Bayesian Gaussian Processes for Regression and Classification

Mark N. Gibbs

A dissertation submitted in candidature for the degree of Doctor of Philosophy, University of Cambridge.

Abstract

Bayesian inference offers us a powerful tool with which to tackle the problem of data modelling. However, the performance of Bayesian methods is crucially dependent on being able to find good models for our data. The principal focus of this thesis is the development of models based on Gaussian process priors. Such models, which can be thought of as the infinite extension of several existing finite models, have the flexibility to model complex phenomena while being mathematically simple.

In this thesis, I present a review of the theory of Gaussian processes and their covariance functions and demonstrate how they fit into the Bayesian framework. The efficient implementation of a Gaussian process is discussed with particular reference to approximate methods for matrix inversion based on the work of Skilling (1993). Several regression problems are examined. Non-stationary covariance functions are developed for the regression of neuron spike data and the use of Gaussian processes to model the potential energy surfaces of weakly bound molecules is discussed.

Classification methods based on Gaussian processes are implemented using variational methods. Existing bounds (Jaakkola and Jordan 1996) for the sigmoid function are used to tackle binary problems and multi-dimensional bounds on the softmax function are presented for the multiple class case. The performance of the variational classifier is compared with that of other methods using the CRABS and PIMA datasets (Ripley 1996) and the problem of predicting the cracking of welds based on their chemical composition is also investigated.

The theoretical calculation of the density of states of crystal structures is discussed in detail. Three possible approaches to the problem are described based on free energy minimization, Gaussian processes and the theory of random matrices. Results from these approaches are compared with the state-of-the-art techniques (Pickard 1997)

Preface

This dissertation describes work carried out between October 1994 and September 1997 in the Inferential Sciences Group of the Cavendish Laboratory, Cambridge. With the exception of Section 3.10.4 which has been done in collaboration with David Brown and where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. It has not, nor has any similar dissertation, been submitted for a degree, diploma or any other qualification at this or any other university. This dissertation does not exceed sixty thousand words in length.

During the three years of my Ph.D., I have had cause to be grateful for the advice, support and understanding of many people. In particular, my supervisor, David MacKay, has been a constant source of good ideas and sound advice. His unwavering enthusiasm and clarity of thought have been an inspiration. I have also benefited from working closely with Chris Pickard, David Brown, Coryn Bailer-Jones and Paul Haines, all of whom provided interesting problems to examine and were remarkably reasonable when yet another piece of code failed to compile.

Much advice and many useful discussions were provided by Simon Wilson, Matthew Davey, Anthony Challinor, Thomas Fink, Christian Waite, Peter Haynes, Zareer Dadachanji, Mike Payne, Steve Gull, Chris Williams, Dirk Husmeier, Steve Waterhouse, Martin Oldfield, Chris Watts, Alex Stark, Scott Schmidler, Richard Tucker, Ben Simons, David Burns and Bill Fitzgerald. I would particularly like to thank Simon and Matthew for comments on drafts of my thesis and for understanding what it is that makes me really, really mad.

As well as the many good friends listed above, there are three others who have made my life significantly easier: Mike by knowing what is and what is not good to eat, Al by having a ceaseless supply of "good" jokes and Chips by being the laziest person I have ever met.

This thesis is dedicated to my father, my mother, Paul and Siân.

Contents

1 Introduction **1**

1.1 Bayesian Modelling . . . 1

1.1.1 A Brief History of Bayesian Inference . . . 2

1.1.2 Inference and Bayes' Rule . . . 2

1.1.3 Problems, Problems 4

1.1.4 Model Properties . . . 4

1.2 Overview . . . 9

2 Regression using Gaussian Processes **11**

2.1 Bayesian Regression . . . 11

2.2 The Gaussian Process Model . . . 12

2.3 The Covariance Function . . . 14

2.3.1 Stationary Covariance Functions . . . 14

2.3.2 Non-stationary Covariance Functions . . . 16

2.3.3 Generating Covariance Functions . . . 19

2.4 Determining the Hyperparameters of a Gaussian Process . . . 19

2.4.1 Evidence Maximization . . . 19

2.4.2 Monte Carlo Approach . . . 22

2.4.3 Evidence vs. Monte Carlo . . . 22

2.5 Relationship to other Models . . . 23

3 Implementation of Gaussian Processes **27**

3.1 Exact Implementations . . . 27

3.1.1 Indirect LU Implementation . . . 28

3.1.2 Indirect SVD Implementation . . . 28

3.2 Approximate Implementation . . . 29

3.3 Conjugate Gradient Inversion . . . 30

3.4 Convergence of the Algorithm . . . 31

3.4.1 Eigenstructure of the Covariance Matrix . . . 32

3.5 Upper Bound on $Q_{\max}$ . . . 36

3.6 Tridiagonal Representation of Conjugate Gradients . . . 37

3.7 Imperfect Orthogonalization and Optimizations based on $\mathbf{A}$ . . . 40

3.8 The Trace . . . 42

3.9 Counting the Cost . . . 42

3.10 Examples . . . 43

3.10.1 1D Toy Example . . . 43

3.10.2 2D Toy Example using Approximate Inversion Techniques . . . 44

3.10.3 Zebra Finch Neuron Spike . . . 45

3.10.4 Vibrational ground state properties of weakly bound molecular systems . . . 55

4 Classification using Gaussian processes **60**
4.1 Binary Classifiers . . . 60
4.2 Variational Gaussian Process Model . . . 61
4.3 Making Predictions . . . 63
4.3.1 Predictions based on the lower bound . . . 63
4.3.2 Predictions based on the upper bound . . . 65
4.4 Determining the Parameters . . . 66
4.4.1 The Lower Bound . . . 67
4.4.2 The Upper Bound . . . 67
4.4.3 Optimization Procedure . . . 68
4.5 Examples . . . 68
4.5.1 1D Toy Example . . . 68
4.5.2 The CRABS and PIMA examples . . . 68
4.5.3 Weld Strength Example . . . 70
4.6 Conclusions . . . 71

5 Multi-class classification **73**
5.1 The Upper Bound . . . 74
5.2 The Lower Bound . . . 74
5.3 Tests of the Lower Bound . . . 75
5.4 Making Predictions . . . 78
5.4.1 Predictions based on the Upper Bound . . . 78
5.4.2 Predictions based on the Lower Bound . . . 78
5.4.3 Using the Gaussian approximations . . . 79
5.5 Determining the Parameters . . . 80
5.6 Examples . . . 81
5.6.1 Toy Problem no.1 . . . 81
5.6.2 Toy Problem no.2 . . . 82
5.7 Conclusions . . . 85

6 Theoretical Densities of States **86**
6.1 The Physics of Electron Energy Loss Spectra . . . 86
6.2 Density of States and *ab initio* quantum mechanics . . . 88
6.3 The Mixture Model Approach . . . 90
6.3.1 Free Energy Method . . . 91
6.3.2 Priors and Hyperparameters . . . 94
6.3.3 Matrix Elements and Crystal Symmetry . . . 95
6.3.4 Performance of the Mixture Model Approach . . . 96
6.4 The Gaussian Process Approach . . . 96
6.4.1 Incomplete Bands . . . 100
6.4.2 Performance of the Gaussian process Approach . . . 101
6.5 Random Matrix Approach . . . 102
6.5.1 Gaussian Ensembles . . . 107
6.5.2 Joint Probability Density Function of the Eigenvalues . . . 110
6.5.3 Implementation of the Random Matrix Approach . . . 111
6.5.4 Is Random Matrix Theory applicable? . . . 111
6.6 Pickard's Approach . . . 113
6.7 Conclusions . . . 114

7 Conclusions 115

A Principal Gaussian Process Equations 117

B The Approximate Inversion Algorithm 119

C Joint Probability Density Function of H 121

Chapter 1

Introduction

It's a dark night and the thunder is booming. Lightning cracks in the sky overhead and the rain hammers down on the old tile roof of the mansion. The electricity is down and we are standing next to Brad outside a dark, sinister looking room. Do we go inside?

There are two crucial problems in data modelling that we need to address: how to learn by example and how to reason in the face of uncertainty.

We peer into the room and feel a shiver go down our spine. Remembering the police report on the radio of an escaped axe-wielding maniac from the nearby prison, we turn nonchalantly to Brad and say "You go first."

Once we have received some data, we can construct a model, adapt the model in the light of the data and then use it to make inferences.

The axe-murderer must be in there. It's been two minutes now and Brad still hasn't come back out of the room. Oh no! Brad's probably been diced. Quick, let's get out of here!

Of course, there are many plausible models we could construct.

Calm down. Maybe there's no dribbling psychotic in the room. It might be our surprise birthday party or perhaps Brad is just fumbling around in the dark looking for the light switch.

How do we decide which is the most plausible model? To put it another way, how can we consistently assign preferences to different models based upon limited data? The incorporation of prior knowledge is another important task. Prior knowledge can have a significant effect on which model we think is most plausible and on the inferences that we make using our model

What to do? Should we flee or stay for the party? Then we remember: it's Friday the thirteenth. As the clock strikes midnight, the full moon comes into view from behind the clouds. We hit the ground running.

