10}$$

Finally, this result (and the analogous result for $\sum_{i=1}^{M}(\mu_3 - b_i)^2$) can be substituted

into

$$\sigma_3^2 = \frac{1}{N+M-1}(\sum_{i=1}^{N}(\mu_3 - a_i)^2 + \sum_{i=1}^{M}(\mu_3 - b_i)^2), \tag{3.7.11}$$

giving us the final result we are after:

Two-Gaussian Merge Covariance Equation (1-D)

$$\sigma_3^2 = \frac{1}{N+M-1}\left\{(N-1)\sigma_1^2 + N(\mu_3 - \mu_1)^2 + (M-1)\sigma_2^2 + M(\mu_3 - \mu_2)^2\right\} \tag{3.7.12}$$

The above results were derived for the case of merging two 1D Gaussian components. These results can be easily extended to the case of merging two N-dimensional Gaussian components. The result in (3.7.6) does not change at all in this case – the only difference is that the means will be N-dimensional vectors instead of scalars. The result in (3.7.12), on the other hand, will have a similar form, but will change slightly for the multidimensional case – squared scalars will be replaced with an outer-product, and the variances are replaced with covariance matrices. The resulting equation for the multidimensional case is:

Two-Gaussian Merge Covariance Equation (multidimensional)

$$\boldsymbol{\Sigma}_3 = \frac{1}{N+M-1}\Big\{(N-1)\boldsymbol{\Sigma}_1 + N(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_1)^T + (M-1)\boldsymbol{\Sigma}_2 + M(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_2)^T\Big\} \tag{3.7.13}$$

3.7.2 Merging a single point into a component

In the case where we have one existing Gaussian component, and would like to merge one point into this Gaussian, we can derive the parameter-update equations based on (3.7.6)

and (3.7.12). The key is to treat the single point as a Gaussian component estimated on the basis of this one point. In other words, if we represent the point as x, then

$$\mu_2 = x \tag{3.7.14}$$

and

$$M = 1. \tag{3.7.15}$$

If we substitute (3.7.14) and (3.7.15) into (3.7.6) and (3.7.12), after some algebraic manipulation[6], we end up with the following equations for the new component that results from the merge:

Point to Gaussian Merge Equations

$$\mu_3 = \frac{1}{N+1}(N\mu_1 + x) \tag{3.7.16}$$

and

$$\sigma_3^2 = \frac{(\mu_1 - x)^2}{N+1} + \frac{N-1}{N}\sigma_1^2. \tag{3.7.17}$$

or, for the multidimensional case,

$$\mathbf{\Sigma}_3 = \frac{(\boldsymbol{\mu}_1 - \mathbf{x})(\boldsymbol{\mu}_1 - \mathbf{x})^T}{N+1} + \frac{N-1}{N}\mathbf{\Sigma}_1. \tag{3.7.18}$$

One important observation is that when $M = 1$, σ_2^2 (or $\mathbf{\Sigma}_2$ in the multidimensional case) does not show up in these equations. This means that for the Gaussian component we use to represent the single point, *it doesn't matter what we choose for the covariance.*

3.8 Summary of DGME Algorithm

Bringing together the previously discussed issues and concepts, let us now summarize the designs of the DGME algorithm and the proposed likelihood function. The "big-picture" version of the DGME algorithm was presented in Section 3.2 and Algorithm 1. The

[6] See Appendix A for the derivation of the point-to-gaussian merge equations.

Algorithm 2: Pseudocode for `ADD_NEW_COMPONENT`

input : mean vector $\boldsymbol{\mu}$
initial covariance matrix $\boldsymbol{\Sigma}_0$
Result: a new mixture-component is added to model with mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}_0$, and weight 1.

Algorithm 3: Pseudocode for `MERGE_OBS`

input : index n indicating which component is to be updated
observation vector $\mathbf{x}$
Result: the parameters of component n are updated to incorporate $\mathbf{x}$.
Function `MERGE_OBS`(n, $\mathbf{x}$)
  $\boldsymbol{\mu} \leftarrow$ mean of mixture component n;
  $\boldsymbol{\Sigma} \leftarrow$ covariance matrix of mixture component n;
  $N \leftarrow$ the number of observations used to estimate $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$;
  $N_{new} \leftarrow N + 1$;
  $\boldsymbol{\mu}_{new} \leftarrow (N\boldsymbol{\mu} + \mathbf{x})/(N_{new})$;
  $\boldsymbol{\Sigma}_{new} \leftarrow (\boldsymbol{\mu} - \mathbf{x})(\boldsymbol{\mu} - \mathbf{x})^T/(N_{new}) + (N-1)\boldsymbol{\Sigma}/N$;
  replace mixture component n parameters with $\boldsymbol{\mu}_{new}$, $\boldsymbol{\Sigma}_{new}$, and N_{new};
end

Algorithm 4: Pseudocode for `GET_MAX_PENALIZED_LIKELIHOOD`

input : a single observation vector $\mathbf{x}$
output : i – the index of the component providing the maximum likelihood
L_{max} – the maximum likelihood value found
Function `GET_MAX_PENALIZED_LIKELIHOOD`($\mathbf{x}$)
  likelihoods $\leftarrow$ [`GET_PENALIZED_LIKELIHOOD`(n, $\mathbf{x}$) | $n \in [1, \cdots$, number of components]];
  $L_{max} \leftarrow$ max(likelihoods);
  $i \leftarrow$ index of likelihoods where L_{max} appears
end

`ADD_NEW_COMPONENT` subroutine (Algorithm 2) simply adds a new mixture component to the model having a weight 1, and the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ parameters specified as inputs. Because the value for L_{th} is always compared with the individual $L_i(\mathbf{x})$ values for each component when evaluating the *do-merge decision* , it is possible to divide both (3.3.1) and (3.3.2) by r_M. This allows us to define the `GET_LIKELIHOOD_THRESHOLD` function so that it always returns -1. It could, however, be defined so that it depends on the current state of the model.

The `MERGE_OBS` subroutine (Algorithm 3) takes an observation $\mathbf{x}$ as an input, as well

Algorithm 5: Pseudocode for `GET_PENALIZED_LIKELIHOOD`

input : n – an index specifying a particular component within model
$\mathbf{x}$ – a single observation vector

Data:
α_q, N_q, $N_{inflection}$ – parameters that influence the computation of α
ϵ – fractional amount by which covariance should be inflated
r_M – Mahalanobis radius

output : L – the likelihood of $\mathbf{x}$ given component n of model

Function `GET_PENALIZED_LIKELIHOOD(`n, $\mathbf{x}$`)`

```
μ ← mean of mixture component n;
Σ ← covariance matrix of mixture component n;
N_obs ← the number of observations used to estimate μ and Σ;
Σ_reg ← REGULARIZE(Σ);                    // regularizing Σ is optional
if N_obs is 1 then
    Σ_W ← σ²_max I;
else
    α ← GET_ALPHA(N_obs, N_inflection, N_q, α_q);
    Σ_W ← α(1 + ε)Σ_reg + (1 − α)σ²_max I;
end
return −MAHALANOBIS_DIST(x, μ, Σ_W) / r_M
```

end

Algorithm 6: Pseudocode for `GET_ALPHA`

input : N_{obs} – the number of observations currently represented by a particular mixture component
N_q, α_q – An α value of α_q will be computed when N_{obs} is N_q
$N_{inflection}$ – the inflection of the sigmoid curve will occur when N_{obs} is $N_{inflection}$.

Result: Computed value for α

Function `GET_ALPHA(`N_{obs}, $N_{inflection}$, N_q, α_q`)`

$$z \leftarrow [\ln \alpha_q - \ln (1 - \alpha_q)] \,/\, [N_q - N_{inflection}] \;;$$
$$\alpha \leftarrow 1/\left[1 + \exp\{-z(N_{obs} - N_{inflection})\}\right] \;;$$

end

as the index i of a particular component g_i of the model, and $\mathbf{x}$ is used to update the parameters of g_i according to equations in (3.7.16) and (3.7.18). The last subroutine referred to in Algorithm 1 is `GET_MAX_PENALIZED_LIKELIHOOD`, whose pseudocode is shown in Algorithm 4. This subroutine calls `GET_PENALIZED_LIKELIHOOD` for each component in the model, and then determines which of these components results in the largest likelihood value, and what the largest likelihood value is. The real work of this

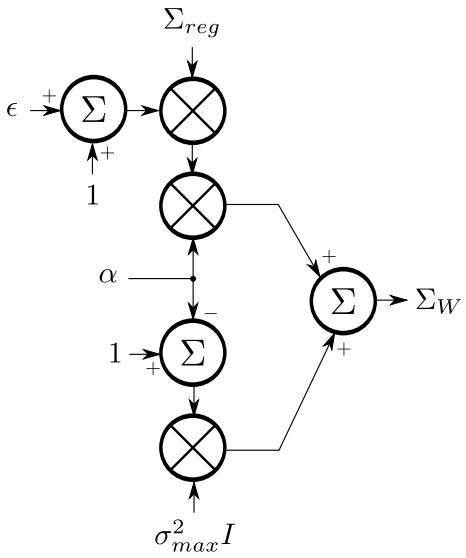

Figure 3.5: Schematic diagram depiction of how the GET_PENALIZED_LIKELIHOOD function computes the working covariance. The $\otimes$ symbols represent a multiplication operation.

subroutine is performed inside the `GET_PENALIZED_LIKELIHOOD` function, which will now be explained in detail.

3.8.1 The `GET_PENALIZED_LIKELIHOOD` Function

The `GET_PENALIZED_LIKELIHOOD` function is responsible for computing a value L_i for the i^{th} mixture component, which is compared to the threshold L_{th} in order to determine whether an observation should get merged into an existing mixture component. Furthermore, L_i will be computed for every i in the model (i. e. $i \in$ [1..num_components]). If an observation will be merged at all, it will be merged to component j, where $j = \text{argmax}_i\, L_i$. Therefore, `GET_PENALIZED_LIKELIHOOD` must be defined in such a way that the largest L_i value is associated with the component into which it would be best to merge the observation.

Algorithm 5 details the procedure for computing the likelihood, and Figure 3.5 shows an alternative way of visualizing how this procedure operates. The overall goal is to compute a Mahalanobis distance from an observation to a mean, given a particular

covariance matrix. The covariance used for this distance computation is called the *working covariance*, and is given the symbol $\mathbf{\Sigma}_W$. The working covariance is not the same as the actual sample covariance, $\mathbf{\Sigma}_{samp}$, of the mixture component. Rather, the working covariance is computed based on several model hyperparameters, as well as $\mathbf{\Sigma}_{samp}$, and the number of observations, N_{obs}, that have already been incorporated into the mixture component we're considering.

If the component for which L_i is being computed has only a single observation that has contributed to its parameters, then $\mathbf{\Sigma}_{samp}$ will not reflect anything other than a prior assumption about how the data is spread. Because of this, the working covariance computation completely ignores $\mathbf{\Sigma}_{samp}$ in this case, and will set $\mathbf{\Sigma}_W$ to be the spherical covariance $\mathbf{\Sigma}_{max} = \sigma^2_{max} I$. If, on the other hand, there are at least two observations that have contributed to the mixture component's parameters, then the working covariance is computed as a convex sum of the two covariances $(1+\epsilon)\mathbf{\Sigma}_{reg}$ and $\mathbf{\Sigma}_{max}$. The first of these covariances is an "inflated" version of $\mathbf{\Sigma}_{reg}$, where the amount of inflation depends on the value of ϵ, which can be chosen freely. A further parameter, α, is a value between 0 and 1. It defines the fraction of $(1+\epsilon)\mathbf{\Sigma}_{reg}$ that will be used to compute $\mathbf{\Sigma}_W$. When $\alpha = 1$, the working covariance is identical to $(1+\epsilon)\mathbf{\Sigma}_{reg}$, and when $\alpha = 0$, then the working covariance is identical to $\mathbf{\Sigma}_{max}$. If $\alpha = 0.5$, then $\mathbf{\Sigma}_W$ is essentially the "average" of the two covariances.

The parameter α is not chosen arbitrarily, but rather is computed based on the current value of N_{obs} for a component (i. e. its unnormalized weight). The computation makes use of a sigmoid function, as shown in Figure 3.6. The inflection point of this sigmoid function (i. e. where the output is 0.5) is set to occur at the point where $N_{obs} = N_{inflection}$. Also, when $N_{obs} = N_q$, the function will output the value α_q. The parameters $N_{inflection}$, N_q, and $\alpha_q \in [0, 1]$ can be freely chosen in order to fully specify the sigmoid function. In summary, the covariance mixing factor α is computed as $(\alpha = \text{SIGMOID}(N_{obs}; N_{inflection}, N_q, \alpha_q))$. The specific details of how this function is implemented can be found in Algorithm 6.

The reason this function was chosen in order to compute α is because initially, right after a component has been created, it doesn't have a sufficient number of observations that have contributed to its parameters in order to allow one to "trust" the accuracy of the parameters. In this case, instead of allowing a very narrow covariance estimate to influence the likelihood computation, we would prefer to use a broad prior covariance that reflects the uncertainty of our estimate. As more and more observations are incorporated

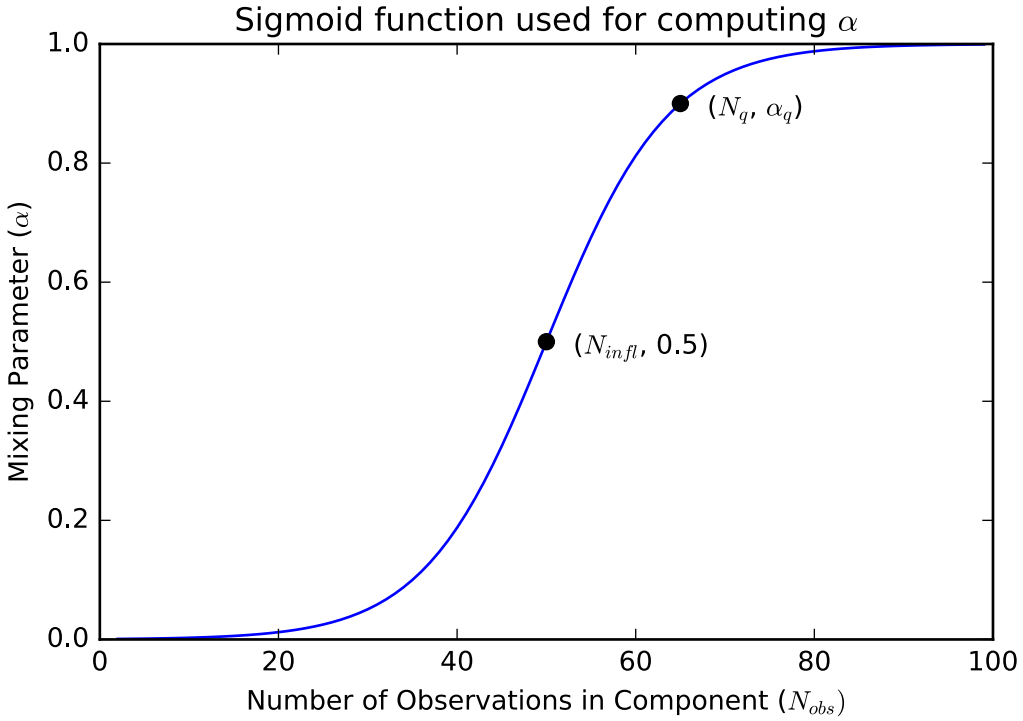

Figure 3.6: This function is an example within the sigmoid family of curves which can be used to compute values of α. In this particular example, $N_{inflection} = 50$, $N_q = 65$, and $\alpha_q = 0.9$.

into the component's parameters, we can rely on its parameters more, and slowly shift the covariance we use for the likelihood computation from a broad (i. e. spherical) prior to something with a more defined / refined shape.

3.9 Experimental Results

3.9.1 Estimating a Known Density Function

In order to examine the effectiveness of DGME in estimating densities of points, an experiment was conducted in which 200 points were randomly sampled from a known mixture distribution with two equally-weighted components: $g_1 \sim \mathcal{N}(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}2.5 & 1.5\\1.5 & 2.5\end{smallmatrix}\right])$ and $g_2 \sim \mathcal{N}(\left[\begin{smallmatrix}7\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}2.5 & -1.5\\-1.5 & 2.5\end{smallmatrix}\right])$. From each component, 100 points were sampled, and these

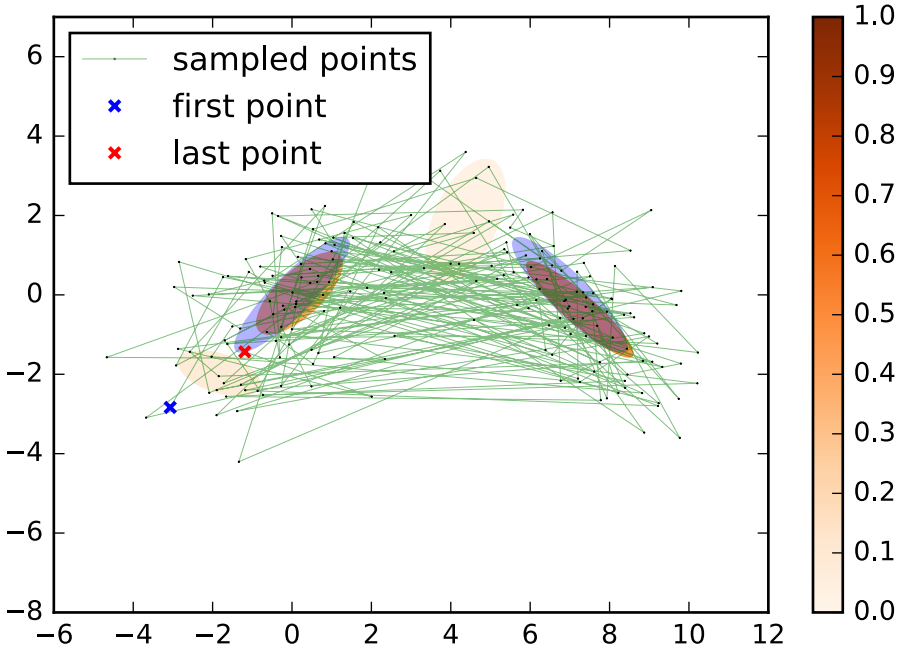

Figure 3.7: Result of applying DGME to observations sampled from a known distribution. The two purple ellipses represent the mixture components from which the data points were drawn. The orange-shaded ellipses represent the components in the model learned by DGME, and are colored according to their normalized weights. The unnormalized component weights, from left-to-right are 17, 81, 6 and 96.

points were randomly shuffled together. An example of the results one obtains when running DGME[7] on this data is shown in Figure 3.7. The figure shows all of the sampled data-points with green lines connecting temporally neighboring points. The light-blue ellipses represent g_1 and g_2, and the other ellipses represent the components of the GMM which resulted from providing DGME with the data. The shade of orange used for these components represents the normalized weight of each component. In this figure, as one would expect, the two darkly shaded ellipses have the highest weights, and are correspond well with the components of the source distribution. The other lesser-weighted components represent points that can be considered outliers with respect to the source distribution, and these components have a correspondingly lesser influence

[7]The DGME parameters used here were $\sigma^2_{max} = 2.6^2$, $\alpha_q = 0.9$, $N_{infl} = 50$, $N_q = 100$, $r_M = 4$, $\epsilon = 0.1$, and $\sigma^2_{min} = 0.001$.

in the estimated GMM.

3.9.2 DGME Comparison with Expectation-Maximization

In order to test the quality of the DGME method, it was compared with an Expectation-Maximization (EM) based approach in its ability to model the "Old Faithful" benchmark dataset. The EM algorithm was used to estimate the parameters of a 2-Gaussian GMM in an offline manner (i. e. using all observations at each iteration). Just as it was necessary to select in advance the number of Gaussians for the EM approach, DGME parameters were chosen which resulted in density estimates of this dataset that most often contain two Gaussians.

The resulting density estimates are shown in Figure 3.8. The upper-left and lower-right plots show the resulting density estimate from the EM and online DGME methods, respectively. Though barely perceptible from these plots, the difference between the plots is shown in the heat map in the background. These results demonstrate that the DGME method can produce density estimates comparable to the EM method. It should be noted that because the results of the DGME online method depend on the stochastic nature of the data, the estimates it produces can vary depending on the order the data is encountered. To get a better idea of the method's general performance, it was run until 100 density estimates containing 2 Gaussians had been produced. The Mean Integrated Square Error (MISE) between each of these 100 density estimates and the EM generated density estimate were calculated, resulting in an average MISE of 0.0358 with a standard deviation of 0.0317. It is clear from these results that the density estimates generated by these two methods are very similar, though the computational and storage requirements are quite different, since the DGME method requires only a single-pass and does not need to store the observations.

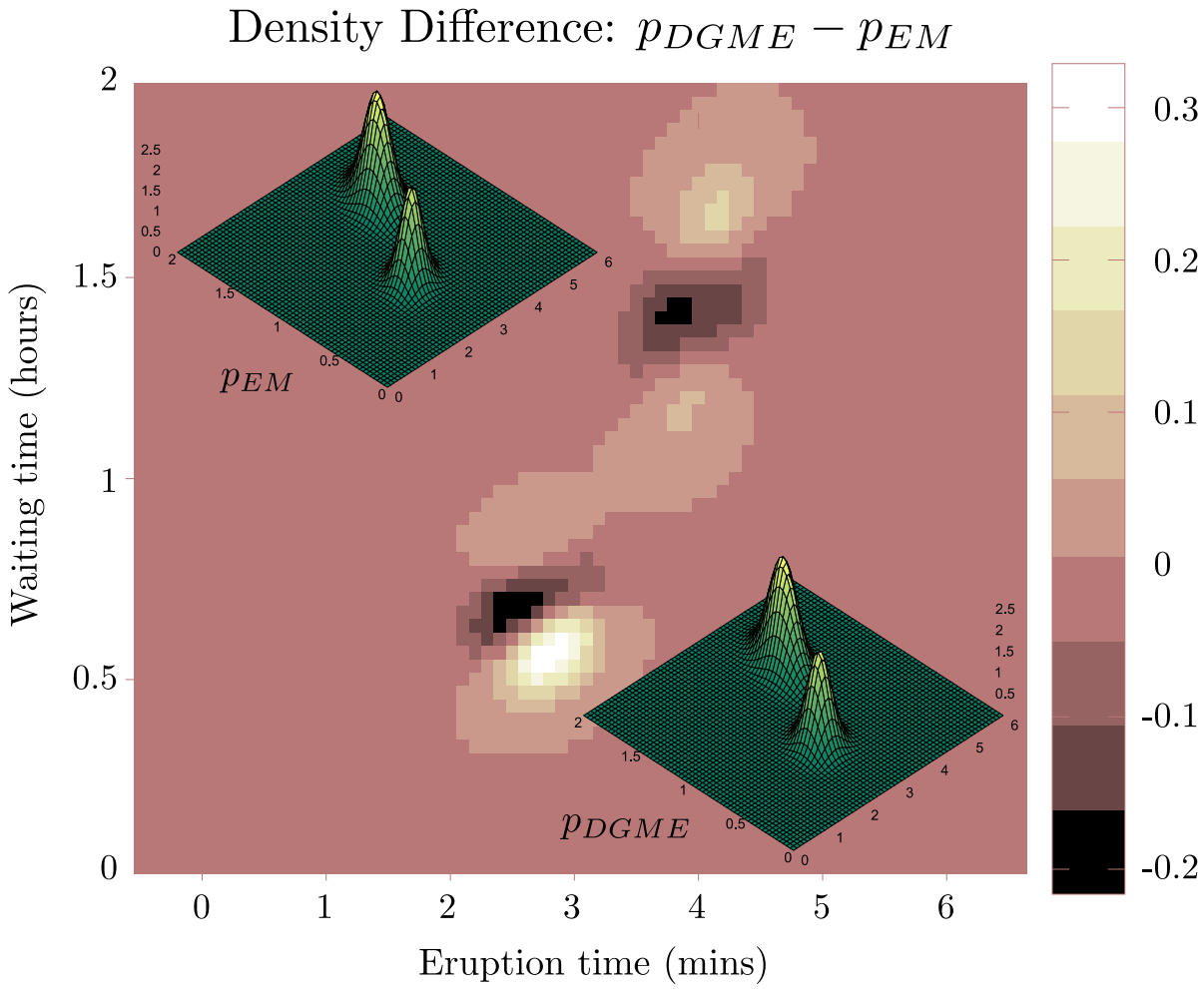

Figure 3.8: Comparison of density estimate p_{EM} derived using the EM algorithm (upper left), and the density estimate p_{DGME} derived using the DGME method (lower right), on the Old Faithful dataset. The heat map in the background shows the difference between p_{DGME} and p_{EM}.