1.1 Bayesian Modelling

The task of data modelling occurs in many areas of research. We design a model for a particular system and, when data arrives from the system, we adapt the model in light of the data. The model provides us with a representation of our prior beliefs about the system and the information about the system that the data has provided. We can then use the model to make inferences. As the previous section pointed out, we need a consistent framework within which to construct our model, incorporating any prior knowledge and within which to consistently compare our model

with others. In this thesis we choose the Bayesian framework.

The Bayesian approach is based upon the expression of knowledge in terms of probability distributions. Given the data and a specific model, we can deterministically make inferences using the rules of probability theory. In principle Bayesian theory always provides us with a unique solution to our data modelling problems. In practice such solutions may be difficult to find. The key to a successful implementation of Bayesian methods is the choice of model and the mathematical techniques we then use to analyse said model. In this thesis we shall focus on a group of models based upon Gaussian process priors and describe techniques that can be used to solve a variety of interesting and challenging real world problems.

1.1.1 A Brief History of Bayesian Inference

Bernoulli was one of the first to raise the question of how we might use deductive logic to solve inductive problems. In his book *Ars Conjectandi* (Bernoulli 1713), he discussed the convergence of Binomial distributions and the relationship of uncertainty to probability but he did not formulate any corresponding mathematical structure. Such a structure was provided by the Reverend Thomas Bayes in a paper published posthumously by a friend (Bayes 1763). In this paper, Bayes discussed the inference of the parameters of a Binomial distribution using observational data and derived his now famous theorem for this specific case.

The present form of Bayes theorem is due to Laplace (1812). While Bayes' formulation applied only to Binomial distributions, Laplace broadened the theorem to include any probability distribution. Unfortunately his approach was largely rejected by the mathematicians of his day. They believed probability to be defined in terms of frequencies and not as a measure of belief as Laplace did. To them probability could only be applied to events which occurred many times. They disliked the subjective element introduced by thinking of probability in terms of belief, preferring the supposedly objective frequentist view.

After Laplace, Bayesian inference was largely ignored until early this century. It was then developed in detail by Jeffreys (1939) and placed on an axiomatic footing by Cox (1946). Interest in the subject grew steadily over the next fifty years. This thesis draws its view of Bayesian theory primarily from the works of Box and Tiao (1973), Gull (1988), MacKay (1992a) and Neal (1996).

1.1.2 Inference and Bayes' Rule

We have constructed a model or hypothesis $\mathcal{H}_i$ with a set of parameters $\mathbf{w}_i$ which we shall use to model some data $\mathcal{D}$. Within the Bayesian approach to data modelling there are then two levels inference. The first level is to infer the parameters $\mathbf{w}_i$ of the model given the data. Then we wish to compare our present model $\mathcal{H}_i$ with others. This process of ranking the models is the second level of inference.

In order to perform both these levels of inference, Bayesians use probability distributions. We start by expressing our prior beliefs about which models are *a priori* most suitable for the data in terms of a probability distribution over all possible models, $P(\mathcal{H}_i)$. If we believe the data is coming from a system with a certain symmetry, for example, we can assign models with such a symmetry a greater probability. If a model has a set of parameters $\mathbf{w}_i$ then we express our prior beliefs about the value of these parameters in terms of another prior distribution, $P(\mathbf{w}_i|\mathcal{H}_i)$. The prior beliefs about the parameters depends on their function within the model and so this distribution is conditional on $\mathcal{H}_i$.

The data now arrives. In the first level of inference we infer the parameters of the model given the data. Each model makes predictions about how likely the data is given that it was generated by that model with a specific set of parameters $\mathbf{w}_i$. These predictions are embodied in the distribution $P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i)$. It is at this point that we turn to Bayes' theorem. We need to know how to combine

the prior knowledge that we had about the setting of the parameters, $P(\mathbf{w}_i|\mathcal{H}_i)$, with the knowledge that we have just gained from the data, $P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i)$. Bayes' theorem tells us that

$$\overbrace{P(\mathbf{w}_i|\mathcal{D}, \mathcal{H}_i)}^{\text{Posterior}} = \frac{\overbrace{P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i)}^{\text{Likelihood}} \overbrace{P(\mathbf{w}_i|\mathcal{H}_i)}^{\text{Prior}}}{\underbrace{P(\mathcal{D}|\mathcal{H}_i)}_{\text{Evidence}}}. \tag{1.1}$$

In words, the total information we have about the value of the parameters, which is embodied in the posterior probability of the parameters $P(\mathbf{w}_i|\mathcal{D}, \mathcal{H}_i)$, is simply the product of $P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i)$ and $P(\mathbf{w}_i\mathcal{H}_i)$ with $P(\mathcal{D}|\mathcal{H}_i)$ as an appropriate normalizing constant. As the terms in Bayes' theorem are so commonly used, they each have names: $P(\mathbf{w}_i|\mathcal{D}, \mathcal{H}_i)$ is the Posterior, $P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i)$ is the Likelihood, $P(\mathbf{w}_i\mathcal{H}_i)$ is the Prior and $P(\mathcal{D}|\mathcal{H}_i)$ is the Evidence.

We can now go onto the second level of inference. Given that we can calculate $P(\mathcal{D}|\mathcal{H}_i)$, the evidence from equation 1.1, we can apply Bayes' theorem once more. $P(\mathcal{D}|\mathcal{H}_i)$ embodies what the data is telling us about the plausibility of each of the models $\mathcal{H}_i$ and we can combine this with our prior beliefs:

$$P(\mathcal{H}_i|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{H}_i)P(\mathcal{H}_i)}{P(\mathcal{D})} \tag{1.2}$$

where $P(\mathcal{D})$ is a normalizing constant. The posterior distribution $P(\mathcal{H}_i|\mathcal{D})$ allows us to rank our different models.

To illustrate the process of Bayesian inference, let us consider an example. A lighthouse lies a distance b off a flat coastline and at a position a along the coast. The lighthouse rotates and, once every revolution, it emits a very brief burst of light at a random angle θ_n toward the shore. This burst is then picked up at position x_n by a line of photo-detectors all along the shore. Given that we have recorded N bursts at positions $(x_1, x_2, \cdots, x_N)$, we wish to infer the position of the lighthouse along the shore, a, given some assumption about the value of b, the distance of the lighthouse off the shore.

The position at which the burst is detected along the shore is related to the angle of emission by $b \tan\theta_n = x_n - a$. We assume that the angle of emission is random and hence we can write

$$P(x_n|a, b)\mathrm{d}x_n = P(\theta_n)\mathrm{d}\theta_n \tag{1.3}$$

$$= \frac{\mathrm{d}\theta_n}{\pi} \tag{1.4}$$

By calculating $\mathrm{d}\theta_n/\mathrm{d}x_n$, we obtain a Cauchy distribution for the probability of x_n:

$$P(x_n|a, b) = \frac{b}{\pi(b^2 + (x_n - a)^2)} \tag{1.5}$$

Each burst is independent and so the probability of all the data $\mathcal{D} = \{x_n\}_{n=1}^N$ (also called the likelihood of a and b) is a product of Cauchy distributions:

$$P(\mathcal{D}|a, b) = \prod_{n=1}^{N} \frac{b}{\pi(b^2 + (x_n - a)^2)} \tag{1.6}$$

For our first model of the data, we shall assume that b has a value of 1.0 (this was the value used to generate the data). We shall also assume a uniform prior on a between -5.0 and 5.0. The unnormalized posterior distribution over a for N detections given the data can be written

$$P(a|b,\mathcal{D}) \propto \begin{cases} \frac{1}{10}\prod_{n=1}^{N} \frac{b}{\pi(b^2+(x_n-a)^2} & -5.0 < a < 5.0 \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

We start with $N = 0$, i.e., no knowledge about the position of the lighthouse. The posterior distribution then only reflects the prior (see Figure 1.1(a)). As data begins to arrive, the posterior changes to reflect the new information we have gained about a (Figure 1.1(b)-(f)). We can see that as N increases, the posterior becomes sharply peaked about $a = 1.0$ (which was the value used to generate the data).

For our second model of the data, we shall assume that $b = 0.75$. We can then compare this new model with the previous one by calculating the evidence, $P(\mathcal{D}|b)$, by numerical integration (see Figure 1.1(g)). The first ten data points do not tell us enough to think either model particularly more plausible than the other. As the amount of data received increases, the model with the correct value of b becomes more plausible than the model with the incorrect value of b.

1.1.3 Problems, Problems ...

Applying Bayesian methods is not always as straightforward as the previous example might suggest. There are three principal areas of difficulty. The first one centres around the mathematical complexity that often occurs in Bayesian approaches. As previously mentioned, approximations often have to be made to avoid intractable integrals and awkward equations. Most commonly, problems arise in evaluating the evidence (see equation 1.1). This is frustrating as it is the evidence which is one of the most useful quantities in the Bayesian framework.