Chapter 4

Prediction and Classification

Prediction and classification are two important tasks encountered within the field of machine learning. The two tasks are related in that they both start with a set of training examples (each typically in the form of an *input vector* and a *target vector*). From this set, a model is estimated that can be used to predict either a target value or a target class given a particular input vector. An example of a regression task would be: *given the age of a person in a particular field, how much money does the person earn on average.* A classification task might be: *given 10 medical diagnostic measurements taken on a patient, decide whether the patient has disease A, disease B, or is healthy.* It is possible to perform either kind of task (prediction or classification) using one of the most common statistical tools for process modelling, *regression.*

A regression model is typically defined in the form of a function which maps from a particular input vector to a target vector that is expected given the input vector. This provides exactly the information needed for the prediction task – in this case, a regression function would map from a vector of given values to a vector of predicted values. For classification problems, on the other hand, the application is not as direct – a classification problem needs to choose one of k classes given a vector of given values. In this case, it is possible to code the target vector using what is known as a *one-hot encoding*, such that the problem can be treated as a regression problem. For the above example in which the classes are *patient has disease A*, *patient has disease B*, and *patient is healthy*, we could one-hot encode these different target classes as the vectors $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 1 & 0\end{bmatrix}$, and $\begin{bmatrix}0 & 0 & 1\end{bmatrix}$, respectively. Then in order to choose a particular class, each element of the predicted vector's value is compared, and the class is chosen whose corresponding

element-location has the highest value. For example, if a predicted target vector was $\begin{bmatrix}0.4 & 0.3 & 0.8\end{bmatrix}$, we could say that the class represented is *patient is healthy*. The clarity / certainty of this choice depends on the relative difference between the values of the target vector elements. If, for example, the two highest values are very near to each other, it might be decided that it's not clear which of the corresponding classes should be chosen. In this sense, the normalized predicted target vector can be viewed as containing the probability values for each class.

This chapter explains how regression can be performed with joint probability densities (and more specifically, Gaussian Mixture Models) in order to make predictions about certain components[1] of an observation vector given specific values for other components. As mentioned above, regression can also be used for classification tasks. Examples showing the performance of DGME and regression for prediction and classification tasks will also be presented in this chapter.

4.1 Regression with Joint Probability Distributions

Regression methods are used in a wide variety of scenarios, including methods ranging from basic linear regression to more flexible variants such as Gaussian Process Regression (GPR). The common goal behind each of these methods is to learn an accurate and general mapping from some random input vector $\mathbf{X}$ to a random output vector $\mathbf{Y}$. To learn this mapping, $\mathbf{X} \mapsto \mathbf{Y}$, a regression method would operate on a set of training examples $(\mathbf{x}_i, \mathbf{y}_i)$, resulting with a regression function $\mathbf{y} = m(\mathbf{x})$. If one wants, however, to invert this function (assuming an inverse exists over some desired range of $\mathbf{y}$ values), and obtain the function $\mathbf{x} = m^{-1}(\mathbf{y})$, the regression method must typically be re-applied to a set of training examples in which $\mathbf{x}_i$ and $\mathbf{y}_i$ are swapped.

This illustrates one of the weaknesses of a regression-only approach to learning associations between variables: the learned relationship is unidirectional. A powerful technique for overcoming this limitation involves combining probability density estimation and regression.

Formally, a regression function is defined in probabilistic terms as the expected value of

[1] "Component" used here refers to an element in a vector, not a mixture component.

some random vector, given a specific value of a different random vector:

$$\mathbf{y} = m(\mathbf{x}) = E[\mathbf{Y} \mid \mathbf{X} = \mathbf{x}]. \tag{4.1.1}$$

If the representation of the joint probability density by which these random vectors are distributed is appropriately chosen, the calculation of this conditional expectation is relatively inexpensive, and can be a viable alternative to standard regression methods that directly estimate a regression function from training examples.

As a simple example, if we have two random variables, U and V, which we wish to know the relationship between, we can estimate the joint probability density $f_{\mathrm{U,V}}(u, v)$ by any number of density estimation techniques. In so doing, the relationship between these variables has been captured, and one can calculate either of the two possible regression functions

$$u = m(v) = E[\mathrm{U} \mid \mathrm{V} = v] = \int u' \, f_{\mathrm{U|V}}(u' \mid v) \, du' \quad \text{or} \tag{4.1.2}$$

$$v = m^{-1}(u) = E[\mathrm{V} \mid \mathrm{U} = u] = \int v' \, f_{\mathrm{V|U}}(v' \mid u) \, dv' \tag{4.1.3}$$

where the relationship between a conditional density function (such as the ones which appear inside the integrals) and a joint density function is given by Bayes' rule as

$$\underbrace{f_{\mathrm{Y|X}}(y \mid x)}_{\text{conditional pdf}} = \frac{\overbrace{f_{\mathrm{Y,X}}(y, x)}^{\text{joint pdf}}}{\underbrace{\int f_{\mathrm{Y,X}}(y, x) \, dy}_{\text{marginal pdf}}}. \tag{4.1.4}$$

Equations (4.1.2) and (4.1.3) are equally valid when U and V are random vectors.

4.2 Regression Using GMMs

In order to calculate a regression function from a GMM estimated by DGME, a method is used that was introduced as a least-squares function approximator in (Ghahramani and Jordan, 1994), and later presented in more detail as *Gaussian Mixture Regression* (GMR) in (Sung, 2004). It is a direct application of (4.1.1) to GMMs, taking advantage of the elegant mathematical properties of the Gaussian function. By using Gaussian functions as mixture components, the calculation of marginal and conditional probabilities required for the regression function becomes trivial, requiring little computational effort.

To properly understand how to use a GMM for regression, we will start by examining how a single Gaussian pdf can be used for regression.

4.2.1 Regression from a Single Multivariate Gaussian

It is possible view (4.1.4) as a partitioning of a multivariate Gaussian $f_{X,Y}$ where

$$f_{X,Y} = f_{Y|X}\, f_X. \tag{4.2.1}$$

One property of this partitioning is that the resulting components $f_{Y|X}$ and f_X are also both multivariate Gaussians. In the work done in (Mardia, Kent, and Bibby, 1979), the analytic solution for the conditional distribution of Gaussian distributed multivariate random-variables is derived. The result is presented here:

Assume we have a Gaussian distributed random vector $V = \begin{bmatrix} X_1 \parallel X_2 \end{bmatrix}$ composed of two 'subvectors' $X_1 \in \mathbb{R}^p$ and $X_2 \in \mathbb{R}^q$. If the variance of X_1 is Σ_{11}, and the variance of X_2 is Σ_{22}, then the variance of V can be written in the form of block submatrices as $\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$, where $\Sigma_{12} = \Sigma_{21}^T$ is the covariance between X_1 and X_2. Using these submatrices, and the means of X_1 and X_2, the conditional distribution of X_2 given X_1 can be written as

$$f_{X_2|X_1}(X_2 \mid X_1 = x_1) \sim \mathcal{N}(\underbrace{\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1)}_{E[X_2|X_1=x_1]},\ \Sigma_{X_2|X_1}) \tag{4.2.2}$$

where

$$\Sigma_{X_2|X_1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}. \tag{4.2.3}$$

The matrix $\Sigma_{X_2|X_1}$ is the *Schur complement* of Σ_{11}. In order to construct $\Sigma_{X_2|X_1}$, Σ_{11} must be inverted once, its unwanted rows (which correspond to the variables we're conditioning upon) must be removed, and then the resulting matrix must be inverted a second time. The matrix $\Sigma_{21}\Sigma_{11}^{-1}$ (part of the $E[X_2 \mid X_1 = x_1]$ expression) is known as the *regression coefficient* matrix, because it contains the coefficients which define the line onto which an input vector must be projected to get a corresponding expected output vector. Equation (4.2.2) indicates that if you want to condition X_2 on a particular value of X_1, the conditional mean is obtained by modifying X_2's mean by adding the product of Σ_{21} with a normalized version of how far X_1 is from its mean.

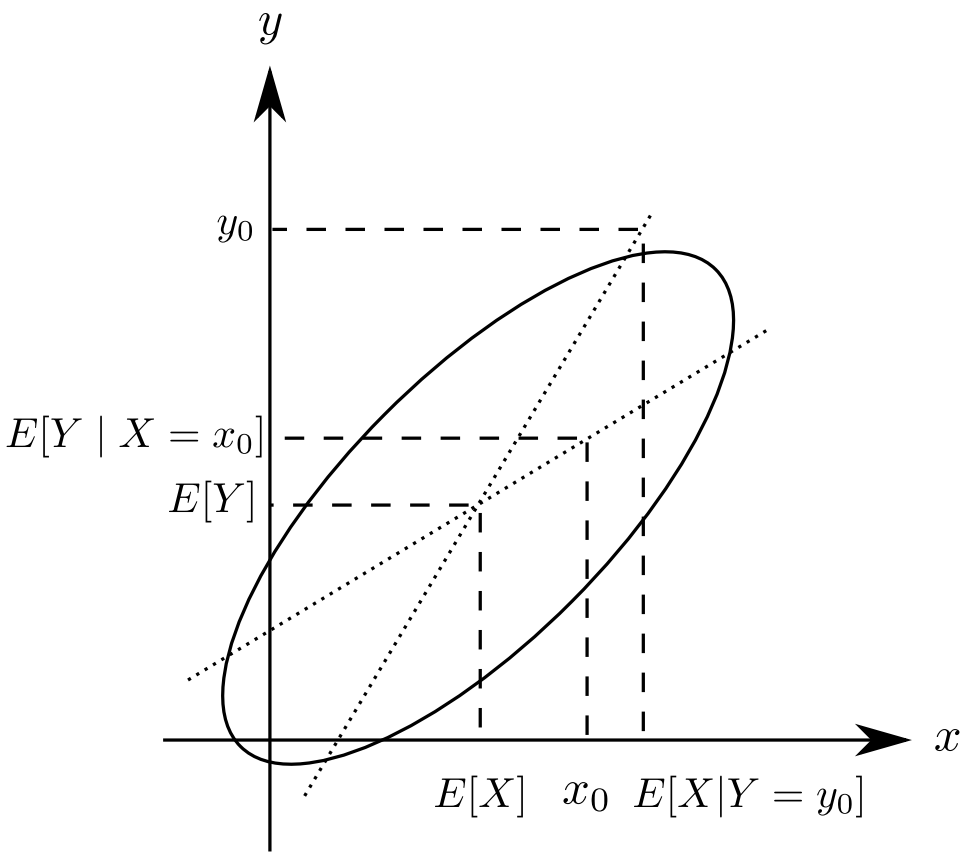

Figure 4.1: Illustration of using a 2D Gaussian pdf for regression: The joint pdf is projected onto the x-y plane in the form of an ellipse. Regression lines are shown for two cases: $E[Y|X = x_0]$ and $E[X|Y = y_0]$.

Based on the results given by (4.2.2) and (4.2.3), we can separate the joint density of a single multivariate Gaussian function according to (4.2.1):

Bayesian Decomposition of a Multivariate Normal Distribution

$$f_{X,Y} = f_{Y|X}\, f_X$$

where

$$f_{X,Y} \sim \mathcal{N}\left(\begin{bmatrix}\mu_X \parallel \mu_Y\end{bmatrix}, \begin{bmatrix}\Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY}\end{bmatrix}\right), \tag{4.2.4}$$

$$f_{Y|X} \sim \mathcal{N}(\mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \mu_X),\, \Sigma_{YY} - \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XY}), \tag{4.2.5}$$

and

$$f_X \sim \mathcal{N}(\mu_X,\, \Sigma_{XX}). \tag{4.2.6}$$

The result in (4.2.5) allows us to write a regression equation

$$E[Y \mid X = x] = \mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(x - \mu_X), \tag{4.2.7}$$

as long as we know the parameters of the joint pdf $f_{X,Y}$. This equation is depicted for the 2D case in Figure 4.1.

4.2.2 Generalizing to Arbitrary Components

The regression equation (4.2.7) applies to an observation vector composed of two concatenated subvectors (one regarded as the regressor and one as the regressand). This result is easily extended to the case where any arbitrarily-ordered subset of a vector's components can be used as the regressor random vector, and likewise any other ordered subset can be used as the regressand random vector.

For example, if each observation has the form $\mathbf{z_1} = \begin{bmatrix}A & B & C & D & F\end{bmatrix}$ where $A \ldots F$ are real scalar random variables, let us assume that $\mathbf{Z_1} \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ where

$$\boldsymbol{\mu}_1 = \begin{bmatrix}\mu_A & \mu_B & \mu_C & \mu_D & \mu_F\end{bmatrix},$$

and

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} \sigma_A^2 & \sigma_A\sigma_B & \sigma_A\sigma_C & \sigma_A\sigma_D & \sigma_A\sigma_F \\ \sigma_B\sigma_A & \sigma_B^2 & \sigma_B\sigma_C & \sigma_B\sigma_D & \sigma_B\sigma_F \\ \sigma_C\sigma_A & \sigma_C\sigma_B & \sigma_C^2 & \sigma_C\sigma_D & \sigma_C\sigma_F \\ \sigma_D\sigma_A & \sigma_D\sigma_B & \sigma_D\sigma_C & \sigma_D^2 & \sigma_D\sigma_F \\ \sigma_F\sigma_A & \sigma_F\sigma_B & \sigma_F\sigma_C & \sigma_F\sigma_D & \sigma_F^2 \end{bmatrix}.$$

If we wish to compute $\mathbb{E}[B, D \mid A = a, C = c]$, it is sufficient to marginalize F out of the pdf, and to re-order the component locations in $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$. Since marginalization of a multivariate Gaussian distribution simply requires removing the elements from $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$ that are related to the variables that are being marginalized out, we would start out with the marginal distribution $\mathbf{Z_2} \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ where

$$\boldsymbol{\mu}_2 = \begin{bmatrix} \mu_A & \mu_B & \mu_C & \mu_D \end{bmatrix},$$

and

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} \sigma_A^2 & \sigma_A\sigma_B & \sigma_A\sigma_C & \sigma_A\sigma_D \\ \sigma_B\sigma_A & \sigma_B^2 & \sigma_B\sigma_C & \sigma_B\sigma_D \\ \sigma_C\sigma_A & \sigma_C\sigma_B & \sigma_C^2 & \sigma_C\sigma_D \\ \sigma_D\sigma_A & \sigma_D\sigma_B & \sigma_D\sigma_C & \sigma_D^2 \end{bmatrix}.$$

Next, we would reorder the elements of $\mathbf{Z_2}$ to get $\mathbf{Z_3} \sim \mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ where

$$\boldsymbol{\mu}_3 = \begin{bmatrix} \underbrace{\mu_B \;\; \mu_D}_{\boldsymbol{\mu}_\mathbf{X}} & \underbrace{\mu_A \;\; \mu_C}_{\boldsymbol{\mu}_\mathbf{Y}} \end{bmatrix}, \tag{4.2.8}$$

and

$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} \overbrace{\begin{bmatrix} \sigma_B^2 & \sigma_B\sigma_D \\ \sigma_D\sigma_B & \sigma_D^2 \end{bmatrix}}^{\boldsymbol{\Sigma}_{XX}} & \begin{matrix} \sigma_B\sigma_A & \sigma_B\sigma_C \\ \sigma_D\sigma_A & \sigma_D\sigma_C \end{matrix} \\ \underbrace{\begin{bmatrix} \sigma_A\sigma_B & \sigma_A\sigma_D \\ \sigma_C\sigma_B & \sigma_C\sigma_D \end{bmatrix}}_{\boldsymbol{\Sigma}_{YX}} & \begin{matrix} \sigma_A^2 & \sigma_A\sigma_C \\ \sigma_C\sigma_A & \sigma_C^2 \end{matrix} \end{bmatrix}. \tag{4.2.9}$$

Finally, we can directly substitute the values identified in (4.2.8) and (4.2.9) into the

regression equation (4.2.7).

4.2.3 Gaussian Mixture Regression (GMR)

Besides being able to perform regression with a single Gaussian pdf, it is also possible to find a closed-form regression function expression for a pdf represented by a mixture model. Starting with the regression equation (4.1.1) which defines regression in terms of conditional expectation, the mixture-model specific regression formula can be derived as follows:

$$\begin{aligned}
m(x) &= E[Y \mid X = x] \\
&= \int y \, p_{Y|X}(Y = y \mid X = x)\, dy \\
&= \int y \, \frac{p_{YX}(Y = y, X = x)}{p_X(X = x)}\, dy \\
&= \frac{1}{p_X(X = x)} \int y \, p_{YX}(Y = y, X = x)\, dy \\
&= \frac{1}{p_X(X = x)} \int y \sum_i w_i \, p_{YX,i}(Y = y, X = x)\, dy \\
&= \sum_i \frac{1}{p_X(X = x)} \int y \, w_i \, p_{Y|X,i}(Y = y \mid X = x) \, p_{X,i}(X = x)\, dy \\
&= \sum_i \frac{w_i \, p_{X,i}(X = x)}{p_X(X = x)} \int y \, p_{Y|X,i}(Y = y \mid X = x)\, dy \\
&= \sum_i \frac{w_i \, p_{X,i}(X = x)}{p_X(X = x)} E_i[Y \mid X = x] \\
&= \sum_i \frac{w_i \, p_{X,i}(X = x)}{\sum_j w_j \, p_{X,j}(X = x)} E_i[Y \mid X = x].
\end{aligned} \tag{4.2.10}$$

One can treat the fractional multiplicand in the last line of (4.2.10) as a weight, in which $w_i \, p_{X,i}(X = x)$ represents the amount contributed to the total probability density $p_X(X = x)$ (evaluated at the specific value x). We can then write (4.2.10) as

$$m(x) = \sum_i k_i(x) \, E_i[Y \mid X = x], \tag{4.2.11}$$

with the per-mixture-component weights at x defined as

$$k_i(x) = \frac{w_i \; p_{X,i}(X = x)}{\sum_j w_j \; p_{X,j}(X = x)}. \tag{4.2.12}$$

Equation (4.2.11) is completely general, and applies to any kind of mixture model. In the case of *Gaussian* mixture models, the conditional distribution defined in (4.2.2) can be used for determining a single Gaussian component's conditional expectation in (4.2.11).

4.2.4 The Problem with Spherical Covariance

When a GMM contains "brand new" Gaussians created via the DGME algorithm, the covariance matrices of these Gaussians represent a spherical distribution, and can have a negative impact on the accuracy of values predicted with the model. To understand why this is, it is instructive to consider how regression is performed in the case of an individual multivariate Gaussian pdf. The slope of the regression function from (4.2.2) is $\mathbf{\Sigma_{YX}\Sigma_{XX}^{-1}}$, where $\mathbf{\Sigma_{YX}}$ and $\mathbf{\Sigma_{YX}}$ are block-matrices in (4.2.4). In the spherical case[2], $\mathbf{\Sigma_{YX}} = 0$, and the "slope" is such that the predicted value of Y has no dependence on a particular value of X. This makes sense and is reasonable *if the covariates of the data at* $\boldsymbol{\mu}$ *are statistically independent*. The problem, however, is that with "real" data, it is almost guaranteed that there will be some minuscule correlation between some of the covariates. This means that while a spherical covariance is a good choice in terms of representing a *lack of prior knowledge*, it is a poor choice to use this covariance as part of predictive / inferential operations (such as regression), because it can introduce potentially large errors.

As the number of observations used to estimate a Gaussian's $\mathbf{\Sigma}$ increases, the "shape" of $\mathbf{\Sigma}$ will become more and more aligned with the "shape" of the underlying distribution of local data around the Gaussian's mean. Such a covariance matrix will have much more "predictive power" than a spherical covariance. To explore the effect that the number of observations has on the "predictive power" of a Gaussian, an experiment was conducted in which N observations are sampled from a distribution with a known "optimal" principal component, $\mathbf{p}^*$. An eigendecomposition is performed on the resulting

[2]Covariance is spherical when the covariates are statistically independent.

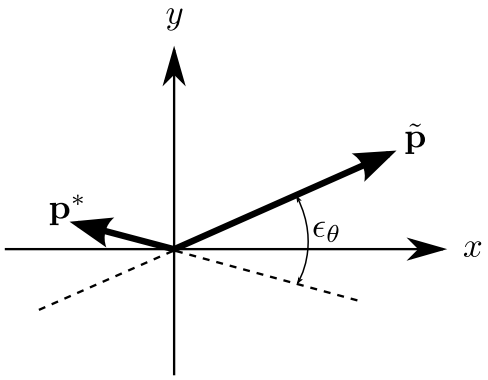

Figure 4.2: Acute Angular Error: for two vectors (e. g. $\tilde{\mathbf{p}}$ and $\mathbf{p}^*$), the acute angular error is defined as the acute angle between the lines through which the vectors pass. Here the error is labeled ϵ_θ. This definition applies also in higher-dimensions, where the dot-product can be used to determine the angle between two vectors.

covariance estimate, and the *acute angular error* between the first principal component, $\tilde{\mathbf{p}}$, is measured. The acute angular error ϵ_θ is defined as the absolute value of the acute angle between a line collinear with $\tilde{\mathbf{p}}$, and another line collinear with $\mathbf{p}^*$ – see Figure 4.2 for a geometric interpretation of this error.

For this experiment, estimates of $\boldsymbol{\Sigma}$ were made 1000 times, each time with the same sample size, N. For each $\boldsymbol{\Sigma}$ estimate, ϵ_θ was determined, and mean and standard-deviation statistics were computed for the ensemble of ϵ_θ values[3].

The ϵ_θ statistics for 2D data having values of N between 2 and 20 are shown in **??**. This figure shows that both the mean and standard deviation of ϵ_θ decrease rapidly with increasing N. The smaller ϵ_θ is, the "more correct" the covariance estimate is, and the more suitable it is for use in predictive models. If, for example, one wanted the 1-σ upper bound on the angular error to be 2°, then it would be necessary to exclude any Gaussians having an unnormalized weight (i. e. number of observations) $w < 4$.

Gaussians with $w = 1$ have spherical covariance matrices and an average ϵ_θ of 45°, which is completely unusable for good predictive results. *It is therefore essential to exclude Gaussians with $w = 1$ when predicting values, and often makes sense to exclude all Gaussians with $w < k$, for some $k > 1$.*

[3]The ϵ_θ values are normally distributed.

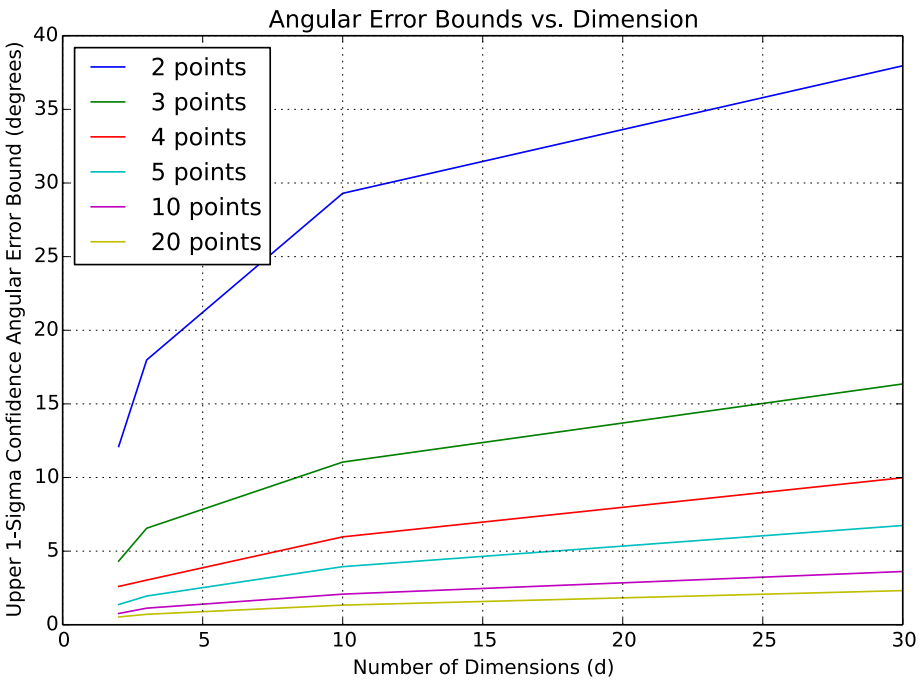

Figure 4.3: Effect of dimension on 1-σ upper bound of ϵ_θ: for a given number of points, the upper bound flattens with increasing dimension. The most important factor influencing the asymptotic upper bound is not the dimension of points (d), but rather the number of points (N). N's effect on ϵ_θ appears to have minimal dependence on the value of d.

The preceding analyses have so far only focused on 2D data. What happens when higher dimensional data is modeled? Figure 4.3 shows the 1-σ upper bound of ϵ_θ for different dimensions (d) and numbers of observations (N). As the dimension increases, the upper bound on ϵ_θ increases, but asymptotically approaches a limit. The value of this limit decreases dramatically after the first few points have been observed, leading one to a decent rule of thumb: *a Gaussian should have an unnormalized weight of at least* $w = 4$ *in order to get a "good" upper bound on* ϵ_θ .

Putting these findings into practice, we will compare the prediction errors of models under a variety of circumstances. Some models were trained using points sampled from the linear function $y = x$, and others were trained using sampled data from the quadratic function $y = 0.1x^2$. Models were trained from this data using DGME with different σ_0 values and different σ_{max} values. Predictions of y-values were made at the same values

used for the training data, and only those Gaussians having a weight $w > k$ were used for prediction, where $k \in \{1, 2, 3\}$. The results of these experiments are summarized in Table 4.1.

Table 4.1: Minimum Gaussian weight threshold (k) effect on mean-squared error (MSE) for linear and quadratic data. In both linear and quadratic cases, the MSE is reduced dramatically by omitting $w = 1$ Gaussians, but MSE can become worse for larger k if the distribution of weights is heavier for weights below k. The $w = n$ columns indicate the quantity of Gaussians having a weight $= n$ in the trained model.

Dataset Type	σ_0	σ_{max}	$w = 1$	$w = 2$	$w = 3$	$w > 3$	k	MSE	σ_{MSE}
linear	0.8	1.0	3	6	10	11	1	0.0174	0.0107
linear	0.8	1.0	3	6	10	11	2	6.312 x 10^{-16}	7.147 x 10^{-16}
linear	0.8	1.0	3	6	10	11	3	6.602 x 10^{-16}	7.510 x 10^{-16}
quadratic	0.6	1.2	2	36	1	5	1	0.0144	0.0139
quadratic	0.6	1.2	2	36	1	5	2	0.0026	0.0026
quadratic	0.6	1.2	2	36	1	5	3	0.1011	0.1565

4.3 Experimental Results

The DGME/GMR approach was tested in the areas of classification and prediction, using a standard benchmark in each of these areas. The two-spiral benchmark problem is a challenging classification task which requires a classifier to separate two classes of points that are not linearly separable. Mackey-Glass time-series prediction benchmark problem defines a chaotic time-series whose future values should be predicted based on past values, given some initial training data from the time-series.

4.3.1 Two-Spirals Benchmark

The Two-Spirals benchmark (initially presented in (Lang and Witbrock, 1988), and later in (Fahlman and Lebiere, 1990)) was chosen to investigate the performance of using DGME/GMR on classification problems. The dataset is by default[4] comprised of a series

[4]The Two Spirals dataset is parameterized by the *density* and *maxRadius* parameters, whose default values are 1 and 6.5, respectively.

of $\begin{bmatrix} x & y & C(x,y) \end{bmatrix}$ observations, where the class function $C(x,y) \in [0,1]$, and is defined as:

$$C(x,y) = \begin{cases} 0 & \text{when } x = -r_i \sin(\theta_i) \text{ and } y = -r_i \cos(\theta_i) \\ 1 & \text{when } x = r_i \sin(\theta_i) \text{ and } y = r_i \cos(\theta_i) \end{cases}$$

where

$$\theta_i = \frac{i\pi}{16},$$

$$r_i = (6.5)\frac{104 - i}{104},$$

and

$$i \in [0, 1, 2, \cdots, 96].$$

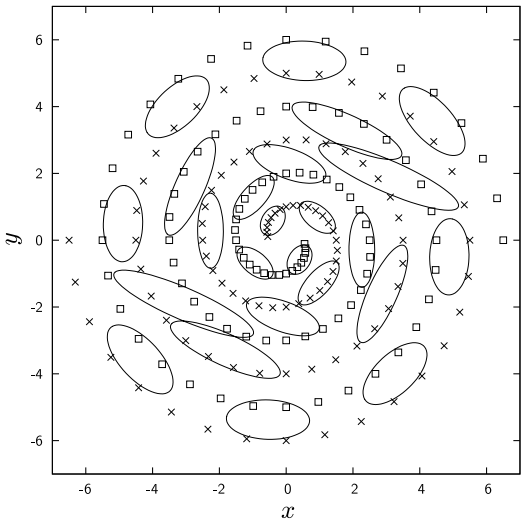

Figure 4.4: Density estimation results on Two-Spiral benchmark. Each ellipse represents an equiprobability contour of a Gaussian component of the joint probability density $f_{X,Y}(x,y)$ (the class random-variable has been marginalized out).

In this experiment, DGME was used to approximate a joint probability density function of the (x,y) coordinates and the class C. A regression function $E[C|X = x, Y = y]$ was extracted from the joint probability density, where C is a continuous valued random number. The output of the regression function is thresholded to determine the class of a point at the coordinates (x,y). Figure 4.4 shows the Gaussians of the joint probability

density $f_{X,Y}(x, y)$ after the merging procedure was applied and the class variable is marginalized out. The ellipses represent equiprobability contours of the Gaussians used to form the joint probability density $f_{X,Y}(x, y)$. Note that each Gaussian has its own weight and the weighted sum of the Gaussians represents the joint probability density given by

$$f_{X,Y}(x, y) = \sum_{i=0}^{M} \hat{w}_i \phi_i, \tag{4.3.1}$$

where M is the number of Gaussians in the mixture, $\hat{w}_i$ is a mixing coefficient and ϕ_i is a Gaussian $\mathcal{N}(\mu_i, \Sigma_i)$. This test of the DGME/GMR method on this classification problem suggests its suitability for classification tasks.

4.3.2 Breast Cancer Data Classification Results

The Wisconsin Diagnostic Breast Cancer dataset (Lichman, 2013) was published in 1991, and has been used as a benchmark dataset in the area of classification. It consists of 569 observations derived from clinical cell-tissue samples taken from tumors of different patients. Each observation includes several features related to the cell sample, as well as a diagnosis of malignant or benign.

In the context of classification, the task is to predict whether a tissue sample is malignant or benign based on the different features that characterize the sample. To apply a mixture model for the purpose of classification, one can build a model from observations of the form $\begin{bmatrix} D & F_1 & F_2 & \cdots & F_N \end{bmatrix}$, where D represents whether the diagnosis was malignant, and takes on the values 0 (non-malignant) or a 1 (malignant). The values F_i in the observation represent the other features measured on each tissue sample. Examples of these features are the mean, standard-error, and worst-case values for cell radius, texture, perimeter, area, smoothness, etc. After a model has been built using observations like this, it is possible to use the model to predict the conditional expected value $\hat{D} = \mathrm{E}[D \mid F_1, F_2, \cdots, F_N]$. If it turns out that $\hat{D} > 0.5$, then we can say that the prediction leans in favor of a malignant diagnosis, and when $\hat{D} < 0.5$, then the diagnosis is predicted to be benign. The closer $\hat{D}$ is to 0 or 1, the more confident this prediction is.

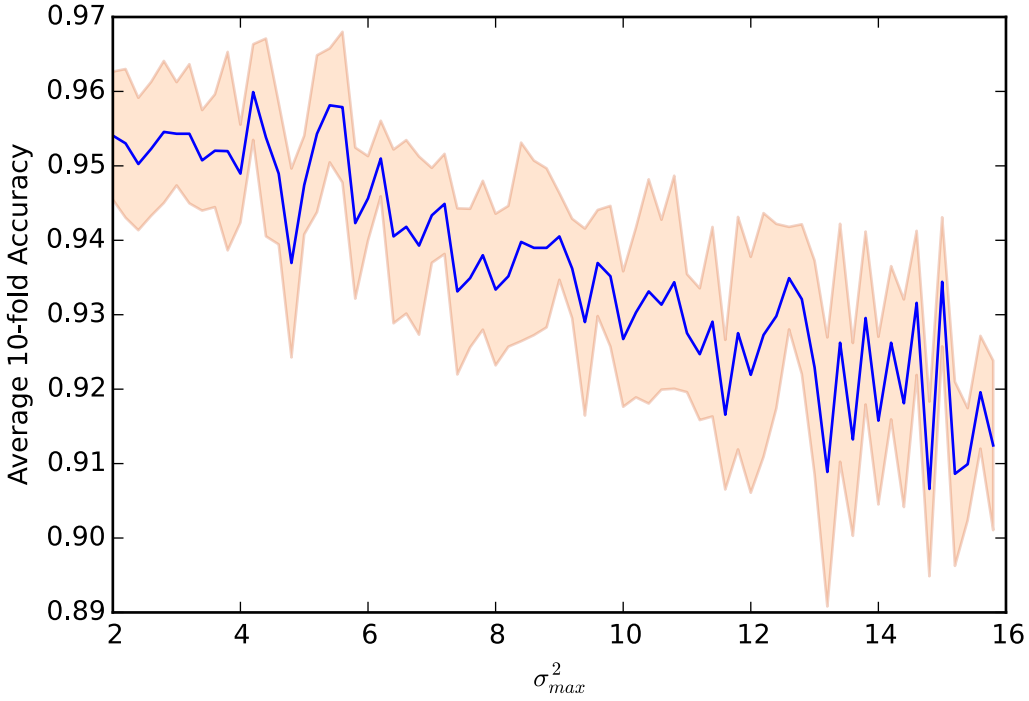

Figure 4.5: Classification accuracy when applying DGME to the Wisconsin Diagnostic Breast Cancer dataset. The hyperparameter σ^2_{max} is swept from 2 to 16, with a fixed value of $\sigma^2_0 = 0.01$. The 1-σ confidence interval is plotted surrounding the average classification rates.

In (Wolberg et al., 1995), a classification task was performed on this dataset using both logistic regression, and a method called "Multisurface Method-Tree" (MSM-T) (Bennett, 1992), which constructs a decision-tree by using linear programming. With varying numbers of 10-fold cross-validation runs, logistic regression was reported to provide a 96.2% classification accuracy, and MSM-T provided a 97.5% classification accuracy. DGME was applied to this same classification task. The training data given to the DGME algorithm was first passed through a whitening transformation which used the means and standard-deviations of each feature, computed over the training data, in order to normalize the data. The validation data was whitened using these same mean and standard-deviation values. Using ten runs of 10-fold cross-validation, average classification accuracies ranging between around 90-96% were achieved, depending on the value chosen for the σ^2_{max} hyperparameter. These results are shown, along with the 1-σ confidence interval, in Figure 4.5. Notice that there is a (possibly linear) negative

correlation between σ^2_{max} and classification accuracy, which underscores the importance of choosing σ^2_{max} well. It is also clear, however, that the accuracy is not overly sensitive to the chosen value of σ^2_{max} – even over a large range of σ^2_{max} values, the worst accuracy value A_w is only about 3.5% less than the best accuracy value A_b (i. e. $\frac{A_b - A_w}{A_b} \approx 0.035$).

What is noteworthy about these results is the fact that DGME is able to give comparable classification rates to those published in (Wolberg et al., 1995), while having the benefit of being a much more flexible representation than logistic regression or MSM-T are. A model estimated by DGME is more flexible in the sense that additional data can be used to incrementally update the model, and expected values can be computed for any subset of the observation elements.

4.3.3 Mackey-Glass Time Series Benchmark

An experiment described in (Kiong, Rajeswari, and Raoa, 2003) using the chaotic Mackey-Glass equation was performed with the DGME/GMR method to investigate the suitability of the method for prediction problems. The task is to predict the time series given by

$$x(t+1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \tag{4.3.2}$$

where $a = 0.1$, $b = 0.2$, $\tau = 17$ and $x(0) = 1.2$. The function to be approximated has the form $x(t+6) = f(x(t), x(t-6), x(t-12), x(t-18))$. The DGME/GMR method is trained on 1000 samples of f where $124 \leq t \leq 1123$, and validated on another 1000 samples of f where $1124 \leq t \leq 2213$. Again, the joint probability density is learned first, and afterwards a regression function is extracted from the joint probability density that approximates the function f. Figure 4.6 shows the performance of the DGME/GMR method in predicting the sequence for the validation set.

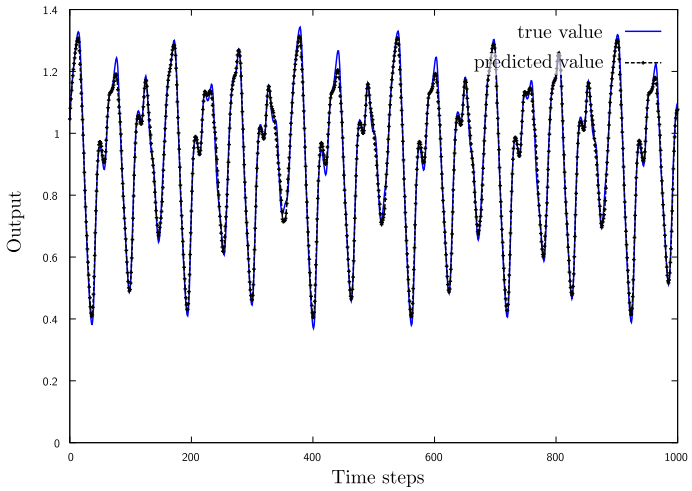

Figure 4.6: Prediction performance of the DGME/GMR method on the Mackey-Glass benchmark. The line with points represents the trajectory generated using this method.

4.3.4 Prediction with the Diabetes Dataset

To explore the regression capabilities of DGME + GMR, it was applied to the *diabetes dataset* available as part of the *scikit-learn* machine learning software (Pedregosa et al., 2011). This dataset consists of 442 training examples, each mapping a set of 10 physiological measurements taken on a patient to a value that indicates the amount that diabetes ended up progressing in that patient. For the following experiments, only a single feature from among the input measurements is considered, so that the results can be easily displayed on a 2D plot. Figure 4.7 shows the results of applying ordinary least-squares regression alongside DGME + GMR to this dataset. The *explained variance regression score* was used to measure the quality of the fit to the data. This score is computed as follows:

$$\text{explained variance} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i y_i^2}$$

where $\hat{y}_i$ represents a predicted value, and y_i represents an actual measured value.

In the case of using least-squares regression, the explained variance was computed to be 0.47, while for DGME + GMR, the explained variance was 0.45. The data was variance-scaled and mean-shifted before fitting the model with DGME, and the metaparameters

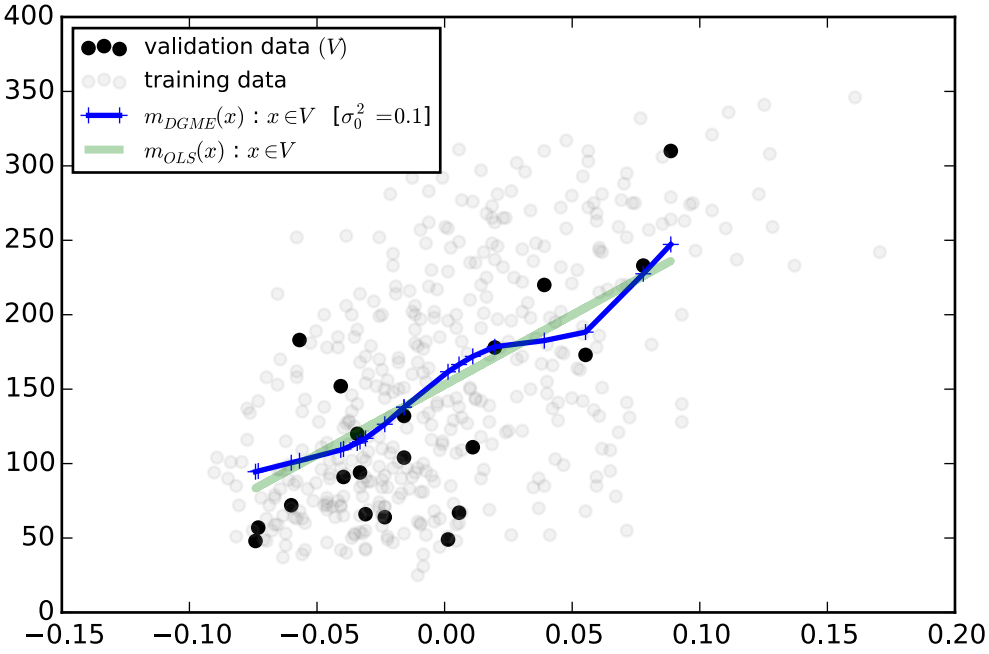

Figure 4.7: DGME+GMR and OLS Regression using the Diabetes Dataset.

used for DGME were $r_M = 4$, $\sigma_0^2 = 0.1$, $\sigma_{max}^2 = 1$, $N_{infl} = 10$, $N_q = 20$, $\alpha_q = 0.9$, and $\epsilon = 0.1$. The resulting number of components in the GMM was 9.

It is clear that if the data exhibits a global linear relationship, then OLS regression will outperform other modelling techniques in terms of speed. Nonetheless, the DGME + GMR performance is still reasonably good, and also offers additional information in the form of varying confidence bounds on predicted values (which come from each mixture-component's $\mathbf{\Sigma}$ value).

4.3.5 Concrete Strength Regression

A further scenario in which DGME was tested was in its ability to predict the compressive strength of High-performance Concrete (HPC) given the densities of different ingredients used to form the concrete and the age of the concrete. Experiments applying both neural networks and standard least-squares linear regression to this problem were reported in

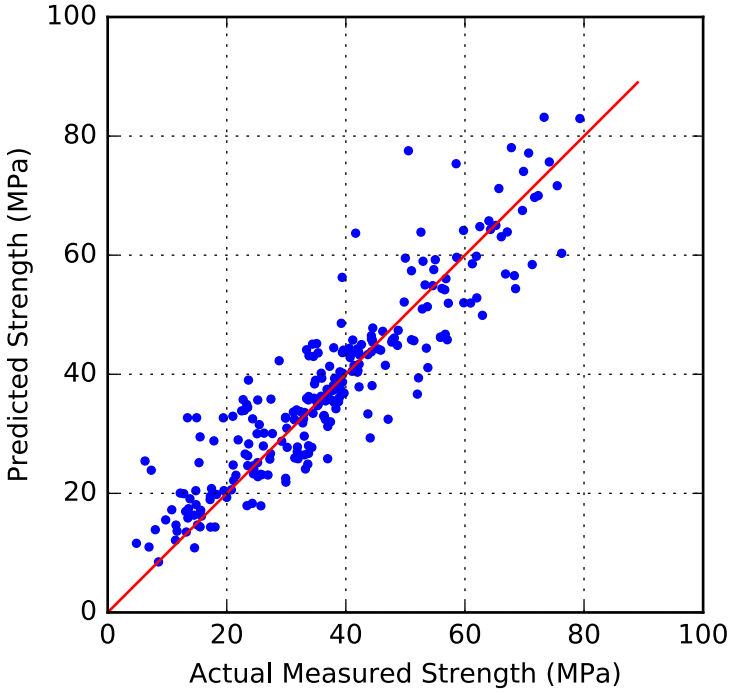

Figure 4.8: Predicted strength values vs. actual strength values that resulted from using DGME and GMR for regression.

(Yeh, 1998). A total of 1030 data-points were used for each experiment, and 3/4 of these points were used as training data, and the remain 1/4 of the points were used to validate the prediction accuracy. The primary measure of prediction accuracy used in (Yeh, 1998) was the coefficient of determination (R^2). The neural network used was trained using backpropagation and had 1 hidden layer and 8 hidden units. The results reported for linear regression had an average and standard deviation of $R^2 = 0.743 \pm 0.030$, whereas the neural network resulted in $R^2 = 0.885 \pm 0.036$ with 3000 learning-cycles. A graph depicting the regression results for DGME + GMR is shown in Figure 4.8.

These results were achieved with $\sigma_0^2 = 1$, $\sigma_{max}^2 = 2$, and $N_{infl} = 25$, $N_q = 50$, and $\sigma_{min}^2 = 0.001$, where the training and validation data was preprocessed by shifting by the mean and scaling by the standard-deviation of the training data. The coefficient of determination associated with these results was $R^2 = 0.796 \pm 0.030$, which is better than the R^2 value from least-squares regression, and worse than the neural-network's R^2 value. Again, while the regression performance doesn't match that of the neural network, this

can at times be a trade-off worth making for the flexibility that DGME offers in terms of online learning and multidirectional regression.