Another potential problem with the Bayesian approach is the prior. Classical statisticians would argue that the Bayesian prior is subjective and hence the Bayesian framework is of questionable worth for any form of comparison. However this is not the case. Any prior placed on an object is determined by our prior knowledge of that object. For the prior to be objective we only require that people having exactly the same knowledge about an object use the same prior. Hence if two people have different prior knowledge then we rightly expect them to use different priors. The problem with priors lies not with subjectivity but with the difficult process of assigning probabilities. How do we decide what numbers truly reflect what we believe? Priors also cause problems when the prior belief is very hard to express in terms of the models that we are using. For example, we might have data from a function which we know has a value of zero outside a certain region of input space. Finding a prior on, say, the weights of a neural network which embodies this information is hard. The temptation is to adapt the priors to fit the model, i.e., to choose mathematically convenient priors for a given model which do not embody our prior beliefs.

Finally, Bayesian model comparison has a closed hypothesis space. If we fail to include the "true" model amongst our set of possible candidates then this model will never be compared with others. There exists no Bayesian criterion for assessing whether or not our hypothesis space is correct.

1.1.4 Model Properties

The problems raised in the previous section highlight the importance of the choice of hypothesis space, i.e., the importance of choosing a good set of models to investigate given the data. For example, we might want to make sure that the hypothesis space contains models which have sufficient flexibility to capture any likely features of the data. Alternatively we might want to consider models which have a parameterization that allows us to express our prior beliefs in a mathematically tractable manner. The properties of the models in our hypothesis space are of

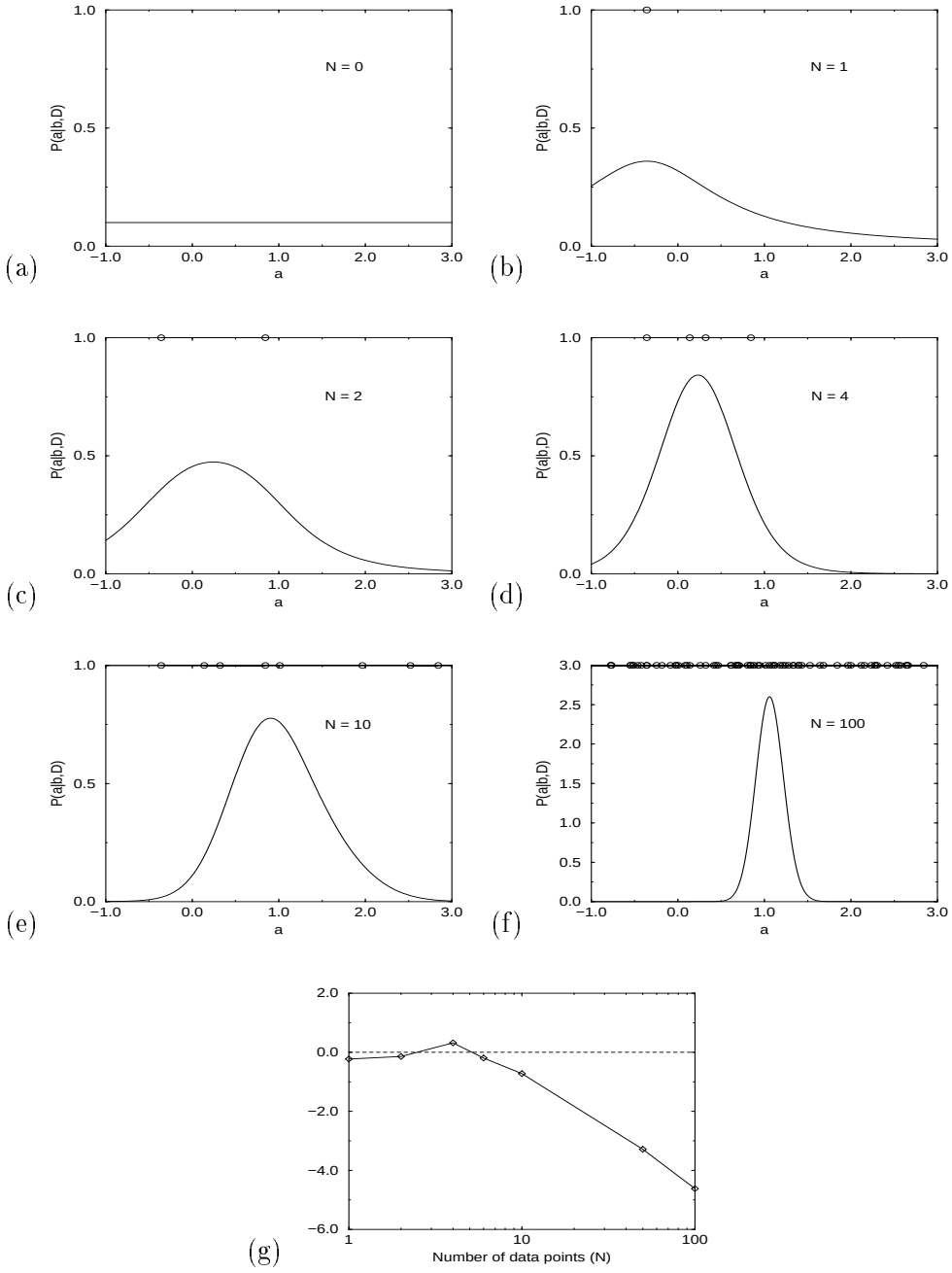


Figure 1.1: Lighthouse Example : (a) shows the uniform prior over the position of the lighthouse along the shore, a. (b)-(f) show the posterior distribution $P(a|b,\mathcal{D})$ as the number of data points, N, increases. The posterior becomes sharply peaked around $a = 1.0$, the value used to generate the data. The data points are shown as circles along the top of each plot. (g) shows the difference between the log of the evidence for Model 1 ($b = 1.0$) and Model 2 ($b = 0.75$) ($\log P(\mathcal{D}|b = 0.75) - \log P(\mathcal{D}|b = 1.0)$).

significant interest. Let us discuss these properties focusing on three inter-related areas: complexity, parameterization and interpretability.

Complexity

Consider the interpolation of some one dimensional data (see Figure 1.2). Initially we shall assume that obtaining the closest possible fit to the data is the ultimate goal of our modelling process. The fit obtained using a linear interpolant will be worse than that obtained using a quadratic function. Using a cubic will be better than a quadratic, a quartic better than a cubic and so on. As the number of parameters in the model increases, the quality of the fit increases and so it would seem appropriate to use an infinitely complex model.

However, our aim in interpolation problems is not usually to obtain the closest fit to the data but to find a balance between fitting the data and making sensible predictions about new events. Hugely complex models are often over-parameterized and, while fitting the data precisely, interpolate and extrapolate poorly. Intuitively we prefer the simplest model which accounts for the data rather than including unnecessary complications. As well as being intuitive this is also computational attractive.

Bayesian theory embodies Occam's Razor (Gull 1988) which states that we should not assign a higher plausibility to a complex model than to a simpler model when the extra complexity is not required to explain the data. Thus application of Bayesian techniques will prevent us from over-fitting our data unless we explicitly favour such complexity through our prior over models. However we should note that the complexity of a model is not necessarily determined by counting the number of parameters. Occam's Razor is not telling us to limit the number of parameters in our model. We can construct a neural network with an infinite number of hidden neurons while still retaining a tractable model with a low level of complexity provided that we have the appropriate priors on the weights (Neal 1996) (see Section 2.5 for details).

Another point to emphasise is that Occam's Razor does not suggest that we should vary the size of our model dependent on the amount of data we receive. Even if we have only a few samples from a system, this is no reason, in the event of no prior knowledge, to limit the complexity of the model. The system generating the data might be very complex and we have merely not obtained enough information about it to be certain how that complexity might manifest itself.

Parameterization

The parameters in our model $\mathbf{w}_i$ which we infer from the data in the first level of inference are somewhat of an inconvenience when it comes to model comparison. In order to compare different models we need to evaluate the evidence (see equation 1.1) and this involves the integral

$$P(\mathcal{D}|\mathcal{H}_i) = \int \mathrm{d}\mathbf{w}_i \; P(\mathcal{D}|\mathbf{w}_i, \mathcal{H}_i) P(\mathbf{w}_i|\mathcal{H}_i) \tag{1.8}$$

Both the likelihood and the prior are generally some non-linear function of the parameters. The form of the likelihood is often obvious: the exponential of a sum squared error term for regression with Gaussian noise; a cross-entropy term for classification. By placing a prior on the parameters that reflects our beliefs (rather than just being mathematically convenient), we are often left with an integral that is analytically intractable. Approximations, using Monte Carlo methods or expansion methods based on the most probable parameters $\mathbf{w}_i^{MP}$, are then necessary.

The exception to this rule is linear models. The likelihood of a linear model is Gaussian in terms of the parameters and so, for certain common priors (polynomial in $\mathbf{w}_i$, linear or quadratic in the exponent in terms of $\mathbf{w}_i$), the integral in equation 1.8 is tractable. Unfortunately, the utility of a linear model is determined by the choice of basis functions and the number of basis functions used.

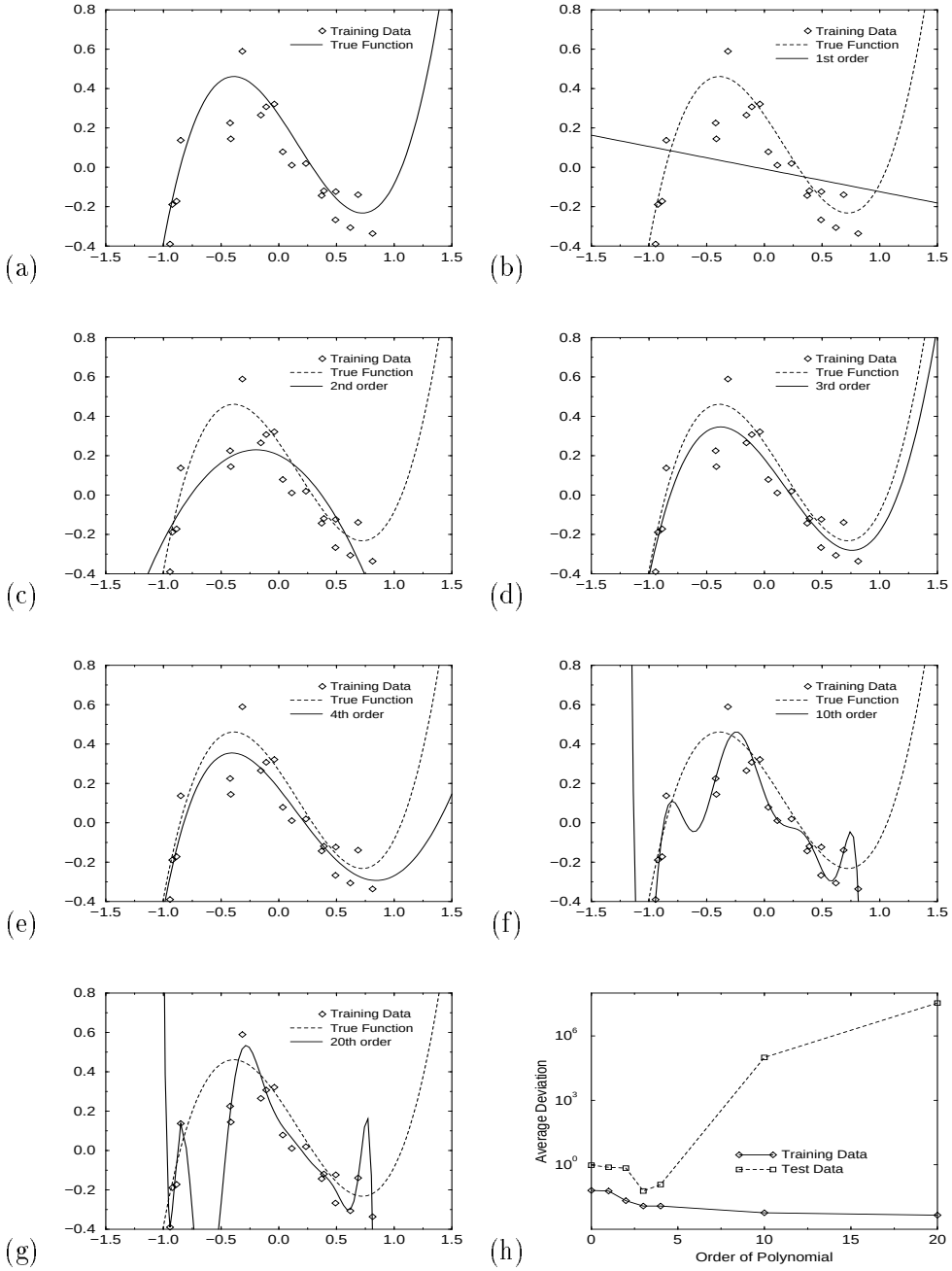


Figure 1.2: Fitting 1D data using polynomials : (a) shows 20 noisy random samples, $\{t_i\}$, taken from an underlying function. The closest fit to this data was found by minimizing the sum squared error $\sum_{i=1}^{20} (t_i - f(x_i))^2$ between the data for a variety of modelling functions $f(x)$. (b)-(g) show the fit obtained using linear, quadratic, cubic, quartic, 10^{th} order and 20^{th} order polynomials respectively. (h) shows the mean squared error between the training data and the predictions of each model and between a test set of 100 evenly spaced points in the region $-1.5 < x < 1.5$ and the predictions of each model. We can see that, while the training error steadily decreases, the test error reaches a minimum and then increases rapidly as the models begin to over-fit the training data.

It does not have the flexibility of a neural network which can be thought of as having adaptive basis functions. If the basis functions of the linear model are ill-suited to the problem at hand then many will be needed to obtain good results e.g. using sinusoids to model step functions. Intuitively we feel that using a large number of basis functions is an inefficient way of modelling. However, in the light of the previous discussion about complexity, perhaps we should reconsider. Can we, using an infinite number of basis functions and appropriate priors on the parameters $\mathbf{w}_i$, capture the flexibility of a non-linear model and retain the analytic solutions of a linear model without the computational cost being unmanageable? The answer is yes; such a model corresponds to a Gaussian process.

One objection to the approach of using infinite basis function networks in this manner is that we have seemingly disregarded the warning against choosing priors for their mathematical convenience. However, the prior on the parameters is not the only prior on the type of function we expect our linear model to produce. The choice of basis functions is also an implicit prior. For the case of an infinite linear model, the choice of basis functions translates into a choice of covariance function for the prior over possible functions (see section 2.5 for details). Thus the restrictions on the prior on the parameters do not prevent us from being able to embody our prior beliefs about the data.

Not all the parameters in a model are of the same type as the weights of a neural network. A higher level of parameters or "hyperparameters" often exists including noise variances and regularization constants. These hyperparameters are part of the second level of inference and define a continuous model space. The remaining structural variables of the model (for example the number of basis functions or the order of a polynomial) are known as the architecture and are also dealt with in the second level of inference. Searches across hypothesis space involve continuous searches over the hyperparameters and discrete searches over the architecture parameters. The continuous searches are preferable because, while automated discrete searches are possible (Green 1995), more sophisticated gradient based algorithms are available for continuous searches which are not appropriate for the discrete case. Ideally we would like to remove any uncertainty in architecture and simply be left with a search over hyperparameters. Of course, if we cannot evaluate the evidence analytically then each step of our search, continuous or otherwise, either involves optimization of the parameters $\mathbf{w}_i$ or is part of a much larger Monte Carlo simulation which also includes the parameters. The benefits of using a linear model are clear.

Interpretability

As well as providing a good fit to the data and plausible predictions based on the data, we might also hope that our model gives us information about the system from which the data came. The easiest way to gain such information is to use models where the parameters and hyperparameters have clearly interpretable meanings. In the case of regression models, the hyperparameters often tell us more about the generating system than the parameters $\mathbf{w}_i$ as there can be a large amount of redundancy among the parameters. The weights of a fully connected feed-forward neural network do not tell us a great deal about the data but the hyperparameters, if defined in a sensible manner, can lead to a greater understanding. For example, automatic relevance determination (MacKay 1995b; Neal 1996), the process of determining whether individual inputs cause significant variation in the outputs, can be performed by specifying different regularization classes for weights connected to each input. Many other models have been devised where the parameters and hyperparameters can be given physical meaning: Hierarchical mixtures of experts (Waterhouse and Robinson 1995), Belief networks (Pearl 1988) and certain fuzzy logic architectures (Mendel 1994) to name a few. Using models with interpretable parameters and hyperparameters also helps us express our priors easily as we can target the desired behaviour by focusing on the associated parameter(s).

1.2 Overview

In the discussions of the previous sections we have established various criteria that we would like an ideal model to meet. As well as providing the flexibility to model complex phenomena, we would like the algebraic simplicity of a linear model. We would also prefer a model which had a continuous model space defined in terms of hyperparameters, making searches over the model space easier. Finally we would like the hyperparameters of the model to have some identifiable significance with respect to the data so that we can use them to gain knowledge about the system from which the data came and to facilitate embodying our prior beliefs.

We have also emphasised the point that the complexity of a model cannot be determined simply by counting parameters. This raises the interesting question of whether infinite models, which we might previously have dismissed as computationally infeasible, can be implemented. In answer to this, one can examine the use of polynomials of infinite order by Young (1977) and the mixture models with an infinite number of components used by Neal (1992). Several of these infinite models (infinite fixed basis function networks and neural networks with an infinite number of hidden units and appropriate priors) correspond to models where the joint distribution of all the data is Gaussian. Such models are known as Gaussian processes.

While the investigation of Gaussian processes is far from new, only recently have people viewed them from a Bayesian perspective with the aim of using them as models for regression and classification tasks (Williams and Rasmussen 1996; Barber and Williams 1996; Neal 1997). It is the aim of this thesis to extend this work with special emphasis on non-stationary forms of covariance functions, efficient implementation of the Gaussian process framework for large training data sets and the application of variational methods to classification using Gaussian processes. Several toy and real world examples will be given to highlight the advantages and disadvantages of the methods presented.