4.3.6 Conclusion

The experiments presented in this chapter show that using joint-probability models for modelling and regression is a viable alternative to standard regression methods in typical application domains. By learning a joint probability density as an intermediate step to deriving a regression function, every observable relationship between variables is captured, regardless of its causality or lack thereof. This makes the joint probability density flexible in the ways it can be used, in contrast to a regression function, which only represents a single (directed) relationship between variables.

The next chapter takes the validation of this method one step further, by demonstrating its use in a more practical scenario – localizing and controlling a legged robot.

Chapter 5

Control

In addition to being useful for prediction and classification tasks, it is also possible to use density functions estimated by DGME in order to control different kinds of systems. This chapter provides a brief introduction to control systems, an explanation of different ways that GMMs can be applied to control tasks, and some examples in which DGME is applied to control tasks in specific domains.

5.1 Control System Basics

5.1.1 Anatomy of a Control System

The goal of a control system is to bring the output state of a process to a desired state by applying a sequence of input control signals. Control systems are characterized as being either open-loop, where the controller (responsible for generating the input control signals) receives no feedback from the system, or closed-loop, where some kind of feedback signal is received. In general, closed-loop systems tend to be more robust in rejecting disturbances in the system, but can also be more susceptible to instability and oscillatory behavior. In this chapter, closed-loop systems will be considered such as the one depicted in Figure 5.1.

The *process* (i. e. "plant") represents the part of the system which is influenced by the controller in order to reach some desired state. The input to a control system is a desired

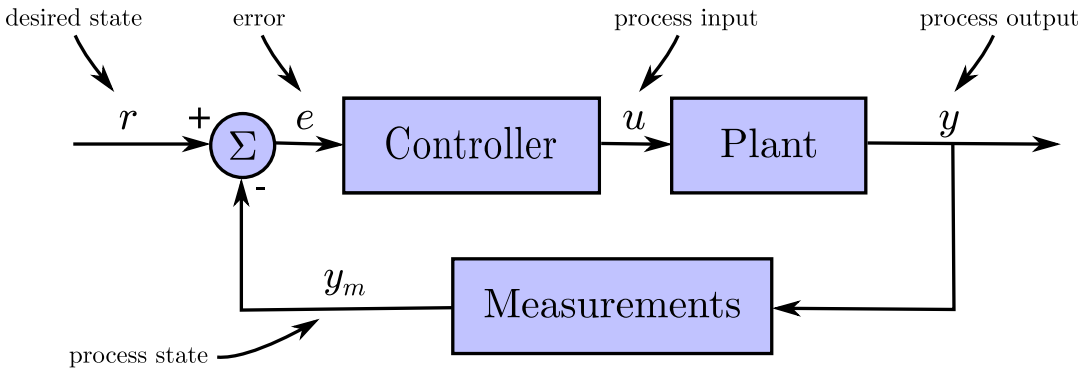

Figure 5.1: Components of a Feedback Control System

state. An *error* signal is traditionally defined as the difference between the process state and desired state. Specifically, the state can be defined as any set of variables representing some aspect of the process. These variables can include both observable and hidden variables. In the case of hidden variables, a model is necessary which specifies the relationship between observable and hidden variables.

The *controller* takes the error signal as its input, and generates a control signal (i. e. "process input") u which influences the process and its resulting state. The goal of the controller is to bring the error signal to have a value of zero, generally without "excessive" oscillation around zero.

Typically the parts of a single-input / single-output control system are modelled by means of a *transfer function*, which is derived by taking the Laplace transform of the input and output time-domain functions. Linear systems with multiple inputs and outputs, on the other hand, are traditionally modelled using *state-space equations*.

5.1.2 System Identification

In order for a controller to minimize the error, it must effectively issue a control signal which moves the present process state towards the desired state. Given a function $f : u \mapsto \Delta y_m$ (mapping from a control to a resulting change in process state), the task of the controller can be seen as performing an inversion of this function. In other words, the controller must implement a function $g = f^{-1} : \Delta y_m \mapsto u$. If this function exists, then given the desired state r and the present process state $y_m^{(0)}$, the desired control signal can

be obtained as $u = g(r - y_m^{(0)})$. This follows from the fact that we want the next process state $y_m^{(1)} = r$, and thus $\Delta y_m = y_m^{(1)} - y_m^{(0)} = r - y_m^{(0)}$. To approximate g, f must be known or approximated. The process of identifying f is known as *system identification*.

5.2 Using Mixture Models for Control

In the context of a control system, a GMM can be used to approximate the function f mentioned in Section 5.1.2. More generally, a GMM can be used to estimate a joint density between inputs, outputs, and possibly other auxiliary variables. After such a model has been estimated, it is possible using the techniques presented in Chapter 4 to infer the required inputs that result in some desired output values. In what follows, DGME is used in order to estimate how a robotic system moves. In this case, the entire robot serves as the *process* that we are controlling, where the inputs are commands issued to the robot, the outputs are the resulting movement of the robot, and auxiliary variables include the state of the environment.

5.3 Practical Application: Robot Localization and Control

In order to control a robot's behavior, it is necessary to have a model of how the robot behaves. Such a model is formally known as a *motion model*. The following sections describe motion models in greater detail, and explain an experiment in which using GMM based regression allows a legged robot to be more accurately controlled by including information in the GMM about the robot's environment.

5.3.1 Motion Modelling

A robot motion model, broadly speaking, represents the relationship between commands issued to a robot and the robot movement that results from these commands. In many modern robotic systems, an accurate and reliable motion model serves a crucial role. Most

Simultaneous Localization and Mapping (SLAM) approaches, for example, use a motion model in the prediction step of a recursive Bayesian estimation algorithm (S. Thrun and Leonard, 2008). An accurate motion model can also be used to perform dead-reckoning, aiding trajectory-following tasks when other forms of odometric information is limited.

In robotics it is a common assumption that a robot's motion model is Markovian. In this case, the motion model is often defined in terms of a probability density function $p(s_t \mid s_{t-1}, u)$, where u is a command issued to the robot when it is in state s_{t-1}. Traditionally, u has been defined (for wheeled robots moving in a 2D plane) as either a set of translational and rotational velocities to be used to directly control the robot's wheels (in which the state s represents the robot pose as a function of time), or as a relative odometry (e. g. a vector $\begin{bmatrix}\delta_1 & r & \delta_2\end{bmatrix}$) where the robot is commanded to turn by an angle, move forward a certain distance, and turn again by another angle (in this case, the robot pose represented by s does not depend on time) (Sebastian Thrun, Burgard, and Fox, 2005).

Typically these probabilistic models depend on the existence of a closed-form parameterized deterministic model, consisting of one or more equations that define a future state in terms of the present state and the command issued. By adding uncertainties to this deterministic model, it is transformed into a probabilistic model. For example, a simple deterministic model relating distance to velocity, $d = v\Delta t$, can be made probabilistic by treating d and v as Gaussian-distributed random variables: due to mechanical tolerances in the robot, v has an associated uncertainty that can be modeled using the random variable's σ parameter. The dependence on closed-form models is found in nearly all works on this topic up to now. While such models are useful when working with wheeled robots in a plane, they are not adequate for modelling the complex motions of legged robots, for which it is often difficult and time-consuming to find a deterministic mapping from the command-space to the configuration-space.

Using DGME to build a mixture model that represents a motion-model not only overcomes this limitation, but is also flexible in that it allows arbitrary sensory information to be directly incorporated into the motion model. By allowing information describing the terrain, for example, to be incorporated into the model, a more accurate motion-model can be estimated, because the movement of a robot is closely tied to the properties of the terrain on which it moves.

5.3.2 Overview of Experiment

Learning the motion model of legged robots is challenging due to the complex kinematics of the robot and the complexity of the interaction it has with the environment during locomotion. Initial experiments were performed with the Scorpion[1] robot to test the extent to which a joint probability density of poses and commands could capture the robot's forward and inverse motion models. The pose of the robot was measured using a camera and marker-based motion capture system. The pose of the robot includes the Cartesian coordinates (x, y) and heading angle θ of the robot. The experimental environment used was a smooth horizontal laboratory floor surface. The space of commands used in the experiment to control the robot was the Cartesian product $C_T = F \times L \times T$, where $F = \{-0.8, 0, 0.8\}$, $L = \{-0.8, 0, 0.8\}$ and $T = \{-0.8, 0, 0.8\}$. The set F stands for forward-backward movements with the maximum and minimum values of 1 and -1 respectively, and the set L stands for lateral left-right movements with the maximum and minimum values of 1 and -1 respectively. Similarly, the set T stands for left-right rotations with the maximum and minimum values of 1 and -1 respectively. The Scorpion robot was sent random commands from the command space C_T and all changes in the pose of the robot were recorded. In the experiment, equal probabilities were given to all of the commands, and on average each command was repeated (non-consecutively) five times on the robot. By repeatedly issuing a command $\mathbf{c}$, and observing the robot's resulting change in pose $\boldsymbol{\gamma}$, a joint probability density over $\begin{bmatrix}\mathbf{c} & \boldsymbol{\gamma}\end{bmatrix}$ vectors was built. Using terminology from Section 5.1.2, $\mathbf{c}$ represents the control signal u, and $\boldsymbol{\gamma}$ represents the change in process state Δy_m.

5.3.2.1 Extracted Forward Motion Model

In order to validate the GMM representing the learned joint probability density, the function $\boldsymbol{\gamma}(\mathbf{c}) = E[\boldsymbol{\Gamma} \mid \mathbf{C} = \mathbf{c}]$ was computed from the GMM. Here, $\mathbf{C}$ and $\boldsymbol{\Gamma}$ represent random vectors for the command and change in pose, respectively. This function represents the forward motion model, and was used to estimate the pose of the robot over 50 time-steps in a separate experiment to assess its prediction quality. Figure 5.2 shows the result obtained after this motion model was used to estimate the robot's pose. From the figure, it can be seen that the extracted function $\boldsymbol{\gamma}(\mathbf{c})$ predicts the robot's

[1] A detailed description of this robot is available in (Klaassen et al., 2002).

position relatively well.

5.3.2.2 Extracted Inverse Motion Model

The inverse motion model $g : \boldsymbol{\gamma} \mapsto \mathbf{c}$ was also extracted from the learned joint probability density, which maps the change in robot pose to a command to be sent to the robot. This model was used to control the robot in a closed loop manner to traverse a figure-8 shaped trajectory. Waypoints were sampled from the trajectory and the nearest waypoint to the current position of the robot was used to calculate the necessary change in pose. The inverse motion model was then used to determine which command to send to the robot, given the necessary change in pose. Figure 5.3 shows the results of using the inverse motion model in this way for trajectory following. As can be seen in the figure, the robot is able to follow the trajectory reasonably well. Along the target trajectory where the curvature is high, the robot tends to execute commands having larger rotational effects.

5.3.3 Review of Motion-modelling Related Works

Before the appearance of probabilistic SLAM methods, getting an accurate estimate of a robot's position often relied on a parameterized model relating data reported by odometry sensors to the estimated motion of the robot. The problem of robot calibration, which involves properly selecting the parameters of this model, has been the topic of various works. For example, the work of Borenstein and Feng (1996) presents methods for manually choosing these parameters for wheeled robots.

After SLAM methods started to become more prevalent, it became possible to use probability distributions reported from the SLAM algorithm to estimate a robot's ground-truth pose. This estimate could then be used in conjunction with calibration methods to automatically learn the parameters of a motion model by simply moving the robot.

Roy and Sebastian Thrun (1999), for example, use a maximum likelihood method for estimating model parameters. The parameters $\boldsymbol{\theta}$ are updated with an exponential estimator that integrates the parameter values $\boldsymbol{\theta}^*$ that maximize the likelihood function

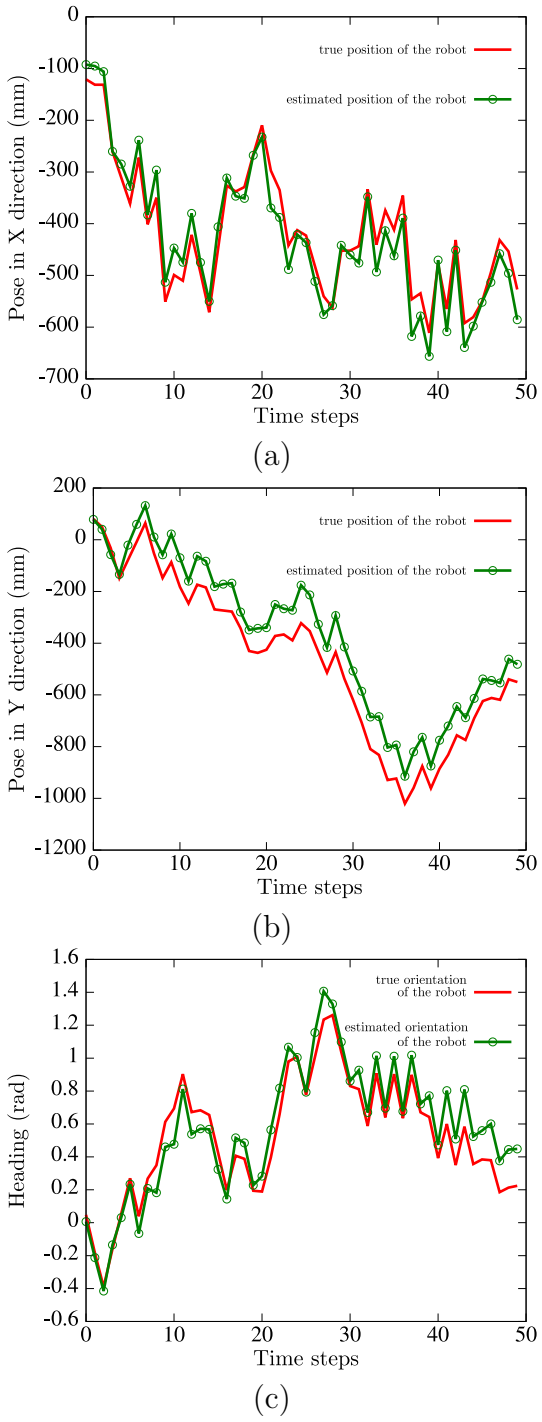

Figure 5.2: Performance of the motion model in estimating the pose of the Scorpion robot on a flat surface: (a) pose estimation in the x-direction vs. time-steps. (b) pose estimation in y-direction vs. time-steps, and (c) heading estimation vs. time-steps.

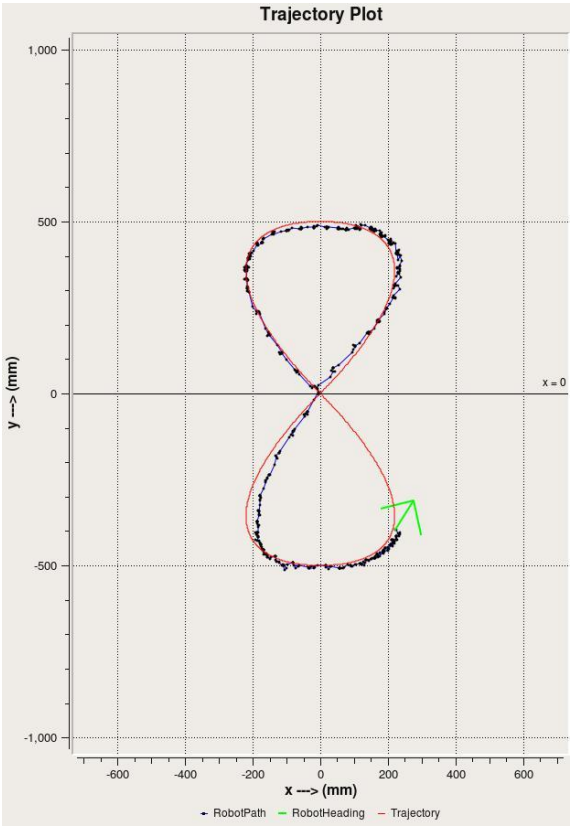

Figure 5.3: The inverse motion model used in trajectory following. The figure shows the robot while following the trajectory, and the arrow in the figure shows the current orientation of robot.

$p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{o}, \boldsymbol{\theta}^*)$, where $\mathbf{s}_t$ are laser-scan measurements and $\mathbf{o}$ are odometry measurements. By using an exponential estimator, incremental updates to the parameters are possible which do not depend on keeping a history of sensor-measurement data.

In work done by Eliazar and Parr (2004b), the model parameters are learned using an Expectation-Maximization (EM) method, in which the expectation step involves using the SLAM algorithm (in this case, DP-SLAM 2.0 (Eliazar and Parr, 2004a)) to generate a set of possible trajectories for a given set of motion model parameters and associated likelihoods, and the maximization step uses a least-squares approach for determining the set of model parameters that maximizes the likelihood values. While the least-squares method presented in (Eliazar and Parr, 2004b) assumes that a history of the training data is available, Visatemongkolchai and Zhang (2007) apply two incremental least-squares methods in order to learn model parameters. Because of their incremental nature, these

methods can be used in an on-line fashion, updating the model parameters after each new set of measurements.

Stronger and Stone (2005) introduce a technique for calibrating sensor and motion models simultaneously. These models are represented in a deterministic manner using polynomials. The polynomial coefficients are learned with a two-step cyclic algorithm in which the first step estimates the sensor-model parameters given the current motion-model, and the second step estimates the motion-model parameters given the current sensor-model.

The method presented by Kaboli, Bowling, and Musilek (2006) also calibrates sensor and motion models simultaneously, but in contrast to (Stronger and Stone, 2005), uses a Markov chain Monte Carlo (MCMC) technique in which a sample is drawn from a posterior distribution over model parameters. To estimate the true model parameters, the sample's elements are either averaged, or the maximum a posteriori sample element is selected. Additionally, the sample's elements can be used to approximate model posteriors. Each of these methods were tested in a Monte Carlo Localization scenario on a simulated wheeled robot and on a Sony AIBO robot.

Martinelli, Tomatis, and Siegwart (2007) present a method of estimating the parameters of an odometry error model by using a modified Extended Kalman Filter (EKF) to simultaneously estimate the robot pose and error parameters. The odometry model relates odometry measurements (i. e. wheel encoder measurements) of a wheeled robot to the robot's estimated true odometry. The error model is split into systematic and non-systematic components, and each component is separately estimated based on the other component's estimate. Sjoberg, Squire, and Martell (2007) present a method that uses a slightly simplified deterministic model from that presented in (Eliazar and Parr, 2004b), but in which the model's random variables are represented using a bimodal Gaussian Mixture Model. The method depends on the use of a SLAM method (DP-SLAM 2.0 is used), and the model parameters are estimated by observing the motion of the particles in a particle-filter. As in (Roy and Sebastian Thrun, 1999), an exponential estimator was used to make the method work in an online fashion, in which the decay factor is adjusted according to the average quality of the particles in a particle-set.

The work done by Hoffmann (2007) does not deal with the problem of parameter estimation, but instead presents a way of incorporating proprioceptive information

into the motion model. To do this, the motion model's error is decomposed into two components: the error ϵ_{coll} due to collisions and slippage, and the error ϵ_{odo} intrinsic to the robot morphology and odometry sensors. Collisions and slippage conditions are modeled as the states of a state-machine, and the value of ϵ_{coll} depends on which state the robot finds itself in. Transitions between states occur due to the observation of specific proprioceptive data patterns.

5.3.3.1 Suitability for Legged Robots

Nearly all of the methods in the previously mentioned works define the motion model as a set of fixed form "ideal model"[2] equations to which uncertainty is added by treating some of the variables as random. Furthermore, every one of these methods requires that the motion model be parameterized with a fixed number of parameters.

For kinematically complex legged robots, developing the equations of an ideal model can be very difficult and time consuming[3]. Because of this, *these methods generally do not lend themselves well to modelling the motion of legged robots.*

To properly capture the motion model of such a legged robot, as few as possible assumptions should be made about the form of the motion model. For this reason, the DGME method discussed in Chapter 3 is used, which allows for a flexible model form that can be dynamically changed over time. The details of how this method is applied to the task of motion-modelling is discussed in the next section.

5.3.4 Motion Model Representation

Motion models are frequently defined in the form of a conditional state-transition probability density, where the state is defined as the robot's pose, and the probability is conditioned on the commands issued to the robot. When operating in physically unobstructed areas, the change in a robot's pose from the current state to the next state depends much more on the command issued to the robot than it does on the robot's

[2] by "ideal model", a model which assumes an ideal, non-stochastic world is meant.

[3] There are some legged robots (e. g. the Sony AIBO) whose kinematics are simple enough to develop equations approximating an ideal model.

current pose. Because of this, if we assume the absence of obstructions, the motion model can be simplified by representing it as a conditional probability of the *change in pose*[4] conditioned upon the command issued.

DGME was chosen for estimating the motion model, so that the number of Gaussian components can vary in order to best fit the characteristics of the underlying system. The resulting model, a weighted set of Gaussian components, can be decomposed into a probability distribution over the change in pose of the robot for each unique command.

5.3.4.1 Formal Description

In general, a motion model can be represented in the form of a joint pdf over commands, environmental states, and change in pose values. When the set of possible commands is finite and discrete, a constraint can be added to DGME such that observations associated with a particular command will never be merged into mixture components associated with a different command. In this case, we can view the model as being decomposed into several sub-models, one per distinct command.

Formally, we can represent the motion model in this case as $\{\mathcal{M} : C \times V \rightarrow P\}$ where C is the set of all commands, V is a set of environmental states[5], and P is a set of probability density functions $p(\boldsymbol{\gamma})$, where $\boldsymbol{\gamma}$ is a change in pose measurement. In simple cases where the robot's environment is not expected to change much, V may be treated as containing one static environmental state. In this case, the model can be written as $\{\mathcal{M}_{\mathcal{S}} : C \rightarrow P\}$. In practice, a robot's command space (i. e. the range of possible commands that can be issued to the robot) can be discretized[6] into n commands $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_n\}$, where each command is a vector representing various parameters that can be controlled for a robot (e. g. for a wheeled robot, these parameters might be the angular velocity of each wheel). If the commands are discretized in this way, then according to the mapping defined by $\mathcal{M}_{\mathcal{S}}$, a corresponding probability density in P (that is a likelihood function for $\boldsymbol{\gamma}$) exists for each of these commands. We represent each command $\mathbf{c}_j$'s corresponding density

[4] This change in pose is measured relative to the robot's pose prior to the execution of a command.

[5] An environmental state simply represents some information about the environment immediately surrounding the robot. For example, it could include information about the terrain's roughness, slope, or moisture content.

[6] Although the remainder of this chapter assumes a discrete command space, the representation presented here can also be used with continuous command spaces.