It is not the aim of this thesis to provide a rigorous comparison between Gaussian processes and other regression and classification methods. Rasmussen (1996) has used the DELVE environment to carefully compare many methods to a Gaussian process approach. While several comparisons are made in this thesis with the work of others on specific datasets (regression of neuron spike data by MacKay and Takeuchi (1994), inference of potential energy surfaces (Brown *et al.* 1996), classification of the PIMA and CRABS datasets by Barber and Williams (1996) and Ripley (1994), weld strength classification (Ichikawa *et al.* 1996), theoretical calculations of densities of states by Pickard (1997)) these are done as illustrations and are not meant as definitive comparisons between Gaussian processes and other methods.

Chapter two covers the Gaussian process approach to regression. I introduce the Gaussian process from a Bayesian perspective and describe the properties of the covariance function. Having given examples of both stationary and non-stationary covariance functions, I show how the hyperparameters of a Gaussian process can be determined using Evidence maximization (MacKay 1992a). I also discuss the pros and cons of Evidence maximization in comparison to a Markov chain Monte Carlo approach. Previous work on Gaussian processes is then briefly reviewed and the relationship of fixed basis function networks and neural networks to Gaussian processes is discussed.

Chapter three concentrates on the practicalities of implementing a Gaussian process. I discuss several methods for calculating the inverse of the covariance matrix. In particular I describe the approximate matrix inversion techniques of Skilling (1993) which are similar to the Lanczos algorithm but make use of the structure of the noisy covariance matrix. Several examples of Gaussian process regression are then given. Two toy problems are used to demonstrate some of the features of Gaussian processes and the Skilling matrix inversion techniques. The regression of noisy neuron spike data is then addressed using a non-stationary covariance function and a stationary function with a non-linear map on the input space. Finally I describe the work of Brown *et al.* (1996) using

Gaussian processes to model the potential energy surfaces of weakly bound molecules.

The problem of binary classification using Gaussian processes is tackled in chapter four. Rather than following the previous approaches of Barber and Williams (1996) and Neal (1997), I make use of the variational methods of Jaakkola and Jordan (1996) in order to compute approximations to the predictive distributions of a binary classifier. I then examine four examples: a one dimensional toy example to demonstrate the features of the method; a comparison of the performance of the variational Gaussian process classifier to other methods on the CRABS and PIMA datasets (from Barber and Williams (1996), Ripley (1994) and Ripley (1996)); finally I study the variation of weld strengths with composition of the welding material and welding procedure and compare this to the results of Ichikawa *et al.* (1996).

Chapter five examines multi-class classification, again using variational techniques. An upper bound is found for the inverse of the normalizing constant $Z = \sum_i e^{a^{(i)}}$. A lower bound for Z^{-1} is proposed and tested numerically. The upper and lower bounds are then used in a similar manner to the binary case to compute approximations to the predictive distributions of the classifier. Two toy examples are then studied: a two dimensional three class problem and a four dimensional three class problem.

In chapter six, I investigate the theoretical calculation of electron energy loss spectra of crystals using the densities of states. I explore three possible methods, the first using a constrained mixture model, the second based on Gaussian processes and the third based on the theory of random matrices. I compare the results with the present state-of-the-art technique (Pickard 1997).

In the final chapter, I discuss the achievements and failings of the Gaussian process framework and suggest possible directions for further research.

CHAPTER 2

Regression using Gaussian Processes

Regression is one of the most common data modelling problems and numerous methods exist for tackling it. In this chapter we shall examine the Gaussian process approach to regression. After a brief discussion of the Bayesian approach to regression, we will look at how Gaussian processes fit into the Bayesian framework. In this chapter we will primarily be concerned with the theory of Gaussian processes rather than their implementation or any examples (see the following chapter). We will examine the role of the covariance function, reviewing the positive definite constraint and discussing stationary and non-stationary covariance functions. Determination of the hyperparameters will be addressed using Evidence maximization (MacKay 1992a) and the advantages and disadvantages of this approach in comparison to MCMC methods will be discussed. Finally, we will review previous work on Gaussian processes and show how Gaussian processes relate to fixed basis function networks and certain types of feed-forward neural networks.

2.1 Bayesian Regression

Let us briefly summarize the problem we wish to solve. We have some noisy data $\mathcal{D}$ which consists of N pairs of L-dimensional input vectors $\{\mathbf{x}_n\}$ and scalar outputs $\{t_n\}$ for $n = 1 \cdots N$. We wish to construct a model for this data and then make predictions using this model. For example, given some new input vector $\mathbf{x}_{N+1}$ we may wish to find the distribution of the corresponding output t_{N+1}.

We can write down a generative model for our data

$$t_n = y(\mathbf{x}_n) + \nu_n \tag{2.1}$$

where $y(\mathbf{x})$ is our modelling function and ν_n is some noise. We define a prior over the space of possible functions to model the data $P(y|\mathbf{A})$ where $\mathbf{A}$ is a set of hyperparameters. We also define a prior over the noise $P(\nu|\mathbf{B})$ where ν is the vector of noise values $\nu = (\nu_1, \nu_2, \cdots, \nu_N)$ and $\mathbf{B}$ is a set of hyperparameters of the noise model. We can write down the probability of the data given the hyperparameters,

$$P(\mathbf{t}_N|\{\mathbf{x}_n\}, \mathbf{A}, \mathbf{B}) = \int dy\, d\nu\; P(\mathbf{t}_N|\{\mathbf{x}_n\}, y, \nu) P(y|\mathbf{A}) P(\nu|\mathbf{B}) \tag{2.2}$$

where $\mathbf{t}_N = (t_1, t_2, \cdots, t_N)$. Now if we define the vector $\mathbf{t}_{N+1} = (t_1, \cdots, t_N, t_{N+1})$ we can write down the conditional distribution of t_{N+1}

$$P(t_{N+1}|\mathcal{D}, \mathbf{A}, \mathbf{B}, \mathbf{x}_{N+1}) = \frac{P(\mathbf{t}_{N+1}|\{\mathbf{x}_n\}, \mathbf{A}, \mathbf{B}, \mathbf{x}_{N+1})}{P(\mathbf{t}_N|\{\mathbf{x}_n\}, \mathbf{A}, \mathbf{B})} \tag{2.3}$$

and hence use this conditional distribution to make predictions about t_{N+1}. It is important at this point to note the notational difference between t_{N+1} (the unknown output value at $\mathbf{x}_{N+1}$) and $\mathbf{t}_{N+1}$ (the vector constructed from $\mathbf{t}_N$ and t_{N+1}).

In general the integration in equation 2.2 is complicated. Given the generative model in equation 2.1, $P(\mathbf{t}_N|\{\mathbf{x}_n\}, y, \nu)$ is a product of delta functions $\prod_n \delta(\nu_n = t_n - y(\mathbf{x}_n))$. Hence, assuming Gaussian noise and integrating over ν, equation 2.2 becomes

$$P(\mathbf{t}_N|\{\mathbf{x}_n\}, \mathbf{A}, \mathbf{B}) = \int dy\, P(y|\mathbf{A}) \frac{1}{Z} \exp\left(-\frac{\beta}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n) - t_n)^2\right) \tag{2.4}$$

where $\beta \in \mathbf{B}$ and Z is an appropriate normalizing constant.

In a Bayesian treatment of neural networks or fixed basis function networks, it is common to place a regularizing Gaussian prior on the weights of the network (MacKay 1992d),

$$P(\mathbf{w}|\mathbf{A}) = \frac{1}{Z_w} \exp\left(-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right) \tag{2.5}$$

where $\alpha \in \mathbf{A}$ is a regularization constant. For fixed basis function networks, a Gaussian prior on the weights is equivalent to placing a Gaussian prior on $\{y(\mathbf{x}_n\}$ which, due to the linear nature of the model, can be obtained using matrix inversion. The integration over y in equation 2.3 is then straightforward. However for more complex multiple layer neural networks (one hidden layer or more) with non-linear activation functions, $P(y|\mathbf{A})$ is not necessarily Gaussian given a Gaussian prior on the weights and the integral is generally analytically intractable.

There have been two standard approaches to avoid such difficulties. MacKay (1992a) uses a Gaussian approximation to the integral in equation 2.2 based upon the most probable weight vector given the data and the priors. Alternatively Neal (1996) uses Monte Carlo techniques to approximate the integral. We shall now consider an approach based on Gaussian process priors.