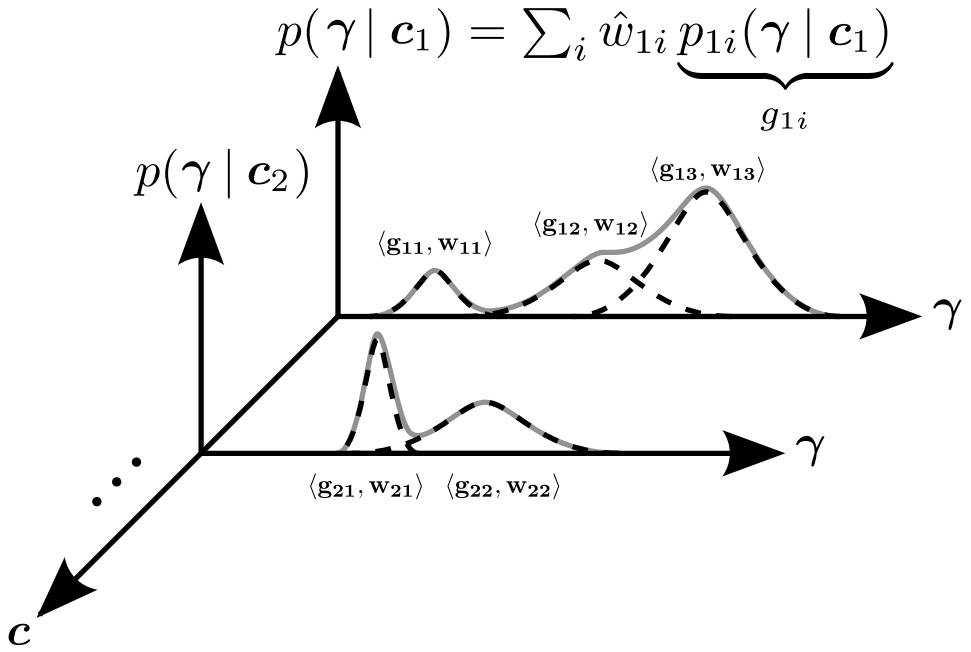

Figure 5.4: Motion model representation: for each command in the discretized command space, a conditional probability distribution is maintained as a weighted sum of Gaussians.

function $p(\boldsymbol{\gamma} \mid \mathbf{c}_j) \in P$ as a variable-sized set of "weighted Gaussian" pairs,

$$G_j \equiv \{(g_{j1}(\boldsymbol{\gamma}), w_{j1}), (g_{j2}(\boldsymbol{\gamma}), w_{j2}), \ldots, (g_{jm}(\boldsymbol{\gamma}), w_{jm})\}, \tag{5.3.1}$$

such that

$$p(\boldsymbol{\gamma} \mid \mathbf{c}_j) = \sum_{i=1}^{m} \hat{w}_{ji} g_{ji}(\boldsymbol{\gamma}), \tag{5.3.2}$$

where $g_{ji}(\boldsymbol{\gamma})$ is a conditional multivariate Gaussian distribution:

$$g_{ji}(\boldsymbol{\gamma}) = p_{ji}(\boldsymbol{\gamma} \mid \mathbf{c}_j) \sim \mathcal{N}(\mu_{ji}, \Sigma_{ji}), \tag{5.3.3}$$

and

$$\hat{w}_{ji} = w_{ji} \;/\; \sum_{k=1}^{m} w_{jk}. \tag{5.3.4}$$

A schematic depiction of this representation for the case of discrete command spaces can be seen in Figure 5.4. The w_{ji} values are *unnormalized* weights, and are equivalent to the number of observations that have contributed to the corresponding Gaussian.

5.3.5 Learning the Motion Model

The method in which DGME was employed for learning a motion model is described here. Though it is possible to easily incorporate several different forms of sensory data into the motion-model, in this discussion the measurements are restricted to include only the robot's pose for the sake of simplicity.

The primary algorithm of this method is depicted in Algorithm 7.

Algorithm 7: Incremental Motion Model Update

Data: Sequence of $\begin{bmatrix} command & pose\ measurement \end{bmatrix}$ pairs.
Result: Dynamically updated motion-model for each *command*.
Select and perform command $\mathbf{c}_j \in C$ on robot;
$\boldsymbol{\zeta} \leftarrow$ new pose measurement;
repeat
  $\boldsymbol{\zeta}_{previous} \leftarrow \boldsymbol{\zeta}$;
  Select and perform command $\mathbf{c}_j \in C$ on robot;
  $\boldsymbol{\zeta} \leftarrow$ new pose measurement;
  $\boldsymbol{\gamma} \leftarrow \boldsymbol{\zeta} - \boldsymbol{\zeta}_{previous}$; ***// change in pose***
  Incorporate $\begin{bmatrix} \mathbf{c}_j & \boldsymbol{\gamma} \end{bmatrix}$ into the model, updating parameters of $p(\boldsymbol{\gamma} \mid \mathbf{c}_j)$;
end

Initially, the robot's pose $\boldsymbol{\zeta}_0$ is recorded, and a command $\mathbf{c}_j$ is chosen from the set of possible commands C. This command is issued to the robot, and the final pose $\boldsymbol{\zeta}_1$ is recorded after the command has completed its execution. The measurement vector $\boldsymbol{\gamma}$ is calculated as $\boldsymbol{\zeta}_1 - \boldsymbol{\zeta}_0$ (i. e. the robot's change in pose), and incorporated into the motion model using the DGME algorithm. Again, a command is selected, and this procedure is repeated ad infinitum, calculating the measurement vector in the same manner (subtracting the previous pose measurement from the current).

Because DGME is incremental in nature, the model can be updated without the need to store a history of observations. This incremental nature reduces both the memory and

computational requirements of this algorithm considerably, in contrast to methods in which some or all observations are required to re-estimate the density.

5.3.6 Learning an "Extended" Model

The discussion thus far has assumed that the model represents only a simple mapping between commands and pose differences. The GMM based model can, however, be extended to include exteroceptive and proprioceptive data[7].

Specifically, if we have some data (e. g. information about the terrain) we would like to incorporate into the model represented as $\mathbf{z}$, the change in pose vector $\boldsymbol{\gamma}$ can be augmented by $\mathbf{z}$ to form a new data vector

$$\begin{aligned} \mathbf{d} &= \boldsymbol{\gamma} \| \mathbf{z} \\ &= \begin{bmatrix} x_1 & \cdots & x_n & z_1 & \cdots & z_m \end{bmatrix}. \end{aligned} \tag{5.3.5}$$

It is then possible to use $\mathbf{d}$ in the same way that $\boldsymbol{\gamma}$ was previously used in Section 5.3.5. The only necessary change is to modify Algorithm 7 such that the measurement $\mathbf{z}$ is taken before the command $\mathbf{c}_j$ is issued to the robot. The vector $\mathbf{d}$ is then formed and passed to `DGME_UPDATE_FROM_OBS` in place of $\boldsymbol{\gamma}$.

The resulting motion model will now represent $p(\boldsymbol{\gamma} \| \mathbf{z} \mid \mathbf{c}_j)$, where each Gaussian component g_{ji} of this density function has a mean vector $\mu^d_{ji} = \mu^\gamma_{ji} \| \mu^z_{ji}$ and covariance matrix $\Sigma^d_{ji} = \begin{bmatrix} \Sigma^{\gamma\gamma}_{ji} & \Sigma^{\gamma z}_{ji} \\ \Sigma^{z\gamma}_{ji} & \Sigma^{zz}_{ji} \end{bmatrix}$.

5.3.7 Using the Extended Model

Having built a motion model with these augmented observation vectors, we can now consider how to make use of the model. For the purposes of modelling motion, the

[7] Data is most effectively incorporated when it can in some way be represented as a vector in a Euclidean space. Fortunately there is often a means of representing data in this way.

density $p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j)$ is not directly useful. Rather, we would like to have $p(\boldsymbol{\gamma} \mid \mathbf{c}_j, \mathbf{z})$. To calculate the latter density from the former, we begin by writing the conditional probability relationship for the latter density:

$$p(\boldsymbol{\gamma} \mid \mathbf{c}_j, \mathbf{z}) = \frac{p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j)}{p(\mathbf{z} \mid \mathbf{c}_j)}. \tag{5.3.6}$$

Because $p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j)$ is represented as a mixture of Gaussians, the numerator in this equation can be written as

$$p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j) = \sum_{i=1}^{m} \hat{w}_{ji}^d g_{ji}^d(\boldsymbol{\gamma}\|\mathbf{z}), \tag{5.3.7}$$

where

$$g_{ji}^d(\boldsymbol{\gamma}\|\mathbf{z}) = p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j) \sim \mathcal{N}(\mu_{ji}^d, \Sigma_{ji}^d). \tag{5.3.8}$$

Given how marginalizing out variables in a Gaussian distribution results in another Gaussian distribution (see (4.2.6)), the denominator in (5.3.6) can be written as

$$p(\mathbf{z} \mid \mathbf{c}_j) = \sum_{i=1}^{m} \hat{w}_{ji}^z g_{ji}^z(\mathbf{z}), \tag{5.3.9}$$

where $\hat{w}_{ji}^z = \hat{w}_{ji}^d$ and

$$g_{ji}^z(\mathbf{z}) = p(\mathbf{z} \mid \mathbf{c}_j) \sim \mathcal{N}(\mu_{ji}^z, \Sigma_{ji}^{zz}). \tag{5.3.10}$$

Here we have marginalized the $\boldsymbol{\gamma}$ components out of $p(\boldsymbol{\gamma}\|\mathbf{z} \mid \mathbf{c}_j)$. Thus, by using (5.3.7) and (5.3.9), both which can easily be calculated from our motion model, we can determine $p(\boldsymbol{\gamma} \mid \mathbf{c}_j, \mathbf{z})$, which represents *an improved motion model that takes advantage of additional sensory data.*

5.3.8 Effect of Incorporating Terrain Data

Cross-validation was used in order to evaluate the accuracy of a motion model: a set of observations (either via simulation or from a real robot) is taken, and one portion of these observations is used for building the model, while the remaining observations are used for validating the model's ability to predict motion. This technique was chosen to assess the extent to which terrain information contributes to model accuracy. In particular, a total of 390 data observations from the Scorpion robot were collected, in

which every possible command of the form $\begin{bmatrix} long & lat & turn \end{bmatrix}$[8] was issued 5 times with three different initial robot orientations (i. e. each command is issued a total of 15 times), where $long, lat, turn \in [-0.5, 0, +0.5]$. Not including the *no-op* command $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ & this results in a set of 26 distinct commands. With the experimental setup pictured in Figure 5.5, the following procedure was used for collecting the data:

1. A command $\mathbf{c}_j$ is chosen randomly (without replacement) from the above defined command set.

2. The robot is placed on an 18° inclined slope with a starting orientation $\theta \in \{0°, 90°, -90°\}$[9]. The orientation angle is cycled each time this step is performed.

3. The current robot pitch and roll angles are recorded in the vector $\mathbf{z}$. This serves (for the case where the terrain is an inclined plane, as in this experiment) as an indirect measurement of the shape of the terrain the robot is on.

4. The robot is issued the command $\mathbf{c}_j$. Each command's execution requires a fixed time duration to carry out.

5. Before and after executing a command, the robot's pose is recorded using a marker based visual pose tracking system[6].

6. The observation $(\mathbf{c}_j, \mathbf{z} \| \boldsymbol{\gamma})$ is added to a sample set S_1, where $\mathbf{z} = \begin{bmatrix} robot\ pitch & robot\ roll \end{bmatrix}$ and $\boldsymbol{\gamma}$ is the change in pose $\begin{bmatrix} \Delta x & \Delta y & \Delta z & \Delta roll & \Delta pitch & \Delta yaw \end{bmatrix}$.

7. Steps 3-6 are repeated 5 times.

8. Steps 2-7 are repeated until the robot has been placed with each starting orientation.

9. Steps 1-8 are repeated until all commands have been chosen.

The observations from this experiment were used for cross-validation in which the model was learned using randomly selected training sets, and was validated by calculating the log-likelihood of the training data given the learned model, $L = \sum_{n=1}^{N} \ln \left\{ \sum_{i=1}^{M} \hat{w}_{ji} g_{ji}(\boldsymbol{\gamma}_n) \right\}$,

[8] The $\begin{bmatrix} long & lat & turn \end{bmatrix}$ values influence the robot's motion in the *forward-backward*, *lateral-left-right*, and *turning-left-right* directions, respectively.

[9] When $\theta = 90°$, the robot is turned so that it looks down the incline.

Table 5.1: Model Quality Comparison: a 10-fold cross-validation is performed for data sets with and without terrain information. Shown are the mean and standard-deviation of the log likelihood values, averaged over 10 runs.

		Log Likelihood	
Data Set	# Observations	Without Terrain	With Terrain
Real	390	-371.9 ± 3.9	-162.2 ± 4.0
Simulated	830	308.5 ± 7.5	402.5 ± 7.9

where $\hat{w}_{ji}$ is the associated normalized mixture-parameter of the i^{th} Gaussian in the model for a command c_j.

The cross validation was performed using the collected data for two different cases:

1. The first case uses the "full perception" sample S_1 that contains all the measured information about the terrain. In this case, the conditional density in (5.3.6) is calculated from the model.

2. In the second case, the model is trained and validated using a "limited perception" sample S_2, in which the $\mathbf{z}\|\boldsymbol{\gamma}$ component of each sample S_1 is replaced with $\boldsymbol{\gamma}$. In other words, for each observation $(\mathbf{c}_j, \mathbf{z}\|\boldsymbol{\gamma})$ in S_1, there is a corresponding observation $(\mathbf{c}_j, \boldsymbol{\gamma})$ in S_2. In this case, the model estimates the conditional densities in (5.3.2).

For both cases, a stratified 10-fold cross validation was performed 10 times, and the results averaged. The results for both cases are shown in Table 5.1. To test the validity of this comparison when there are a larger number of observations, another data set was generated (containing a total of 830 observations) using a simulation. This data was analyzed in exactly the same manner as the data collected from the real robot, and the results are also shown in Table 5.1. With both the real and simulated data, the likelihood was much higher in the case where terrain data was included in the model. This shows that *incorporating and making use of terrain data in the model substantially improves the extent to which the model describes the robot's motion.*

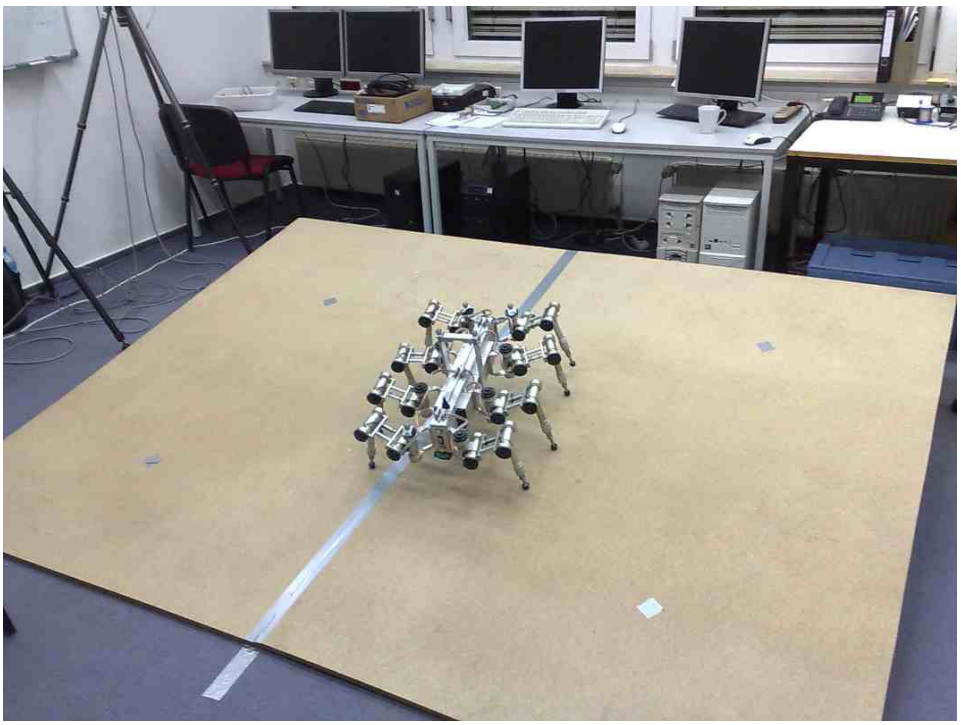

Figure 5.5: Experimental Setup: the Scorpion robot executes several movement commands in different orientations on a sloped plane.

5.4 Controlling Multimodal Processes

Using the notation from Section 5.1.2, most controllers are based on the assumption that the mapping between Δy_m and u is bijective (i. e. that every Δy_m value maps to a unique u value, and vice-versa). In real-world situations, this will not always be the case, however.

5.4.1 Limitations of Modern Control Methods

The requirement that the $u \Leftrightarrow \Delta y_m$ relation be bijective is not realistic for most natural processes, and as a result, much effort is invested in methods for making this possible, typically by engineering a new process or augmenting an existing process specifically so that it possesses this property. Furthermore, most modern control systems depend on the assumption that the process being controlled can be modelled using linear equations. When this is not the case, further effort must be expended augmenting the process in

order to linearize it.

Though most control methods do assume that a system is linear or linearizable, there are some modern control methods which do not assume any sort of globally linear mappings from input to output space. Such methods, known as switching state-space models (see Shumway and Stoffer, 1991), divide the state-space into multiple regions over which different controllers are made responsible. Although to a certain extent these methods share the local modelling property of DGME, they still rely heavily upon traditional controller design, and are bound by the limitations of traditional controllers.

In order for such a switching state-space model to properly handle multimodality in the modelled process, every mode in a process' input-output joint pdf that corresponds to a given change in state must be split into separate regions, each having a separate controller. This can become unscalable as the amount of multimodality in a system increases, and is unable to gracefully adapt to systems in which the quantity and behavior of states changes over time. In contrast, the system-identification and control methods presented in this thesis (in Chapter 3 and this chapter, respectively) do not require separate controllers for each case of a process' multimodality, but rather provide a general means of handling any amount of multimodality in a process.

5.4.2 The Challenge of Multimodal Systems

A typical means of controlling a system based on a probabilistic model is to employ conditional expectation, determining the expected value over a subspace of some random vector[10], given a specific value for some other random vector. Though this works well for cases in which a distribution has a single mode, when the (conditional) distribution has more than one mode, using conditional expectation to control a system can often produce undesirable results. This is illustrated in Figure 5.6. In contrast to the case of unimodal distributions, for which the expected value corresponds to the most probable value, the expectation of a multimodal distribution can often end up being a rather improbable value. If we were, for example, to interpret x in Figure 5.6 as the final resting place of some projectile which has been propelled, then there is a high likelihood that the projectile will come to rest somewhere near x_1 or x_2, but it is rather unlikely that

[10]For generality, a random vector can consist of one or more elements, each being a random variable.

the projectile will stop moving near $E[x]$. presented.

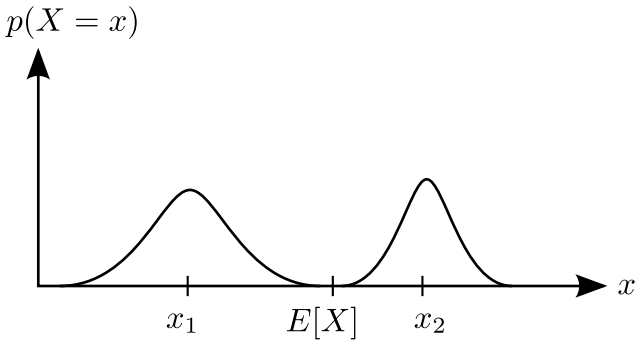

Figure 5.6: Expectation of a Multimodal Distribution: instead of representing the value of highest likelihood, expectation results in a value with a relatively low likelihood.

5.4.2.1 Projectile in a Magnetic Field

An example will now be presented that provides a more detailed illustration of the problem just described, and will be used throughout the remainder of this chapter. The *projectile in a magnetic field* problem is defined as follows: a ferromagnetic (i. e. able to be influenced by magnetic fields) projectile which is constrained to move in a 2D plane is positioned at a starting location. From this location, it is propelled within a rectangular area towards the opposite side of the rectangle. The initial velocity and projectile mass are held constant, while its starting angle can be chosen as desired. See Figure 5.7 for a visual depiction of this scenario.

If we were asked to select an angle which would cause the projectile to intersect the opposite side of the rectangle at some specified location, this would not be very difficult provided that we had a model which maps angles to locations. In the case of our proposed scenario, however, the problem is that there is a magnetic field which exerts a force on the projectile while it is propelled. To make matters worse, this field is not constant – its direction and magnitude at a given time are not known in advance.

In the interest of aiding understanding, the projectile problem we will consider here will constrain the initial projectile angle (in degrees) θ_0 so that $\theta_0 \in [-15°, 15°]$. Furthermore, the initial position of the projectile will always be at $(0, 0)$ in the x-y coordinate system

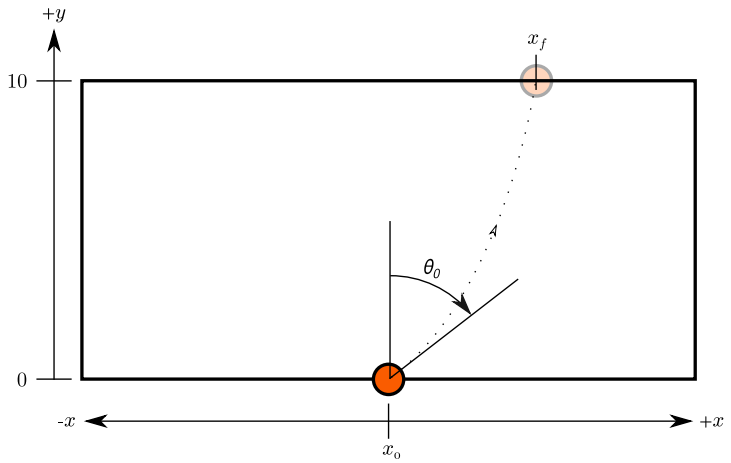

Figure 5.7: Basic Scenario of the *projectile in a magnetic field* problem: the projectile starts at position x_0 at an initial velocity of v_0 and initial angle of θ_0, and ends when it intersects $y = 10$ at position x_f.

shown in Figure 5.7, the initial velocity is fixed at $v_0 = 5\,\mathrm{m/s}$, and the mass of the projectile is fixed at $m = 0.4000\,\mathrm{kg}$. Thus the only value that can be varied is θ_0 – this can be considered to be the *input* variable in the projectile control system. The *output* variable of this system is the x-position of the point at which the projectile intersects the $y = 10$ line.

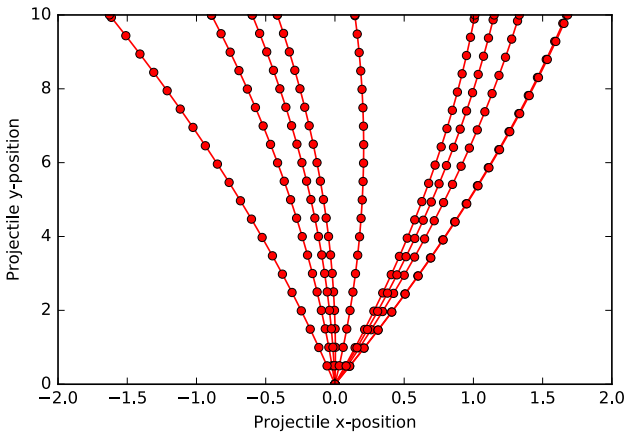

Figure 5.8: Projectile motion under the influence of a constant unidirectional magnetic field: several θ_0 values were randomly chosen, and each corresponding projectile trajectory is shown. Each point represents the projectile's location at a particular simulation step.

5.4.2.2 Simple Case: Constant Magnetic Field

Now that we have defined the problem, let's consider how we might apply mixture-model based control to a simplified variant of the problem, where the magnetic field F_B has a constant magnitude and direction. Let's assume that that the field always exerts a force on the projectile in the negative x direction. Figure 5.8 shows several examples of the simulated projectile motion when $F_B = 0.1000\,\mathrm{N}$ and the simulation step size is $0.1000\,\mathrm{s}$.

In this simple scenario, it turns out that the relationship between θ_0 and x_f is linear. If we train a mixture model with several $\begin{bmatrix}\theta_0 & x_f\end{bmatrix}$ training examples, we end up with a mixture model like the one shown in Figure 5.9. With such a simple model, predicting an appropriate θ_0 value to achieve a desired x_f value works quite well. For example, in Figure 5.10, GMR was used to predict the θ_0 value that would result in $x_f = 0$ (i. e. the final projectile location being $(0, 10)$. The predicted value was $\theta_0 = 2.725°$ and the resulting final-x position of the projectile was $x_f = -2.578 \times 10^{-4}$.

5.4.2.3 Complex Case: Bidirectional Magnetic Field

Though traditional methods of regression (in addition to GMM based regression) will perform well on the constant-field scenario just described, they fail when faced with more complex scenarios. Let us now assume that instead of having a magnetic field which has a constant direction and magnitude, it has a direction which changes from time to time. In particular, the direction is held fixed for the duration of any particular run of the projectile simulation, however it can change from one run to the next. Furthermore, we will assume that there are only two possible directions that the field can have, either in the negative x direction, or in the positive x direction, and that the magnitude of these two fields is the same ($F_B = 0.1000\,\mathrm{N}$). Lastly, let us assume that the field which pushes the projectile to *the left* (i. e. in the negative x direction) is present 70% of the time, while the other field is present 30% of the time.

Figure 5.11 shows a GMM which was learned from $\begin{bmatrix}\theta_0 & x_f\end{bmatrix}$ observations in this new bidirectional scenario. The difference between this model and the model in Figure 5.9 is that in the bidirectional case we end up with a conditional model having two modes

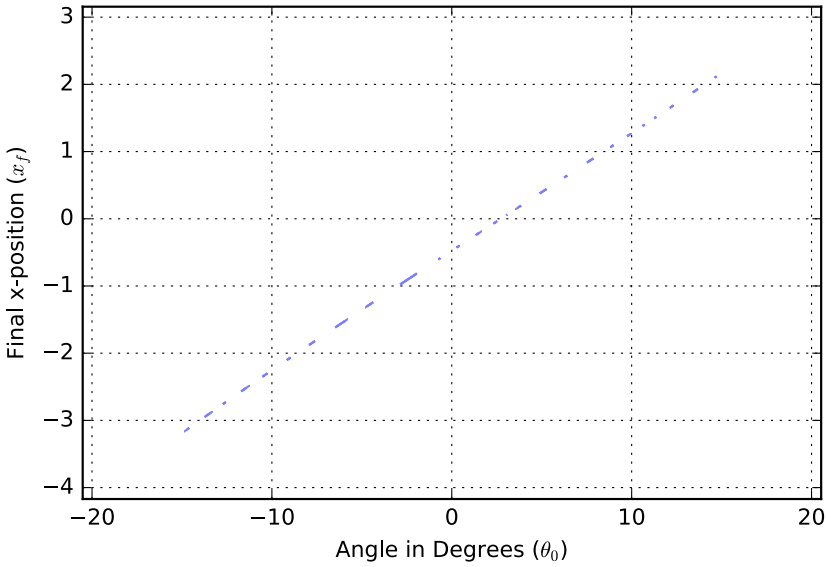

Figure 5.9: Mixture model resulting from applying DGME using $\begin{bmatrix}\theta_0 & x_f\end{bmatrix}$ training examples when the magnetic field is set to a constant value of $F_B = 0.1000\,\mathrm{N}$ and only pushes the projectile to the left. Because the ellipses are extremely narrow (nearly line-segments), they have been enlarged by a factor of 2 to improve their visibility.

instead of one (see Figure 5.12 for a depiction of this conditional model). If we use GMR naively with the bidirectional model and attempt to achieve a target value of $x_f = 0$, the initial angle predicted in this case is $\theta_0 = 1.076°$ which results in the trajectory shown in Figure 5.13. This figure shows the results of using this value of θ_0 for both the left-pulling magnetic field (which occurs for approximately 70% of the simulation-runs), and the right-pulling magnetic field (which occurs for approximately 30% of the simulation-runs). The left and right-pulling cases result in x_f values of -0.2873 and 0.6631, respectively. These results are relatively poor due to the invalid assumption that the model we are using for regression is unimodal. Most other regression methods (which also make this invalid assumption) will exhibit the same poor result.

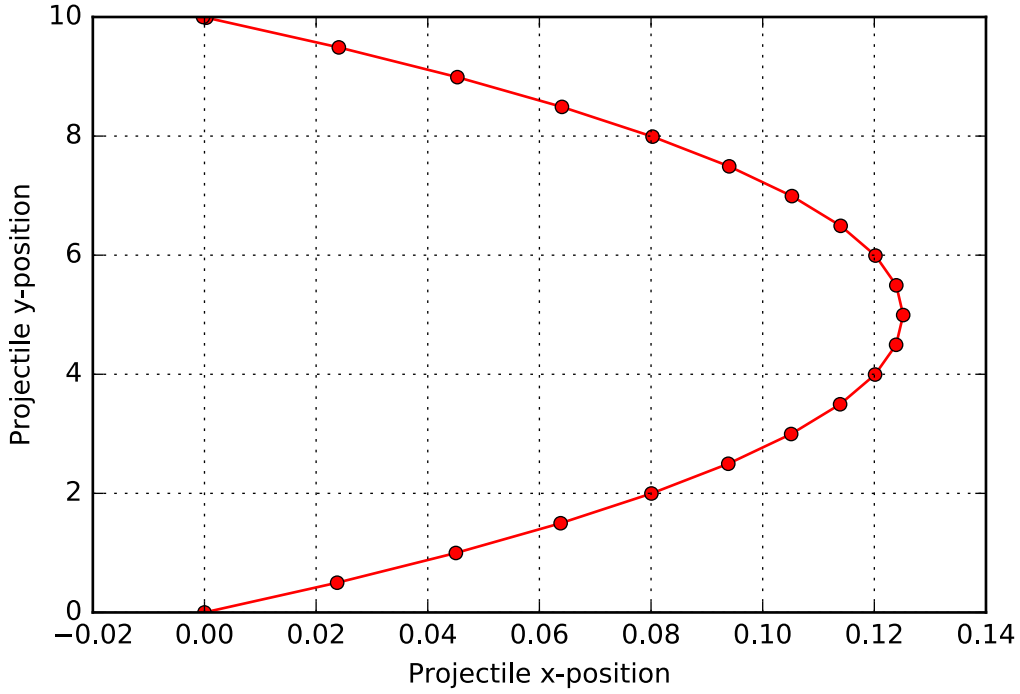

Figure 5.10: Result of using GMR with the model in Figure 5.9 to control the projectile in a unidirectional magnetic field. The desired target value is $x_f = 0$, and the predicted control value is $\theta_0 = 2.725°$.

5.4.2.4 Properly Handling Multimodality: Gaussian Mixture Control

To overcome the problems that occur when assuming a model is unimodal despite it actually being multimodal, a new method is proposed here called **Gaussian Mixture Control (GMC)**. The basic idea is that instead of blindly choosing the expected value of a conditional distribution, each mode of the conditional distribution is evaluated in terms of its suitability for leading to a desired outcome. The mode that results in the most favorable outcome is chosen and *its* expected value is used instead of the expected value of the full conditional distribution. This may be better explained by observing how the method is applied in the scenario mentioned in Section 5.4.2.3.

To explain how GMC is implemented, we will first introduce a bit of notation: a Gaussian component (i. e. consisting of the parameters $\begin{bmatrix} index & w & \boldsymbol{\mu} & \boldsymbol{\Sigma} \end{bmatrix}$) having the index b from model A is represented with the notation G_b^A. Similarly, w_b^A, $\boldsymbol{\mu}_b^A$, and $\boldsymbol{\Sigma}_b^A$ represent the

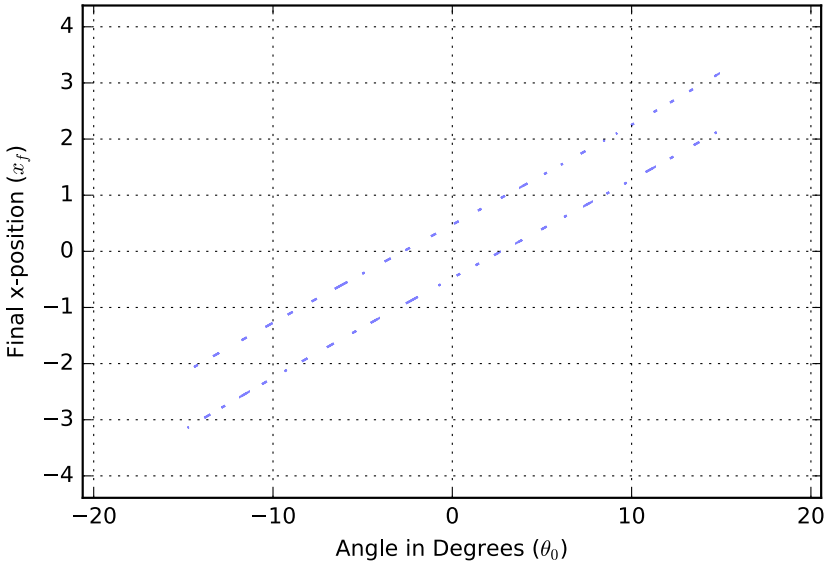

Figure 5.11: Mixture model resulting from applying DGME using $\begin{bmatrix}\theta_0 & x_f\end{bmatrix}$ training examples when the magnetic field is set to a constant value of $F_B = 0.1000\,\text{N}$ and only pushes the projectile to the left 70% of the time, and to the right 30% of the time. Because the ellipses are extremely narrow (nearly line-segments), they have been enlarged by a factor of 2 to improve their visibility.

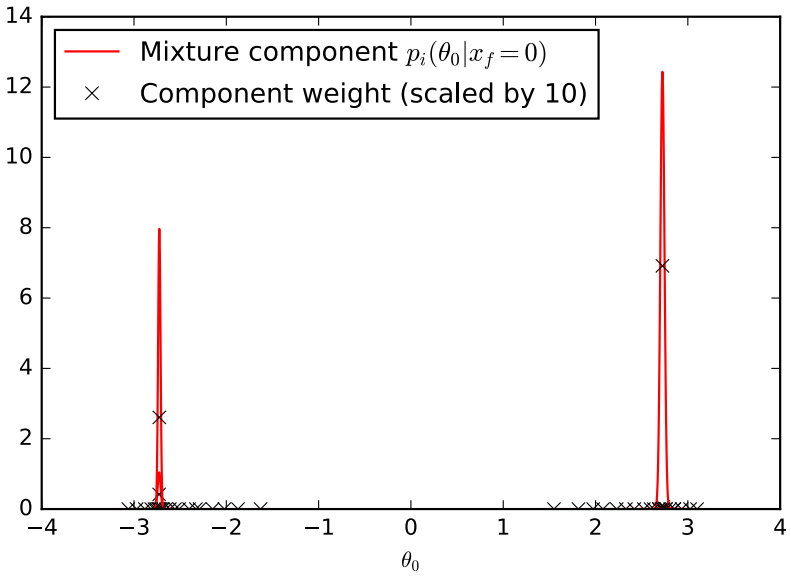

Figure 5.12: Conditional model derived from the model in Figure 5.11, conditioned on the final position being set to $x_f = 0$. Depicted here are the individual components of the conditional mixture model, scaled by their mixture weights, along with an indication of the actual weight of each component. Notice that there are only two components having larger weights, and these represent the two θ_0 values that lead to $x_f = 0$.

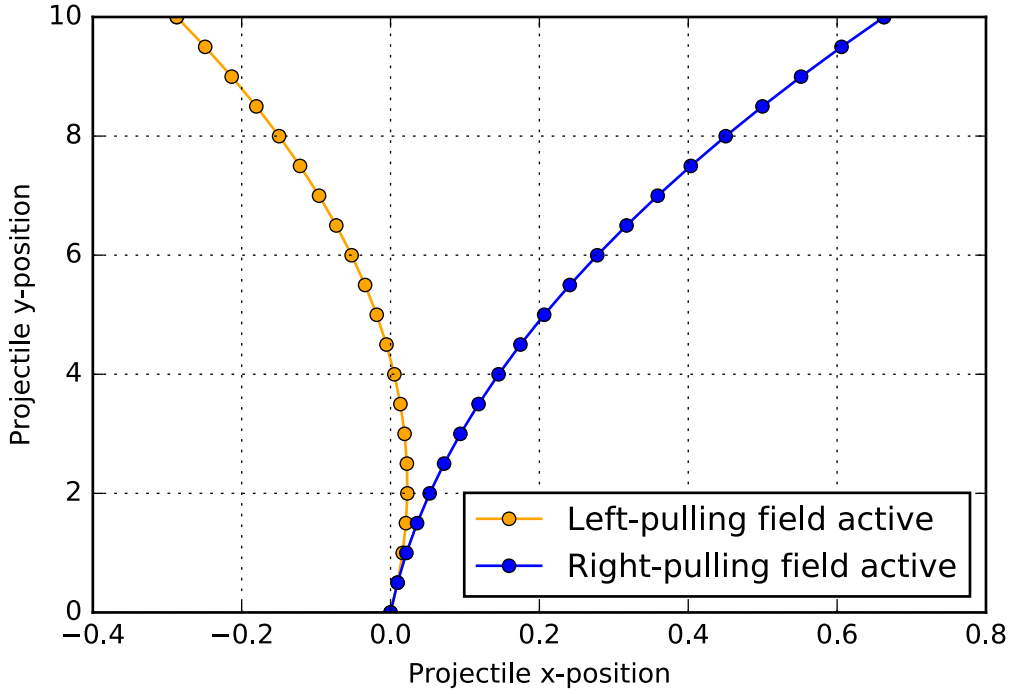

Figure 5.13: Results of naively applying GMR to predict the control value θ_0 in an attempt to achieve a target value $x_f = 0$. The poor prediction results are due to the incorrect assumption that the model used for prediction (shown in Figure 5.12) is unimodal.

weight, mean, and covariance of the component G_b^A, respectively. A Gaussian component not necessarily connected with a particular model can be simply represented as G_b, where b is some index that is used to identify the component. Note that the index used doesn't imply anything about the size of a model – it is possible, for example, for a model to be comprised only of a single component having an index value of 50.

Gaussian Mixture Control consists of the following basic steps:

1. Compute the *inverse conditional model* V from the mixture distribution representing the joint pdf. The *inverse model* is a conditional model where the variables that are conditioned upon are the *outputs* to the system under control, and the resulting distribution is defined over the *input* variables. In the projectile problem described in Section 5.4.2.3, the *output* is the final x position of the projectile, and the

input is the initial angle. Therefore, in this case we compute $p(\theta_0 \mid x_f = 0)$, depicted in Figure 5.12, from the pdf $p(x_f, \theta_0)$.

2. Each "high-weight" component in V is used as a basis to construct one of several *forward conditional models* F_t. A user-selected component-weight threshold is used to select all of the high-weight components in V. For every component G_t^V selected from model V, the mean $\boldsymbol{\mu}_t^V$ will be used to construct a forward conditional model, F_t. This forward model will be evaluated for its likelihood of achieving the desired result. Note that F_t has the same number of components as V, and that each component $G_j^{F_t}$ is associated with G_j^V. If we use the same bidirectional magnetic-field scenario described in Section 5.4.2.3, and set the normalized component-weight threshold to 0.05, then we are left with two components, pictured in Figure 5.14. The forward conditional models associated with these two components are shown in Figures 5.15 and 5.16.

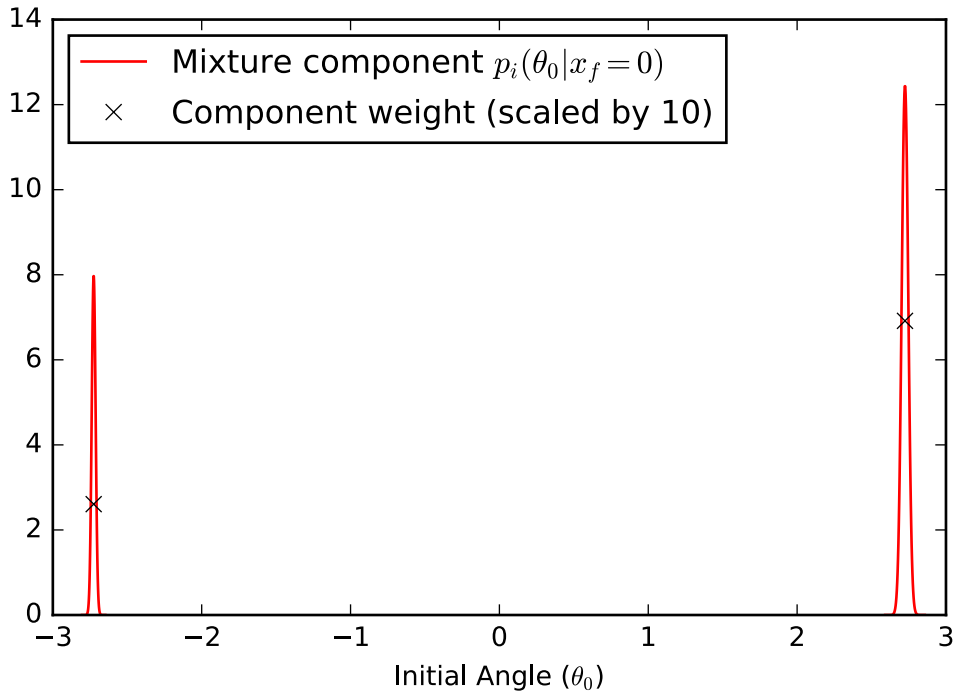

Figure 5.14: Mixture components of model V whose normalized weights exceed 0.05. These components have (μ_θ, w) values of $(-2.726, 0.2607)$ and $(2.725, 0.6916)$.

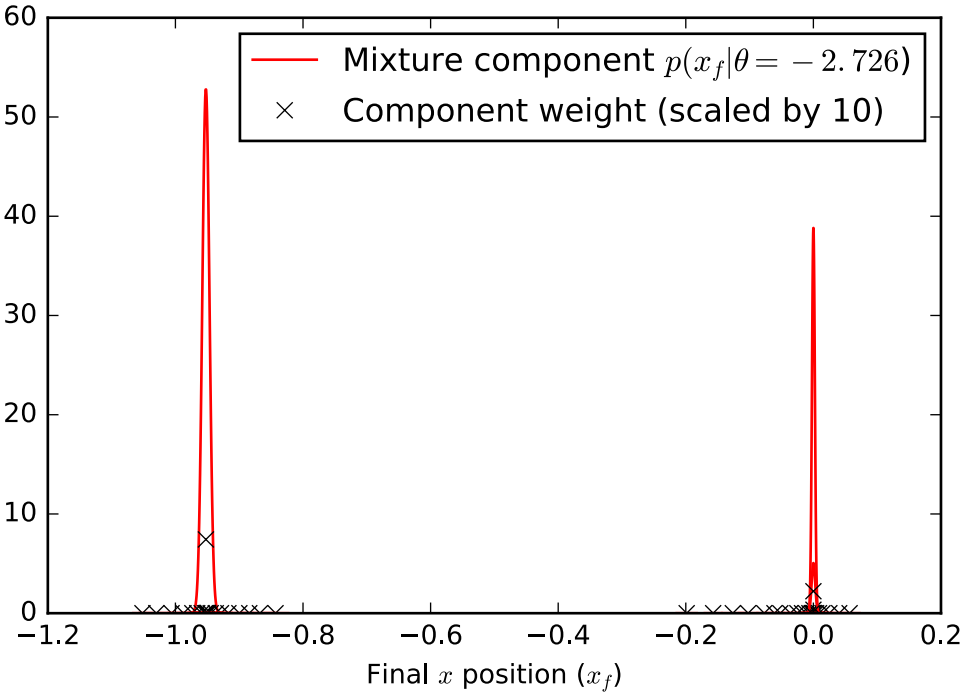

Figure 5.15: The forward model $p(x_f \mid \theta_0 = -2.726)$.

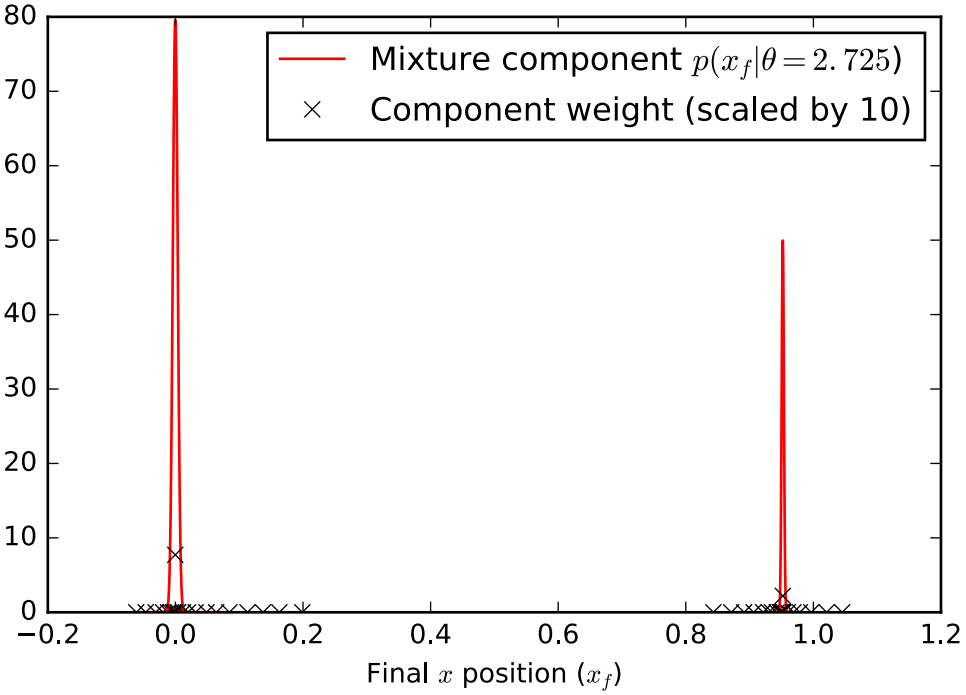

Figure 5.16: The forward model $p(x_f \mid \theta_0 = 2.725)$.

3. Smooth each forward conditional model. How do we determine from a particular forward conditional model F_t whether the mean $\boldsymbol{\mu}_t^V$ associated with this model is likely to achieve the desired result (i. e. the desired output value)? We must first *smooth* F_t. This means merging together all the clusters of components in F_t that are "near" each other (determined by some user-defined distance metric[11] and distance threshold d_{th}). This smoothing operation results in yet another model, S_t, where $G_q^{S_t}$ is the component in S_t into which $G_t^{F_t}$ was merged. Notice how each index t of the components selected from V in step 2 is linked to an index q of a component in model S_t. The pseudocode for the algorithm used to smooth the model is shown in Algorithm 8.