2.2 The Gaussian Process Model

A Gaussian process is a collection of random variables $\mathbf{t} = (t(\mathbf{x}_1), t(\mathbf{x}_2), \cdots)$ which have a joint distribution

$$P(\mathbf{t}|\mathbf{C}, \{\mathbf{x}_n\}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{t} - \mu)^T\mathbf{C}^{-1}(\mathbf{t} - \mu)\right) \tag{2.6}$$

for any collection of inputs $\{\mathbf{x}_n\}$. $\mathbf{C}$ is the covariance matrix of the data defined by the parameterized covariance function $C^{mn}(\mathbf{x}_m, \mathbf{x}_n; \Theta)$ where $C_{mn} = C^{mn}(\mathbf{x}_m, \mathbf{x}_n; \Theta)$ and Θ is some set of hyperparameters. The superscript mn denotes that the covariance function distinguishes between $\mathbf{x}_m$ and $\mathbf{x}_n$ even if $\mathbf{x}_m = \mathbf{x}_n$. This is especially relevant when dealing with noise models within the covariance function. In future the superscript will be suppressed for brevity but it applies to all covariance functions in this thesis. μ is the mean vector. The reason that Θ is described as a set of hyperparameters is that the parameters of the covariance function play a similar role to the hyperparameters of a neural network (see Section 2.5).

Let us define our data vector $\mathbf{t}_N$ to be generated by a Gaussian process as in equation 2.6 with a covariance matrix $\mathbf{C}_N$ and a mean vector $\mu_N = \mathbf{0}$. Obviously not all data can be modelled using a zero mean process. However we shall assume throughout this discussion that the data has been appropriately scaled such that a zero mean assumption is reasonable.

By explicitly stating the probability of the data we have now bypassed the step of expressing individual priors on the noise and the modelling function. Both of these priors have been combined into the covariance matrix $\mathbf{C}$. Thus we can see that the nature of $\mathbf{C}$ and hence of the covariance

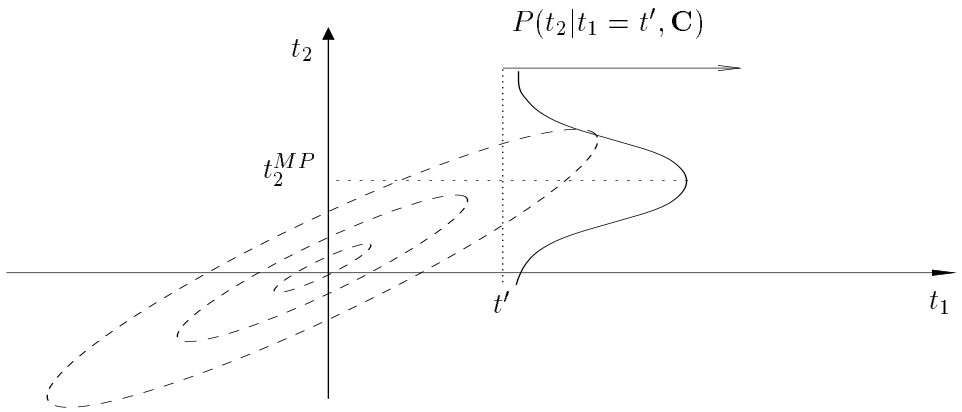


Figure 2.1: Conditional Distributions: The figure shows the contours of the Gaussian joint probability density of two data points t_1 and t_2. The density is characterized by its covariance matrix $\mathbf{C}$ which determines the scale and orientation of the Gaussian. Let t_1 take some value, say t'. We can then calculate the Gaussian conditional distribution of t_2 (shown in the diagram) and hence find the most probable value of t_2 given this particular value of t_1.

function is crucial to the whole Gaussian process approach. We shall discuss the covariance function at length in a later section.

When first encountering the Gaussian process approach, it is easy to misinterpret equation 2.6. We are not constraining the function used to model the data to be a mixture of Gaussians. We are expressing the correlations between the different points in input space. This can best be illustrated by a simple example. Consider Figure 2.1. This shows the Gaussian joint probability density of two data points t_1 and t_2 characterized by its covariance matrix $\mathbf{C}$. Given some value of $t_1 = t'$, we can find the Gaussian conditional distribution $P(t_2|t_1 = t', \mathbf{C})$ and hence find a most probable prediction of t_2 given $t_1 = t'$. The scale and orientation of the Gaussian joint probability density is wholly determined by the covariance matrix, once again underlining the pivotal role $\mathbf{C}$ plays in this approach to regression.

Returning to the more general case of making predictions about t_{N+1} given the data $\mathcal{D} = (\{\mathbf{x}_n\}, \mathbf{t}_N)$, we can use equation 2.6 and the fact that the joint probability of any number of data points must be Gaussian to calculate the Gaussian conditional distribution over t_{N+1}.

$$P(t_{N+1}|\mathcal{D}, C(\mathbf{x}_m, \mathbf{x}_n; \Theta), \mathbf{x}_{N+1}) = \frac{P(\mathbf{t}_{N+1}|C(\mathbf{x}_m, \mathbf{x}_n; \Theta), \mathbf{x}_{N+1}, \{\mathbf{x}_n\})}{P(\mathbf{t}_N|C(\mathbf{x}_m, \mathbf{x}_n; \Theta), \{\mathbf{x}_n\})} \tag{2.7}$$

$$= \frac{Z_N}{Z_{N+1}} \exp\left[-\frac{1}{2}\left(\mathbf{t}_{N+1}^T \mathbf{C}_{N+1}^{-1} \mathbf{t}_{N+1} - \mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N\right)\right] \tag{2.8}$$

where Z_N and Z_{N+1} are appropriate normalising constants. Figure 2.2 shows the relationship between $\mathbf{C}_{N+1}$ and $\mathbf{C}_N$. Collecting together those terms that are functions of t_{N+1} in equation 2.8, we can derive the Gaussian conditional distribution of t_{N+1} at $\mathbf{x}_{N+1}$.

$$P(t_{N+1}|\mathcal{D}, C(\mathbf{x}_m, \mathbf{x}_n; \Theta), \mathbf{x}_{N+1}) = \frac{1}{Z} \exp\left(-\frac{\left(t_{N+1} - \hat{t}_{N+1}\right)^2}{2\sigma^2_{\hat{t}_{N+1}}}\right) \tag{2.9}$$

where

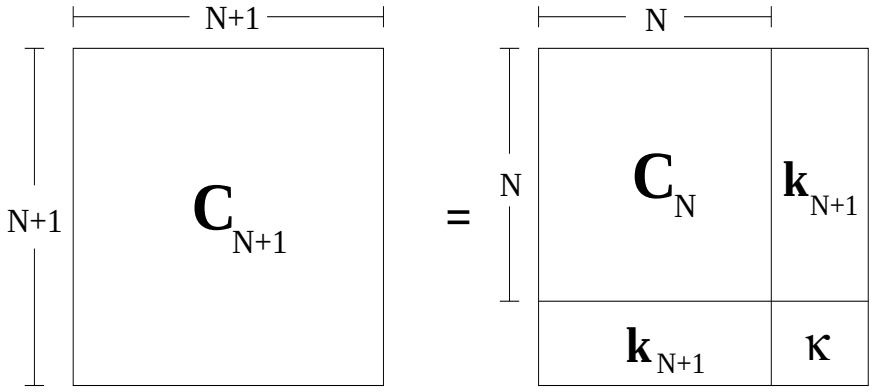


Figure 2.2: Covariance Matrices : The larger covariance matrix $\mathbf{C}_{N+1}$ is constructed from the smaller matrix $\mathbf{C}_N$, the vector $\mathbf{k}_{N+1} = (C(\mathbf{x}_1, \mathbf{x}_{N+1}; \Theta), \cdots, C(\mathbf{x}_N, \mathbf{x}_{N+1}; \Theta))$ and the scalar $\kappa = C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}; \Theta)$. This partitioned form is used for calculating the inverse of $\mathbf{C}_{N+1}$ given $\mathbf{C}_N^{-1}$.

$$\hat{t}_{N+1} = \mathbf{k}_{N+1}^T \mathbf{C}_N^{-1} \mathbf{t}_N \tag{2.10}$$

$$\sigma^2_{\hat{t}_{N+1}} = \kappa - \mathbf{k}_{N+1}^T \mathbf{C}_N^{-1} \mathbf{k}_{N+1} \tag{2.11}$$

with $\kappa = C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}; \Theta)$ and $\mathbf{k}_{N+1} = (C(\mathbf{x}_1, \mathbf{x}_{N+1}; \Theta), \cdots, C(\mathbf{x}_N, \mathbf{x}_{N+1}; \Theta))$. $\hat{t}_{N+1}$ is the mean prediction at the new point and $\sigma_{\hat{t}_{N+1}}$ is the standard deviation of this prediction. It is important to notice that both equations contain $\mathbf{C}_N^{-1}$ and not $\mathbf{C}_{N+1}^{-1}$ (see Appendix A for derivations of equations 2.10 and 2.11). Consequently, for fixed data and a fixed hyperparameters of the covariance function, prediction only requires the inversion of a single $N \times N$ matrix.

2.3 The Covariance Function

As we have mentioned before, the covariance matrix and therefore the covariance function plays a pivotal role in the Gaussian process model. The predictive distributions that occur for given data sets are highly dependent on the covariance function and its hyperparameters. Let us now consider the constraints on the form of the covariance function and some examples of covariance functions for different problems.

2.3.1 Stationary Covariance Functions

There is only one constraint on the covariance matrix $\mathbf{C}_N$. The $\mathbf{C}_N$ formed using the covariance function $C(\mathbf{x}_m, \mathbf{x}_n; \Theta)$ and a set of inputs $\{\mathbf{x}_n\}_{n=1}^N$ must be positive definite for any N and any set of inputs to ensure that the distribution $P(\mathbf{t}_N | \mathbf{C}_N, \{\mathbf{x}_n\})$ is normalizable.

For the moment let us consider only stationary covariance functions, i.e., those covariance functions that are only dependent on the relative position of the two input vectors, satisfying:

$$C(\mathbf{x}_i, \mathbf{x}_i + \mathbf{h}; \Theta) \equiv C_{stat}(\mathbf{h}; \Theta) \tag{2.12}$$

Such a stationary covariance function $C_{stat}(\mathbf{h}; \Theta)$ must satisfy the positive definite constraint if, for

a positive bounded symmetric measure $G(.)$,

$$C_{stat}(\mathbf{h};\Theta) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \exp\left(i\mathbf{w}^T\mathbf{h}\right) G(d\mathbf{w}) \tag{2.13}$$

This is known as Bochner's theorem (Bochner 1979). It can easily be demonstrated by considering the following expression of positive definiteness:

$$\mathbf{r}^T\mathbf{C}\mathbf{r} > 0 \tag{2.14}$$

for any arbitrary real vector $\mathbf{r}$. In terms of a stationary covariance function this can be written as

$$\sum_{m,n} r_m r_n C_{stat}(\mathbf{x}_m - \mathbf{x}_n;\Theta) > 0 \tag{2.15}$$

Now substituting in the form of $C_{stat}(\mathbf{h})$ from equation 2.13 we obtain

$$\begin{aligned}
\sum_{m,n} r_m r_n C_{stat}(\mathbf{x}_m - \mathbf{x}_n;\Theta) &= \sum_{m,n} r_m r_n \int e^{i\mathbf{w}^T(\mathbf{x}_m - \mathbf{x}_n)} G(d\mathbf{w}) && (2.16)\\
&= \int \left(\sum_m r_m e^{i\mathbf{w}^T\mathbf{x}_m}\right)\left(\sum_n r_n e^{-i\mathbf{w}^T\mathbf{x}_n}\right) G(d\mathbf{w}) && (2.17)\\
&= \int \left|\sum_m r_m e^{i\mathbf{w}^T\mathbf{x}_m}\right|^2 G(d\mathbf{w}) && (2.18)
\end{aligned}$$

Thus $C_{stat}(\mathbf{h};\Theta)$ must be positive definite for any set of inputs $\{\mathbf{x}_n\}$ if it obeys equation 2.13 as $G(.)$ is a positive measure.

In fact equation 2.13 does not limit our choice of covariance functions drastically. For instance any Fourier transforms of $f(\mathbf{h})$ where $f(\mathbf{h}) \geq 0 \; \forall \; \mathbf{h}$ satisfies the positive definite constraint. However for many regression problems there are two properties that we wish the covariance function to embody.

1. For most regression problems we expect the correlation between two data points that are close together in input space (according to some distance measure) to be greater than the correlation between two distant data points.

2. We often find that our data has been corrupted by noise and we therefore want to incorporate some form of noise model into our covariance function.

As the sum of any positive definite functions is also positive definite we shall now split our definition of the covariance function into two parts: the first is associated with the form of the function we wish to use to model our data and the second is associated with the noise.

$$C_{stat}(\mathbf{x}_m - \mathbf{x}_n;\Theta) = \mathcal{C}_f(\mathbf{x}_m - \mathbf{x}_n;\Theta) + \delta_{mn}\theta_3 \tag{2.19}$$

One possible form for $\mathcal{C}_f$ is

$$\mathcal{C}_f(\mathbf{x}_m, \mathbf{x}_n;\Theta) = \theta_1 \exp\left\{-\frac{1}{2}\sum_{l=1}^{L} \frac{\left(x_m^{(l)} - x_n^{(l)}\right)^2}{2r_l^2}\right\} + \theta_2 \tag{2.20}$$

where $x_n^{(l)}$ is the l^{th} component of $\mathbf{x}_n$. The above expression embodies the first of the properties that we demanded of our covariance function; that points which are close in input space are strongly correlated and hence give rise to similar values of t. The definition of "close" is expressed in terms of a set of length scales $\{r_l\}$. There is a length scale corresponding to each input which characterizes the distance in that particular direction over which t is expected to vary significantly. A very large length scale would signify that the predictions made using the Gaussian process would have little or no bearing on the corresponding input. The input could be said to be insignificant. Such an interpretation is closely related to automatic relevance determination (MacKay 1994), (Neal 1996). The θ_1 hyperparameter gives the overall vertical scale relative to the mean of the Gaussian process in output space. The θ_2 hyperparameter gives the vertical uncertainty. This reflects how far we expect the mean of the function to fluctuate from the mean of the Gaussian process.

The second term in the covariance function $\delta_{mn}\theta_3$ represents the noise model. We assume the noise to be random and so do not expect any correlations between the noise on individual outputs. Hence the second term only contributes to the diagonal elements of the covariance matrix. The noise variance is given by θ_3 which is assumed for stationary covariance functions to be independent of the inputs (input dependent noise models will be discussed in the next section).

The covariance function outlined above provides a good basic form to work with and it is used extensively both in the applications given in this thesis and in research by others (Williams and Rasmussen 1996; Neal 1997). However there are many other useful forms for the stationary covariance function. For example let us consider modelling data using a function that is periodic with some known wavelength λ_l in the l^{th} input direction. A covariance function which embodies such periodicity is

$$\mathcal{C}_f(\mathbf{x}_m, \mathbf{x}_n) = \theta_1 \exp\left[-\frac{1}{2}\sum_{l=1}^{L}\left(\frac{\sin\left(\frac{\pi}{\lambda_l}(x_m^{(l)} - x_n^{(l)}\right)}{r_l}\right)^2\right] \tag{2.21}$$

Functions described using this covariance function not only have strong correlations between points that are close together in input space but also strong correlations between points that are separated from each other by a distance λ_l in the l^{th} input direction. Figure 2.3 shows some random samples drawn from $P(\mathbf{t}_N|\mathbf{C}_N, \{\mathbf{x}_n\})$ for a variety of different covariance functions.

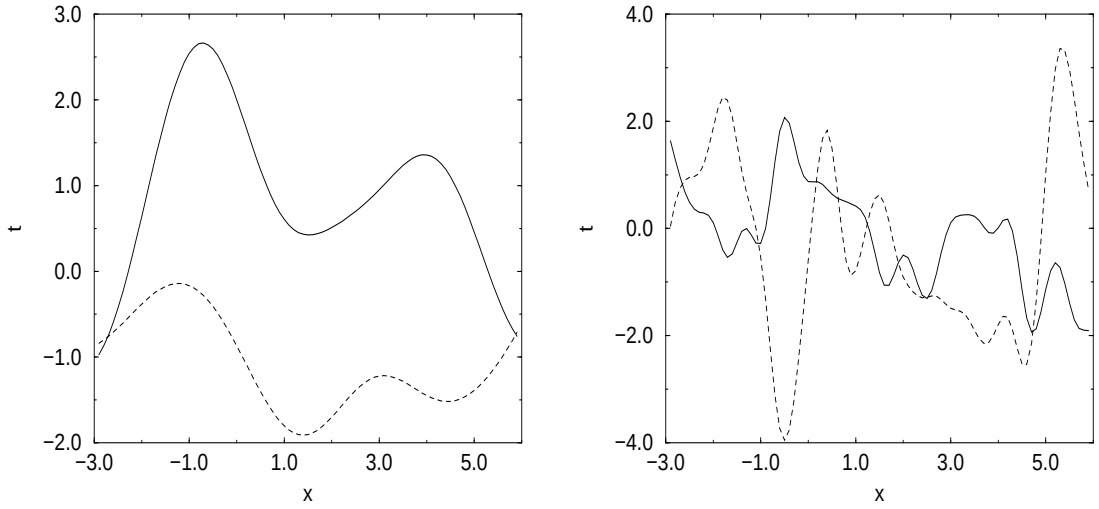
2.3.2 Non-stationary Covariance Functions

While many data sets can be effectively modelled using a stationary covariance function, there are some cases in which our covariance function must exhibit some form of non-stationarity. Unfortunately, it is difficult to construct a framework within which to generate and categorize stationary positive definite functions, let alone non-stationary ones. We therefore restrict ourselves to describing three particularly useful non-stationary forms.

Linear Term : As we move a large distance away from the data, our predictions using a covariance function such as equation 2.20 tend to a constant value. We may, however, believe that there is some linear trend in the data.