4. Evaluate each smoothed model. In order to evaluate a particular smoothed model S_t, a loss metric L_t is defined as follows:

$$L_t = \frac{\sum_{k \neq q} w_k^{S_t}}{w_q^{S_t}} \tag{5.4.1}$$

This metric essentially tells us how likely it is that something *other* than the desired output value will result from choosing $\boldsymbol{\mu}_t^V$ as our input value. If we have N smoothed models, then we will compute N corresponding loss metric values. After we have loss values for each S_t, we can determine the value of t which specifies the high-weight component from V who, along with its neighboring mixture components, is most likely to result in the best input value. The best t value is given as:

$$t^* = \operatorname*{argmin}_k L_k \,. \tag{5.4.2}$$

Of course it may be that there is no good choice available for an input value, which would be the case if $\min_k L_k > C$ where C is some chosen maximum loss value that is deemed tolerable.

5. Choose the best input value. After the best model, S_{t^*}, has been chosen in step 4, we use the set of indices T computed in `SMOOTH` in connection with this model in order to select an appropriate subset of the components in V. This subset forms a new inverse

[11] In the examples given in this dissertation, the distance metric used is the Euclidian distance between the means of the components.

Algorithm 8: Pseudocode for `SMOOTH` Algorithm

input : M – unsmoothed model with components $\{G_1, G_2, \cdots, G_N\}$
t – index of component in M to be tracked
d_{th} – distance threshold
output : S – smoothed model
T – the indices of components in M merged with G_t^M
q – index of component in S that G_t^M was merged into

Function `SMOOTH`(M, t, d_{th})

```
N ← ‖M‖;
Let E = {e_ij | 1 ≤ i < j ≤ N};          // e_ij is the distance from G_i to G_j
N_orig ← N;
R ← {};                                  // indices of components removed from model
T ← {t};
q ← t;
while min(E) > d_th do
    N ← N + 1;                           // index of new mixture component
    e_ij ← min(E);
    E ← E \ {e_ij};
    R ← R ∪ {i, j};
    G_N ← component resulting from merging G_i and G_j;
    if i == t and j ≤ N_orig then
        q ← N;
        T ← T ∪ {j};
    end
    if j == t and i ≤ N_orig then
        q ← N;
        T ← T ∪ {i};
    end
    M ← M ∪ {G_N};                       // add new component to M
    E ← E ∪ {e_kN | k ∈ {1, ··· , N − 1} \ R};   // add new distances to E
    E ← E \ {e_pq | p ∈ R or q ∈ R};     // remove old distances from E
end
S ← M \ {G_k | k ∈ R};
return S, T, q;
end
```

model $V_{GMC} = \{G_i^V \mid i \in T\}$ (shown in Figure 5.17) that will be used instead of V to predict a desired input value (i. e. θ_0) using GMR.

1. Results of Applying GMC to Bidirectional Magnetic Field Problem The filtered inverse model V_{GMC} predicts a desired initial angle of $\theta_0 = 2.725°$. The resulting performance of the simulation with this predicted θ_0 value is shown in Figure 5.18. The result is good for the predominantly occurring magnetic-field state (left-pushing), while it is understandably worse for the right-pushing magnetic field. A key observation is that most of the time the target result of $x_f = 0$ is reached within a very tight tolerance, though occasionally it is not reached. In contrast to this, the case where we predicted θ_0 using naive GMR under the assumption of a unimodal model ends up with the target result *never* being very close to the desired x_f value. This illuminates an important consideration: **GMC is important to use in cases when *extreme accuracy* some or most of the time is far more important than *rough accuracy* all of the time.**

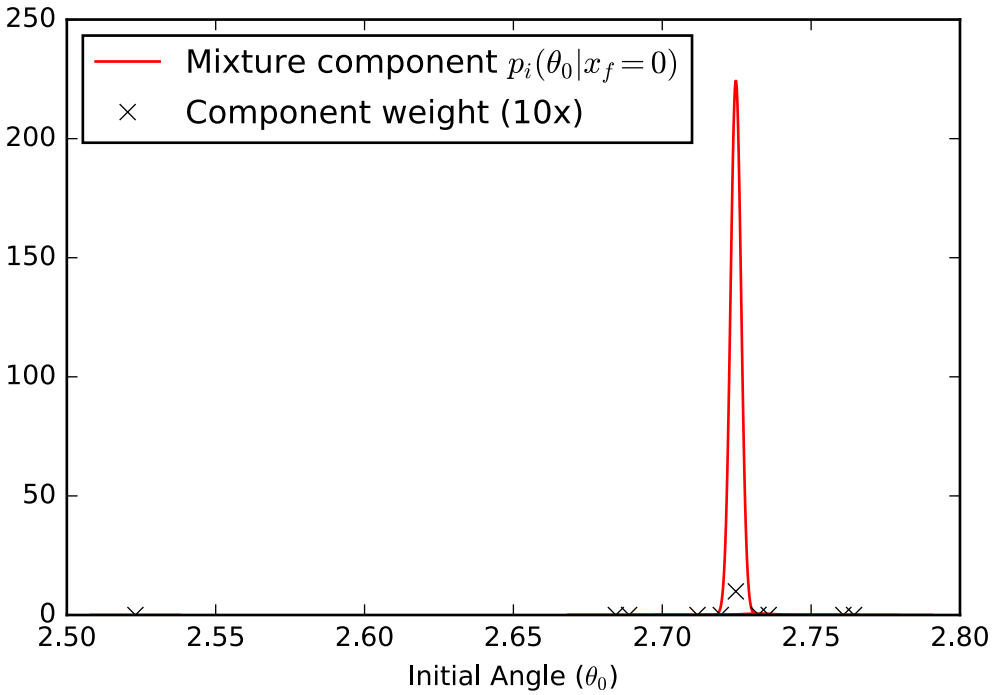

Figure 5.17: Filtered inverse model V_{GMC}, which possesses the components that are best fit for predicting the value of θ_0 which will lead to a resulting target value of $x_f = 0$.

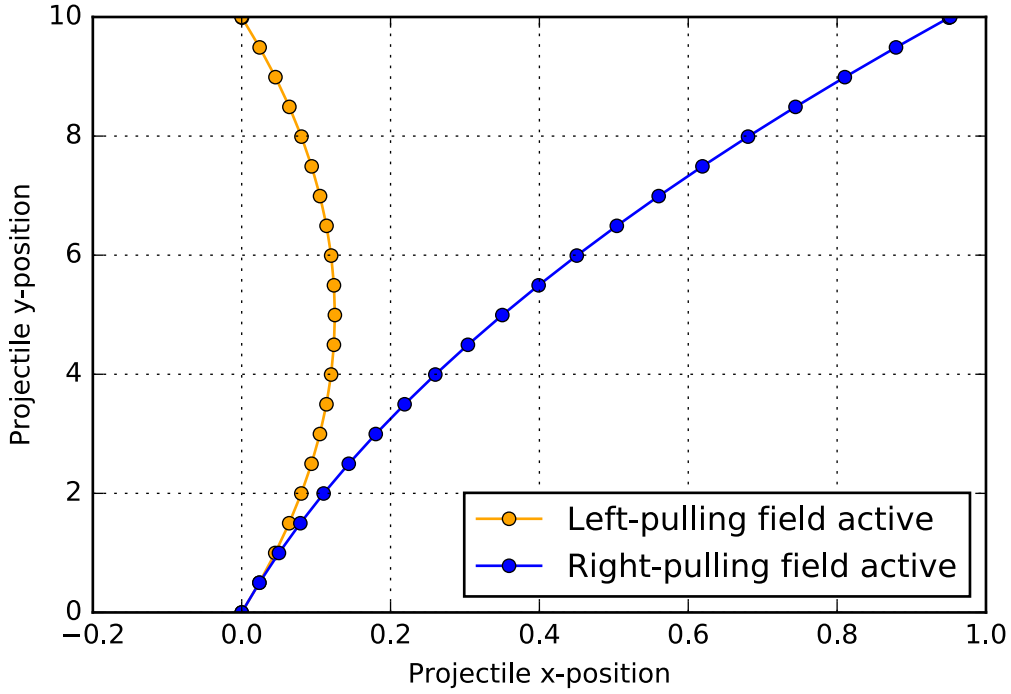

Figure 5.18: Result of applying V_{GMC} in Figure 5.17 for predicting θ_0. The predicted best θ_0 value was 2.725°, and the resulting target value was $x_f = -0.000\,209\,2$ under a left-pushing magnetic field. The right-pushing field gives a worse x_f result, as expected, but occurs less frequently than the left-pushing field.

The results of using GMC and the naive prediction methods are summarized in Table 5.2. When using GMC method for control, the mean absolute deviation of the actual target value from the desired target value is lower than when using the naive GMR method.

Magnetic Field*	Technique	Predicted θ_0	x_f (left)	x_f (right)	MAD†
100% Left	Naive GMR	2.725°	-2.578×10^{-4}	N/A	2.578×10^{-4}
70% Left, 30% Right	Naive GMR	1.076°	-2.873×10^{-1}	6.631×10^{-1}	3.978×10^{-1}
70% Left, 30% Right	GMC	2.725°	-2.092×10^{-4}	9.521×10^{-1}	2.801×10^{-1}

* This column specifies the probabilities that a given simulation run will have a particular kind of magnetic field.

† Mean Absolute Deviation: $\frac{1}{N}\sum_{i=1}^{N} \mid x_f^{(i)} - 0 \mid$

Table 5.2: Summary of Results Comparing GMC with Naive Methods.

Chapter 6

Conclusions and Future Directions

This dissertation has presented two significant contributions to the scientific community in the areas of density estimation and control. The first, Dynamic Gaussian Mixture Estimation (DGME), enables one to flexibly model online streams of observations so that prediction, classification, and inference can be performed on the estimated models. The second, Gaussian Mixture Control (GMC), complements DGME, and provides a means of controlling processes that exhibit multimodal behavior. Though these methods are valuable contributions in their own right, there are still many areas to be explored, and ways to enhance their functionality. Two general areas worth exploring in the future are included here: handling non-stationarity in modelled processes, and deep mixture-model architectures.

6.1 Extending DGME to Handle Non-stationary Processes

Although Chapter 3 explained the standard DGME algorithm in detail, there are various ways that the algorithm can be extended. This section introduces ideas that suggest a potential way of extending DGME to enable it to better handle the modelling of processes that are quasi-stationary or non-stationary.

6.1.1 Suspending Judgment: Dual-model DGME

The simple rule of using likelihood to determine the "belongingness" of an observation to a model works well when the system being modelled is stationary (i. e. the relationships between the system variables that are being modelled do not change with respect to time). When the system exhibits *non-stationary* behavior, however, using the likelihood alone can lead to problems.

For example, assume that a process that is being modelled exhibits one kind of behavior initially, and this behavior continues for some time. After some time, the model becomes "established" or "entrenched" – the Gaussians which exist in the model will have a very high probability mass in comparison with new observations. This is not a problem, *unless the behavior of the process changes at some later point in time.*

When there is a lot of "pre-existing" weight (probability mass) in the model, then a novel-observation will result in the generation of a new component that has a comparatively small probability mass with respect to the rest of the model's components. As a result, if further novel observations happen to be encountered in the neighborhood of this new component, they will result in the generation of additional components instead of being merged with the already-existing low-mass component. This can lead to problems due to not having enough data to adequately estimate each component's covariance matrix (see Section 4.2.4).

The standard *single-model* version of DGME suffers from the problem described above. However, a variant of DGME called *Dual-model DGME* handles this problem by building *two* models in parallel, a "primary" model M_1 which represents the "status quo" knowledge that has been collected thus far, and a "secondary" model M_2 which keeps track of novel observations. As new observations are made available to the DGME algorithm, they will be evaluated in terms of how suitable they are to be merged into existing components of M_1. If they are found to be novel with respect to M_1, then they will be incorporated into M_2 instead of being added as new components in M_1. These two models are regularly compared with each other, and for each component of M_2, a decision is made regarding whether to:

1. do nothing (leave the component in M_2)

2. merge the component from M_2 into a component of M_1 (removing it from M_2).

3. add the component from M_2 as a new component of M_1 (removing it from M_2).

6.1.2 Quasi-stationary and Non-stationary Processes

One difficulty faced by many modelling techniques (including nearly all batch estimation techniques) has to do with the common assumption that a modelled process is *stationary* – that its behavior follows a pattern that is independent with respect to absolute time[1]. This poses a serious problem because it significantly restricts the set of processes which can be modelled. There likely exist as many or more non-stationary processes in "the real world" than there are stationary processes. Processes can be further classified as *quasi-stationary*, which indicates that a process's behavior lies somewhere between stationary and non-stationary.

When using single-model[2] DGME, the primary difference between quasi-stationary and non-stationary processes has to do with the extent to which new components will be created as the model is being built. Observations from quasi-stationary processes will tend to get merged into existing components, whereas the observations from a non-stationary process will tend to create new components, distinct from the components representing a system's "older" behavior.

6.1.2.1 Handling quasi-stationary processes

In Section 6.1.1, the concept of parallel models was introduced. Besides solving the problem of being able to summarize novel observations when a model possesses much "pre-existing probability mass", splitting the model into primary and secondary sub-models also solves a problem related to quasi-stationary processes[3]. In particular, it prevents observations that are only slightly-novel with respect to the primary model

[1] This can be confusing when one considers a process whose observation vectors include time-durations. The key to understanding the concept here is that a process may depend on relative time *differences*, but cannot change its joint pdf (i. e. the "true" components of the GMM) as a function of time.

[2] See Section 6.1.1 for the distinction between dual-model and single-model DGME.

[3] A quasi-stationary process can be roughly defined as a process whose behavior changes slow with respect to the speed at which a model of the process can be updated.

(M_1) from getting immediately absorbed into an existing Gaussian component (and possibly corrupting / compromising its parameters). Having a secondary model allows these observations to be "quarantined" for a certain period of time until there's enough evidence to decide whether they should indeed be merged into the existing component, or whether they are more likely to belong to one or more other components.

To illustrate this problem in more detail, see Figure 6.1. In this figure, the upper plot shows a sequence of one-dimensional observations made over time. If only one model is used to capture the observed data, the modelled system changes so slowly that observations continue to get merged into an existing Gaussian. By the time t_1 is reached, the Gaussian represents all of the observed data starting from time t_0. This may be desirable if the observation values will continue to fluctuate between x_0 and x_1, but if the observation values have simply made a one-time gradual shift from around x_0 to around x_1, then we need a way for the model to "let go" of the observations nearer to x_0, representing only those which are closer to x_1. By splitting the model into M_1 and M_2 as mentioned above, this can be achieved: several components of the model will be generated, and the relative importance of each component can be adjusted over time based on the amount of time since it was last seen to support observations (this is discussed in more detail in the following section on non-stationary processes).

6.1.2.2 Dreaming: synthesizing information from non-stationary processes

It is hypothesized that one function of dreaming is to "connect the dots" between different observations and memories one has stored in the brain. If this is true, then dreaming may also serve to enhance new knowledge with old knowledge. For example, one may need to observe several related events in order to come to a more concrete understanding about a causal process. However, there is no guarantee that one will observe all of the necessary events at the same time, or even in close temporal proximity. In fact, it may happen (for a person) that something which was observed only once several years ago plays a significant role in forming a hypothesis when combined with present observations. How does this process work? It is not likely that the person will have the "old" observation(s) readily available at all times for immediate reasoning about things. Similarly, it is a common human experience to try to remember someone's name, or how a particular event from the past transpired, and at first to not be able to recall this information.

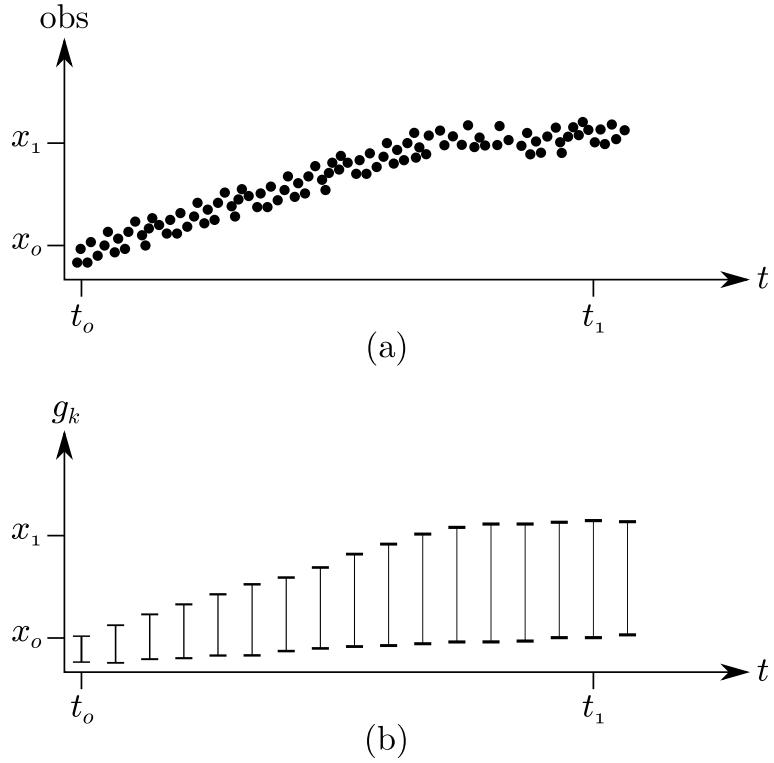

Figure 6.1: Illustration of component-corruption problem due to quasi-stationarity of a modelled process: (a) depicts a one dimensional observation of a quasi-stationary process plotted over time, and (b) depicts the variance of a Gaussian component (the mean is located at the center of these variance markers) over time. The component can end up representing all of the observations, because at any point in time an observation is only "quasi-novel" with respect to that component.

After some time, however, the information is usually able to be recalled.

Without making any claims on the actual physical mechanism that takes place within the human brain when trying to recall information from long ago, DGME *can* be used to approximate this human ability. In particular, each Gaussian component in a model can be given an "age" attribute Δt, which represents the amount of time elapsed since an observation was merged into that component (affecting the component's parameters). This age is used to compute an *effective* (unnormalized) weight $\widetilde{w}_k$ of the k^{th} component, where the weight decays exponentially with a time constant τ_d:

$$\widetilde{w}_i = w_i e^{-\Delta t/\tau_d} \tag{6.1.1}$$

The τ_d parameter can be either hand-selected, or learned by observing some "reward" metric of the model's effectiveness over time (e. g. in terms of its ability to provide accurate predictions), and trying to maximize a weighted average of this reward.

By having an effective weight that is based on a component's "vibrancy", it is possible to restrict the "knowledge" (i. e. components of the model) used when making predictions to only the knowledge supported by more recent observations. This might be akin to when a person is only able to make use of more recent (or regularly accessed) memories when consciously reasoning about a situation. When one dreams, however, it may be that one's "old" knowledge is fused with recently used and acquired knowledge in order to enhance or complete it. In the terms of the DGME paradigm, this would be accomplished by comparing each *active* (i. e. "recent") Gaussian with the *dormant* (i. e. "old") Gaussians. If strong correlations are found between any dormant Gaussian and an active Gaussian, then the dormant Gaussian can be eliminated and merged into the active Gaussian, altering the active Gaussian's parameters to reflect the combination of both Gaussians.

The beauty of this paradigm is that despite DGME being an online algorithm, when effective weights are used it doesn't need to discard "old" knowledge, and at the same time it does not require that this "old" knowledge be processed when the model is "awake" (i. e. in the midst of incorporating new observations into the model). In fact, the sleep-like "unconscious" process of fusing dormant and active knowledge can be executed in parallel with the "conscious" process of assimilating new observations into shorter-term (i. e. active) memory[4]. Algorithms like `SMOOTH` (see Algorithm 8) could be adapted for this purpose.

[4]This kind of memory is sometimes referred to as "working memory" in the psychology and neuroscience literature.

6.2 Extending DGME to Work with Deep Architectures

Another area worth future exploration related to DGME is the area of "deep architectures". The contributions in this dissertation have been restricted to working with representations that are so-called "shallow architectures". If a given representational architecture (e. g. that is used for inference, etc.) is cast in the form of a graph connecting inputs (conditions) to outputs (predictions), *shallow* architectures are those for which the maximum number of edges connecting an input to any output (i. e. the *depth*) is relatively low (e. g. less than 4). Bengio (2009) have argued, however, that in comparison with *deep* architectures, such architectures require a significantly (i. e. exponentially) higher number of processing components to represent certain complex processes. Furthermore, Serre et al. (2007) propose an architecture based on the anatomy of the human cortex which appears structurally similar to a deep architecture (the proposed architecture has a depth greater than 8).

Because of the mathematical ease which which GMMs can be used, introducing a hierarchical version of the GMM representation and the associated estimation and exploitation methods presented in this thesis could be a relatively straightforward task. Since the primary operations one needs to be able to perform on a GMM are conditionalization, marginalization, and expectation, these operations need to be generalized to a hierarchy of GMMs, where the "output components" of one GMM link to the "input components" of a higher-level GMM. Furthermore, the DGME algorithm would need to be expanded to not only decide between Gaussian creation / merging (for a single level), but also to decide when a new hierarchical layer is appropriate. This would be one of the more challenging aspects of extending the work of this thesis to a hierarchical representation.

6.3 Final Remarks

This dissertation has provided a novel and promising method, **Dynamic Gaussian Mixture Estimation**, for building density estimates that describe probabilistic systems. It provides a unique combination of features including the following:

- the number of mixture components used in the density estimate model are adapted to fit the process being modelled
- the assumption that prior data can be stored and retrieved is not required – the method operates in an *online* manner
- the method can be extended to permit modelling non/quasi-stationary processes

Furthermore, a novel mixture-model based control method **Gaussian Mixture Control** has been presented that works hand-in-hand with models estimated via DGME. This control method fills a gap in the world of control methods, in that it allows some level of control over systems exhibiting uncertainty and multimodality. Systems that have been deemed uncontrollable using traditional control methods *can* be controlled with GMC, where lower-variance in control output is traded for increased frequency of accurate control of a process' state.

Appendix A

Derivation of Point-to-Gaussian Merge Equations

By simply substituting $\mu_2 = x$ and $M = 1$ into (3.7.6), it is trivial to derive the point-to-Gaussian merge equation (3.7.16):

$$\mu_3 = \frac{1}{N+1}(N\mu_1 + x) \tag{A.0.1}$$

Here, μ_3 is the new mean that results from merging x into a Gaussian having a mean μ_1, and an unnormalized weight of N (i. e. the number of observations that have already contributed to the estimate of μ_1).

Derivation of the second merge equation, (3.7.17) requires a bit more algebraic manipulation. We begin with (3.7.12):

$$\sigma_3^2 = \frac{1}{N+M-1}\left\{(N-1)\sigma_1^2 + N(\mu_3-\mu_1)^2 + (M-1)\sigma_2^2 + M(\mu_3-\mu_2)^2\right\}$$

Into this, we again substitute $\mu_2 = x$ and $M = 1$, and proceed to derive the result:

$$\sigma_3^2 = \frac{1}{N+M-1}\left\{(N-1)\sigma_1^2 + N(\mu_3-\mu_1)^2 + (M-1)\sigma_2^2 + M(\mu_3-\mu_2)^2\right\} \tag{A.0.2}$$

$$= \frac{1}{N}\left\{(N-1)\sigma_1^2 + N(\mu_3-\mu_1)^2 + (\mu_3-x)^2\right\} \tag{A.0.3}$$

Equation (A.0.1) can be substituted into the quantity $(\mu_3 - \mu_1)$ as follows:

$$\mu_3 - \mu_1 = \frac{1}{N+1}(N\mu_1 + x) - \mu_1 \tag{A.0.4}$$

$$= \frac{N}{N+1}\mu_1 + \frac{1}{N+1}x - \frac{N+1}{N+1}\mu_1 \tag{A.0.5}$$

$$= \frac{-1}{N+1}\mu_1 + \frac{1}{N+1}x \tag{A.0.6}$$

$$= \frac{1}{N+1}(x-\mu_1). \tag{A.0.7}$$

Likewise, for $(\mu_3 - x)$, we have:

$$\mu_3 - x = \frac{1}{N+1}(N\mu_1 + x) - x \tag{A.0.8}$$

$$= \frac{N}{N+1}\mu_1 + \frac{1}{N+1}x - \frac{N+1}{N+1}x \tag{A.0.9}$$

$$= \frac{N}{N+1}\mu_1 - \frac{N}{N+1}x \tag{A.0.10}$$

$$= \frac{N}{N+1}(\mu_1 - x). \tag{A.0.11}$$

We can now substitute (A.0.7) and (A.0.11) into (A.0.3), and continue the derivation:

$$\sigma_3^2 = \frac{1}{N}\left\{(N-1)\sigma_1^2 + N(\mu_3-\mu_1)^2 + (\mu_3-x)^2\right\} \tag{A.0.12}$$

$$= \frac{1}{N}\left\{(N-1)\sigma_1^2 + N\left(\frac{1}{N+1}(x-\mu_1)\right)^2 + \left(\frac{N}{N+1}(\mu_1-x)\right)^2\right\} \tag{A.0.13}$$

$$= \frac{N-1}{N}\sigma_1^2 + \left(\frac{1}{N+1}(x-\mu_1)\right)^2 + \frac{1}{N}\left(\frac{N}{N+1}(\mu_1-x)\right)^2 \tag{A.0.14}$$

$$= \frac{N-1}{N}\sigma_1^2 + \left(\frac{1}{N+1}\right)^2 \underbrace{(x-\mu_1)^2}_{=(\mu_1-x)^2} + N\left(\frac{1}{N+1}\right)^2 (\mu_1-x)^2 \tag{A.0.15}$$

$$= \frac{N-1}{N}\sigma_1^2 + (N+1)\left(\frac{1}{N+1}\right)^2 (\mu_1-x)^2 \tag{A.0.16}$$

$$= \frac{N-1}{N}\sigma_1^2 + \left(\frac{1}{N+1}\right)(\mu_1-x)^2 \tag{A.0.17}$$

This derivation method is also valid for the multidimensional case, where the same trick can be used to go from (A.0.15) to (A.0.16), because the inner product matrix $(\mathbf{x}-\boldsymbol{\mu}_1)(\mathbf{x}-\boldsymbol{\mu}_1)^T$ will be the same as $(\boldsymbol{\mu}_1-\mathbf{x})(\boldsymbol{\mu}_1-\mathbf{x})^T$.

Appendix B

Rotation Invariance of Mahalanobis Distance

Introduction

The Mahalanobis distance is the multidimensional generalization of the 1D *standard-score* $z = (x - \mu)/\sigma$. In contrast to the standard-score, with the Mahalanobis distance, $\mathbf{x}$ and $\boldsymbol{\mu}$ are multidimensional (column vectors). The intuition behind the Mahalanobis distance is that the Euclidean distance between $\mathbf{x}$ and $\boldsymbol{\mu}$ is scaled in each dimension according to the values of a positive definite matrix $\boldsymbol{\Sigma}$, which can be thought of in terms of eigenvalues and eigenvectors as a set of σ values, one for each eigenvector. Each component of $x - \mu$ is scaled according to how much it projects onto each of the eigenvectors.

Claim and Demonstration

The intent of the following is to show one thing: that the Mahalanobis distance of a point with respect to a covariance matrix is the same, regardless of whether the coordinate system of the data-points has undergone a unitary transformation (e. g. rotation, reflection, etc.).

The Mahalanobis distance from a point $\mathbf{x}$ to a Gaussian $g \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is defined as:

$$d_M(\boldsymbol{\Sigma}, \mathbf{r_1}) = \sqrt{\mathbf{r_1}^T \boldsymbol{\Sigma}^{-1} \mathbf{r_1}} \tag{B.0.1}$$

where $\mathbf{r_1} = \mathbf{x} - \boldsymbol{\mu}$.

Furthermore, "rotating" (i. e. transforming) a covariance matrix Σ involves the following steps:

1. Perform the eigendecomposition $\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$, where $\mathbf{Q}$ is a matrix containing column-eigenvectors, and $\boldsymbol{\Lambda}$ is a diagonal matrix whose $\boldsymbol{\Lambda}_{(i,i)}$ element is the eigenvalue corresponding to the eigenvector $\mathbf{Q}_{(\cdot,i)}$.

2. Transform each eigenvector in $\mathbf{Q}$ according to some unitary transform (i. e. "rotation") matrix $\mathbf{T}$. This results in a new set of (rotated) column basis-vectors $\mathbf{Q}_2 = \mathbf{TQ}$

3. The new (transformed) covariance matrix is given by $\boldsymbol{\Sigma_2} = \mathbf{Q_2}\boldsymbol{\Lambda}\mathbf{Q_2}^{-1}$

Now, if we also transform $\mathbf{r_1}$ using $\mathbf{T}$: $\mathbf{r_2} = \mathbf{T}\,\mathbf{r_1}$, and calculate the Mahalanobis distance

$$d_M(\boldsymbol{\Sigma_2}, \mathbf{r_2}),$$

we find that this is identical to the value calculated in (B.0.1). A few 2D examples demonstrating this are shown in Figure B.1.

Proof of Rotation Invariance. To prove the equivalence of the rotated Mahalanobis distance d_{M2} and the original distance d_{M1}, it is sufficient to show that

$$M_2 = d_{M2}^2 = d_{M1}^2 = M_1$$

for any unitary linear transformation (e. g. rotations), since the Mahalanobis distance is always nonnegative.

Let us define $\mathbf{Q}_2 = \mathbf{RQ_1}$, and $\mathbf{r}_2 = \mathbf{R}\mathbf{r}_1$. We write the untransformed squared Maha-

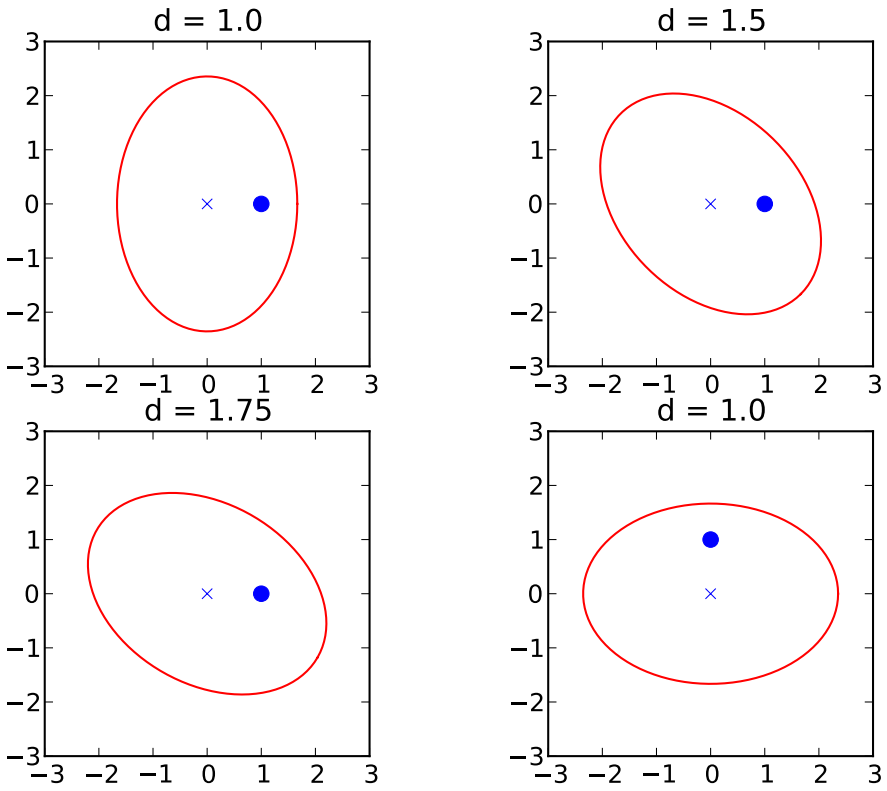

Figure B.1: Point to Gaussian distance examples. Distances (d) are Mahalanobis distances. Ellipses are drawn such that they contain 75% of the probability mass. The query point is indicated by the larger blue dot.

lanobis distance as:

$$\begin{aligned} M_1 &= \mathbf{r}_1^T \boldsymbol{\Sigma}_1^{-1} \mathbf{r}_1 \\ &= \mathbf{r}_1^T \mathbf{Q}_1 \boldsymbol{\Lambda}^{-1} \mathbf{Q}_1^{-1} \mathbf{r}_1, \end{aligned}$$

where $\mathbf{r}_1 = \boldsymbol{\mu}_1 - \mathbf{x}_1$, and proceed to show that the transformed squared Mahalanobis distance (M_2) is equivalent:

$$
\begin{aligned}
M_2 &= \mathbf{r}_2^T \boldsymbol{\Sigma}_2^{-1} \mathbf{r}_2 \\
&= \mathbf{r}_2^T \{\mathbf{Q}_2 \boldsymbol{\Lambda}^{-1} \mathbf{Q}_2^{-1}\} \mathbf{r}_2 \\
&= (\mathbf{R}\mathbf{r}_1)^T \{(\mathbf{R}\mathbf{Q}_1) \boldsymbol{\Lambda}^{-1} (\mathbf{R}\mathbf{Q}_1)^{-1}\} (\mathbf{R}\mathbf{r}_1) \\
&= \mathbf{r}_1^T \mathbf{R}^T \mathbf{R} \mathbf{Q}_1 \boldsymbol{\Lambda}^{-1} \mathbf{Q}_1^{-1} \mathbf{R}^{-1} \mathbf{R} \mathbf{r}_1 \\
&= \mathbf{r}_1^T \mathbf{I} \mathbf{Q}_1 \boldsymbol{\Lambda}^{-1} \mathbf{Q}_1^{-1} \mathbf{I} \mathbf{r}_1 && \text{because } \mathbf{R} \text{ is unitary} \\
&= \mathbf{r}_1^T \mathbf{Q}_1 \boldsymbol{\Lambda}^{-1} \mathbf{Q}_1^{-1} \mathbf{r}_1 \\
&= \mathbf{r}_1^T \boldsymbol{\Sigma}_1^{-1} \mathbf{r}_1 = M_1
\end{aligned}
$$

Thus,

$$d_{M1} = d_{M2}. \qquad \square$$

Therefore, **the Mahalanobis distance is invariant to *unitary*[1] linear coordinate system transformations.**

[1] *Orthogonal* transformations are a subset of unitary transformations, which are restricted to having real numbers. Rotation matrices belong to the set of orthogonal matrices.

Appendix C

Source Code

The Dynamic Gaussian Mixture Estimation (DGME) and Gaussian Mixture Regression (GMR) methods have been incorporated into a fork of the *scikit-learn* machine-learning library (Pedregosa et al., 2011). This source code can be acquired at the following URL: `https://github.com/edgimar/scikit-learn`

List of Tables

List of Algorithms

List of Figures

Bibliography

By the Author

The following is a list of selected publications that the author of this thesis has co-authored:

Edgington, M., J. de Gea, et al. (2008). "Using the Body in Learning to Recognize Objects". In: *Proc. of the Int. Conf. on Intelligent Autonomous Systems (IAS)*. Baden-Baden, Germany: IOS Press, pp. 110–118.

Edgington, M., Y. Kassahun, and F. Kirchner (2009). "Dynamic Motion Modelling for Legged Robots". In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. St. Louis, MO, USA.

Edgington, Mark, Yohannes Kassahun, and Frank Kirchner (2009). "Using Joint Probability Densities for Simultaneous Learning of Forward and Inverse Models". In: *Proceedings of the 2nd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot System. IEEE International Conference on Intelligent Robots and Systems (IROS-09)*. Ed. by Nils T. Siebel; Josef Pauli. St. Louis, Missouri, United States, pp. 19–22.

Kassahun, Yohannes, Mark Edgington, José de Gea Fernández, Elsa Kirchner, et al. (2006). "Using Kinematically Complex Robots for Case Studies in Embodied Cognition". In: *Proceedings of the 9th International Conference on Climbing and Walking Robots (CLAWAR 2006)*. Brussels, Belgium, p. 258.

Kassahun, Yohannes, Mark Edgington, José de Gea Fernández, and Frank Kirchner (2006). "Towards a Multimodal Sensorimotor Coordination-Based Object Recognition

System". In: *IEEE International Conference on Robotics and Biomimetics (ROBIO 2006)*.

Kassahun, Yohannes, Mark Edgington, José de Gea Fernández, and Frank Kirchner (2007). "Exploiting Sensorimotor Coordination for Learning to Recognize Objects". In: *Twentieth International Joint Conference On Artificial Intelligence. IJCAI-07, January 6-12, Hyderabat, India*, pp. 883–888.

Kassahun, Yohannes, Mark Edgington, Jan Hendrik Metzen, et al. (2007). "A Common Genetic Encoding for Both Direct and Indirect Encodings of Networks". In: *Genetic and Evolutionary Computation Conference (GECCO-2007)*, pp. 1029–1036.

Kassahun, Yohannes, José de Gea Fernández, Mark Edgington, et al. (2008). "Accelerating Neuroevolutionary Methods Using a Kalman Filter". In: *Proceedings of the 10th Genetic and Evolutionary Computation Conference. 10th Genetic and Evolutionary Computation Conference (GECCO-2008), July 12-16, Atlanta, Georgia, United States*. Ed. by M. Keijzer. ACM, pp. 1397–1404.

Kassahun, Yohannes, José de Gea Fernández, Jan Hendrik Metzen, et al. (2008). "EANT +KALMAN: An Efficient Reinforcement Learning Method for Continuous State Partially Observable Domains". In: *KI 2008: Advances in Artificial Intelligence, in Conjunction with 31st Annual German Conference on AI (KI-2008), September 23-26*. Ed. by A. Dengel; K. Berns; T.M. Breuel; F. Bomarius; T.R. Roth-Berghofer. Vol. 5243. Lecture Notes in Artificial Intelligence. Kaiserslautern, Germany: Springer, Berlin/ Heidelberg, pp. 241–248.

Kassahun, Yohannes, Jan Hendrik Metzen, Mark Edgington, et al. (2009). "Incremental acquisition of neural structures through evolution". In: *Design and Control of Intelligent Robotic Systems*. 1st ed. Vol. 177. Studies in Computational Intelligence. Springer Verlag, pp. 187–208.

Kassahun, Yohannes, Jan Hendrik Metzen, José de Gea Fernández, et al. (2007). "A General Framework for Encoding and Evolving Neural Networks". In: *KI 2007: Advances in Artificial Intelligence. 30th Annual German Conference on Artificial Intelligence*. Ed. by J. Hertzberg; M. Beetz; R. Englert. Vol. 4667. Lecture Notes in Computer Science. Osnabrück, Germany: Springer, pp. 205–219.

Kassahun, Yohannes, Jakob Schwendner, et al. (2009). "Learning Complex Robot Control Using Evolutionary Behavior Based Systems". In: *Genetic and Evolutionary Computation Conference 2009. Genetic and Evolutionary Computation Conference 2009 (GECCO-2009), July 8-12, Montréal, Canada*.

Metzen, Jan Hendrik et al. (2007). "Performance Evaluation of EANT in the Robo Cup Keepaway Benchmark". In: *Proceedings of the 6th International Conference on Machine Learning and Applications (ICMLA-2007), December 13-15, Cincinnati, OH, USA. Cincinnati*. Cincinnati, Ohio, USA: IEEE, pp. 342–347.

Metzen, Jan Hendrik et al. (2008a). "Analysis of an Evolutionary Reinforcement Learning Method in a Multiagent Domain". In: *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008). Estoril*. Estoril, Portugal, pp. 291–298.

Metzen, Jan Hendrik et al. (2008b). "Evolving Neural Networks for Online Reinforcement Learning". In: *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature. PPSN-2008, September 13-17, Dortmund, Germany*. Ed. by G. Rudolph; T. Jansen; S.M. Lucas; et al. Vol. 5199. Lecture Notes in Computer Science. Springer, pp. 518–527.

Metzen, Jan Hendrik et al. (2008c). "Towards Efficient Online Reinforcement Learning Using Neuroevolution". In: *Proceedings of the 10th Genetic and Evolutionary Computation Conference. GECCO-2008, July 12-16, Atlanta„ GA, United States*. Poster, pp. 1425–1426.

Römmermann, Malte et al. (2008). "Learning Walking Patterns for Kinematically Complex Robots Using Evolution Strategies". In: *10th International Conference on Parallel Problem Solving from Nature. PPSN-2008, September 13-17, Dortmund, Germany*. Ed. by G. Rudolph; T. Jansen; S.M. Lucas; et al. Vol. 5199. Lecture Notes in Computer Science. Springer, pp. 1091–1100.

Other

These references only include publications not already referenced above.

Austerweil, Joseph L et al. (2015). "Structure and flexibility in bayesian models of cognition". In: *Oxford handbook of computational and mathematical psychology*. Oxford University Press. Chap. 9, pp. 187–208.

Bengio, Yoshua (2009). "Learning deep architectures for AI". In: *Foundations and Trends in Machine Learning* 2.1, pp. 1–127.

Bennett, Kristin P (1992). *Decision tree construction via linear programming*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin.

Borenstein, Johann and Liqiang Feng (1996). "Measurement and correction of systematic odometry errors in mobile robots". In: *IEEE Transactions on Robotics and Automation* 12.6. Ed. by Liqiang Feng, pp. 869–880.

Bubic, Andreja, D Yves Von Cramon, and Ricarda I Schubotz (2010). "Prediction, cognition and the brain". In: *Frontiers in human neuroscience* 4, p. 25.

Dasgupta, Abhijit and Adrian E Raftery (1998). "Detecting features in spatial point processes with clutter via model-based clustering". In: *Journal of the American Statistical Association* 93.441, pp. 294–302.

Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

Eliazar, Austin I. and Ronald Parr (2004a). "DP-SLAM 2.0". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. Vol. 2, pp. 1314–1320.

Eliazar, Austin I. and Ronald Parr (2004b). "Learning Probabilistic Motion Models for Mobile Robots". In: *Proc. of the International Conference on Machine Learning (ICML)*. ACM Press, pp. 32–39.

Fahlman, S. E. and C. Lebiere (1990). "The cascade-correlation learning architecture". In: *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, pp. 524–532.

Fraley, Chris and Adrian E Raftery (1998). "How many clusters? Which clustering method? Answers via model-based cluster analysis". In: *The computer journal* 41.8, pp. 578–588.

Ghahramani, Zoubin and Michael I. Jordan (1994). "Supervised learning from incomplete data via an EM approach". In: *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, pp. 120–127.

Griffiths, Thomas L and Zoubin Ghahramani (2006). "Infinite latent feature models and the Indian buffet process". In: *Advances in Neural Information Processing Systems*, pp. 475–482.

Hesslow, Germund (2012). "The current status of the simulation theory of cognition". In: *Brain research* 1428, pp. 71–79.

Hoffmann, Jan (2007). "Proprioceptive Motion Modeling for Monte Carlo Localization". In: *RoboCup 2006: Robot Soccer World Cup X*. Berlin, Heidelberg: Springer-Verlag, pp. 258–269.

Hommel, Bernhard et al. (2001). "The theory of event coding (TEC): A framework for perception and action planning". In: *Behavioral and Brain Sciences* 24.5, pp. 849–878.

Kaboli, Armita, Michael Bowling, and Petr Musilek (2006). "Bayesian Calibration for Monte Carlo Localization". In: *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pp. 964–969.

Kiong, L. C., M. Rajeswari, and M. V. C. Raoa (2003). "Extrapolation detection and novelty-based node insertion for sequential growing multi-experts network". In: *Applied Soft Computing* 3.2, pp. 159–175.

Klaassen, B. et al. (2002). "Biomimetic Walking Robot SCORPION: Control and Modeling". In: *Robotics and Autonomous Systems* 41.2, pp. 69–76.

Kristan, Matej (2010). "Multivariate online kernel density estimation". In: *Computer Vision Winter Workshop*, pp. 1–8.

Lang, Kevin J and Michael J Witbrock (1988). "Learning to Tell Two Spirals Apart". In: *Proc. of 1988 Connectionist Models Summer School*.

Lichman, M. (2013). *UCI Machine Learning Repository*. URL: `http://archive.ics.uci.edu/ml`.

Mardia, Kantilal Varichand, John T. Kent, and John M. Bibby (1979). *Multivariate analysis*. Probability and mathematical statistics. London: Academic Press. XV, 521.

Martinelli, Agostino, Nicola Tomatis, and Roland Siegwart (2007). "Simultaneous localization and odometry self calibration for mobile robot". In: *Autonomous Robots* 22.1, pp. 75–85.

Parzen, Emanuel (1962). "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3, pp. 1065–1076.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Priebe, Carey E. and David J. Marchette (1993). "Adaptive mixture density estimation". In: *Pattern Recognition* 26.5, pp. 771–785.

Roy, Nicholas and Sebastian Thrun (1999). "Online self-calibration for mobile robots". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2292–2297.

Serre, Thomas et al. (2007). "A quantitative theory of immediate visual recognition." In: *Progress in brain research* 165, pp. 33–56.

Shumway, Robert H and David S Stoffer (1991). "Dynamic linear models with switching". In: *Journal of the American Statistical Association* 86.415, pp. 763–769.

Sjoberg, Eric, Kevin Squire, and Craig Martell (2007). "Online parameter estimation of a robot's motion model". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS)*, pp. 735–740.

Stronger, Daniel and Peter Stone (2005). "Simultaneous Calibration of Action and Sensor Models on a Mobile Robot". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Sung, Hsi Guang (2004). "Gaussian mixture regression and classification". PhD thesis. Rice University.

Szewczyk, William F. (2005). "Time-evolving adaptive mixtures". In: *National Security Agency, Tech. Rep*, pp. 1–21.

Thrun, S. and J. J. Leonard (2008). "Simultaneous Localization and Mapping". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer, pp. 871–889.

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic Robotics*. MIT Press.

Visatemongkolchai, Artit and Hong Zhang (2007). "Building Probabilistic Motion Models for SLAM". In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*.

Wolberg, William H et al. (1995). "Computer-derived nuclear features distinguish malignant from benign breast cytology". In: *Human Pathology* 26.7, pp. 792–796.

Yeh, I-C (1998). "Modeling of strength of high-performance concrete using artificial neural networks". In: *Cement and Concrete research* 28.12, pp. 1797–1808.