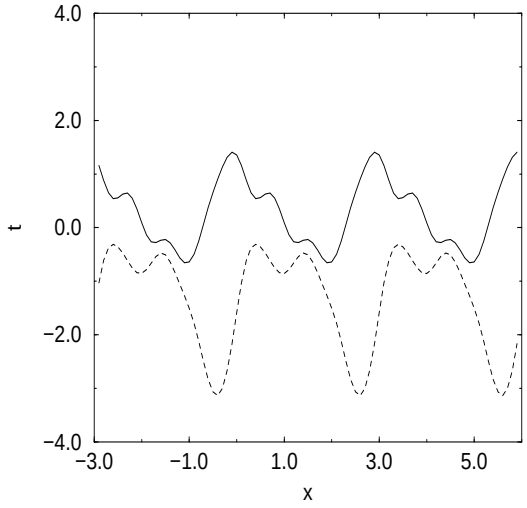
Consider the plane $y(\mathbf{x}) = \sum_l a_l x^{(l)} + c$. If the $\{a_l\}$ and c have Gaussian distributions with zero mean and variances σ_a and σ_c respectively then the plane has a covariance function

$$C_{lin}(\mathbf{x}_m, \mathbf{x}_n; \{\alpha\}) = \sum_{l=1}^{L} \sigma_a^2 x_m^{(l)} x_n^{(l)} + \sigma_c^2 \tag{2.22}$$

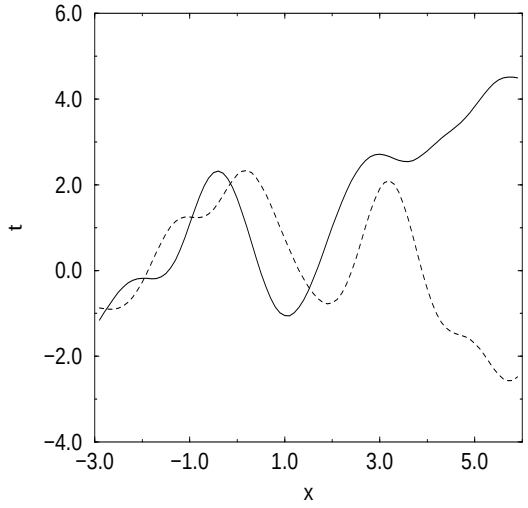


(a) $2.0 \exp\left(-\frac{(x_n - x_m)^2}{2(1.5)^2}\right)$

(b) $2.0 \exp\left(-\frac{(x_n - x_m)^2}{2(0.35)^2}\right)$



(c) $2.0 \exp\left(-\frac{\sin^2(\pi(x_n - x_m)/3.0)}{2(0.5)^2}\right)$



(d) $2.0 \exp\left(-\frac{(x_n - x_m)^2}{2(1.5)^2}\right) + x_n x_m$

Figure 2.3: Samples drawn from a Gaussian process prior : This figure shows two functions drawn from each of four Gaussian process priors with different covariance functions. The corresponding covariance function is given below each plot. The decrease in length scale from (a) to (b) produces more rapidly fluctuating functions. The periodic properties of the covariance function in (c) can clearly be seen. The covariance function in (d) contains the non-stationary term $x_n x_m$ corresponding to the covariance of a straight line (see Section 2.3.2).

Hence adding a $\sum_l \alpha_l x_m^{(l)} x_n^{(l)}$ term into the covariance function in equation 2.20 produces a linear component to the predictions. Far away from the data, this will give mean predictions lying on the plane

$$y(\mathbf{x}) = \sum_{l=1}^{L} \left(\sum_{n=1}^{N} \alpha_l x_n^{(l)} (\mathbf{C}_N^{-1} \mathbf{t}_N)_n \right) x^{(l)} + \left(\sum_{n=1}^{N} \theta_2 (\mathbf{C}_N^{-1} \mathbf{t}_N)_n \right). \tag{2.23}$$

An example of a function using the linear term can be seen in Figure 2.3(d).

Unfortunately, the length scale r_l will be correlated to the hyperparameter α_l as they both control the variability of the predictions in the l^{th} input direction. This can produce a likelihood of Θ with multiple modes which can cause problems for approaches to Gaussian processes which rely on optimization of the hyperparameters (see Section 2.4.1). A possible solution is to fit the data using only a linear term initially and then, having fixed the associated hyperparameters, introduce the other elements of the covariance function.

Input-dependent noise model : Previously we have considered noise models which are input independent. However it is perfectly reasonable to expect the noise level to vary as we cross the input space. Building such a dependence into a Gaussian process model is straightforward. We can define

$$\theta_3(\mathbf{x}_m; \Theta) = \exp\left(\sum_{j=1}^{J} \beta_j \phi_j(\mathbf{x}_m) \right) \tag{2.24}$$

where the $\{\phi_j(\mathbf{x})\}$ are a set of basis functions and the $\{\beta_j\} \in \Theta$ are appropriate coefficients. Such a noise model is non-stationary but, because it only contributes to the diagonal elements of the covariance matrix, it is positive definite.

Spatially varying length scales : The standard covariance function assumes that, in each direction, the length scale r_l is fixed. It is easy to imagine a case where this might be a poor model for the data. Hence we would like to make the $\{r_l\}$ functions of $\mathbf{x}$. We cannot simply substitute a parameterized form for $r_l(\mathbf{x})$ into equation 2.20 as this will not, generally, give us a positive definite covariance function. In Section 3.10.3, we will show that the covariance function

$$\mathcal{C}_f^{ns}(\mathbf{x}_m, \mathbf{x}_n; \Theta) = \theta_1 \prod_l \left\{ \frac{2 r_l(\mathbf{x}_m; \Theta) r_l(\mathbf{x}_n; \Theta)}{r_l^2(\mathbf{x}_m; \Theta) + r_l^2(\mathbf{x}_n; \Theta)} \right\}^{1/2} \exp\left(- \sum_l \frac{\left(x_m^{(l)} - x_n^{(l)} \right)^2}{r_l^2(\mathbf{x}_m; \Theta) + r_l^2(\mathbf{x}_n; \Theta)} \right), \tag{2.25}$$

is positive definite and has the required property of spatial variation of the length scales where $r_l(\mathbf{x}; \Theta)$ is an arbitrary parameterized function of $\mathbf{x}$. It also has the useful property that the variance is independent of $\mathbf{x}$ and equal to θ_1.

Another possible approach to modelling data generated by a function with spatially varying length scales is to transform the data into an input space in which the underlying function is stationary using a non-linear map. This non-linear map approach is also discussed in Section 3.10.3.

2.3.3 Generating Covariance Functions

So far we have only listed some widely used or useful covariance functions. We would like to be able to generate covariance functions with specific properties of our choosing. As mentioned in Section 2.3.1, the sum of any two positive definite functions is also positive definite. The same is also true of the product of two positive definite functions:

Sum of covariance functions : If $\mathcal{C}_1(.)$ and $\mathcal{C}_2(.)$ are both positive definite functions in $\Re^d$ then $\mathcal{C}_1(.) + \mathcal{C}_2(.)$ is also a positive definite function in $\Re^d$.

Product of covariance functions : If $\mathcal{C}_1(.)$ is a positive definite function in $\Re^{d_1}$ and $\mathcal{C}_2(.)$ is a positive definite functions in $\Re^{d_2}$ then $\mathcal{C}_1(.)\mathcal{C}_2(.)$ is a positive definite function in $\Re^{d_1+d_2}$.

Hence we can generate new functions using simpler covariance functions as the building blocks. For example, it is common to use a sum of several of the Gaussian terms which appear in equation 2.20, each with independent hyperparameters, in the covariance function. This gives us the possibility of modelling large scale fluctuations in one direction with one Gaussian and smaller scale fluctuations with another, i.e., producing an additive model.

To generate complicated covariance functions, we have to find the correct building blocks. Fortunately, such simple covariance functions abound in the literature. A possible alternative to a lengthy search for the right function is to make use of the analogy of Gaussian processes to fixed basis function networks (see Section 2.5). This provides a straightforward method to construct the required covariance functions, provided that a set of basis functions can be found that also have the required properties.

2.4 Determining the Hyperparameters of a Gaussian Process

We have constructed a model for our data. We now need to find a consistent way in which to deal with the undetermined hyperparameters Θ of that model. Ideally we would like to integrate over all the hyperparameters in order to make our predictions, i.e., we would like to find

$$P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.)) = \int P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.), \Theta) P(\Theta|\mathcal{D}, C(.)) d\Theta \tag{2.26}$$

where $C(.)$ represents the form of the covariance function. For arbitrary $C(.)$ this is analytically intractable. However we can obtain approximations to the posterior distribution by either approximating the average prediction over all the possible values of the hyperparameters using the most probable values of hyperparameters (Evidence maximization (MacKay 1992a)) or by performing the integration over Θ numerically using Monte Carlo methods (Williams and Rasmussen 1996; Neal 1997).

2.4.1 Evidence Maximization

Evidence maximization uses an approximation to the integral in equation 2.26 based on the most probable set of hyperparameters Θ_{MP}:

$$P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.)) \simeq P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.), \Theta_{MP}) \tag{2.27}$$

The approximation is based on the assumption that the posterior distribution over Θ, $P(\Theta|\mathcal{D}, C(.))$, is sharply peaked around Θ_{MP} relative to the variation in $P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.), \Theta)$ (see Figure 2.4). This approximation is generally good and Evidence maximization predictions are often very close to those found using the true predictive distribution (MacKay 1996).

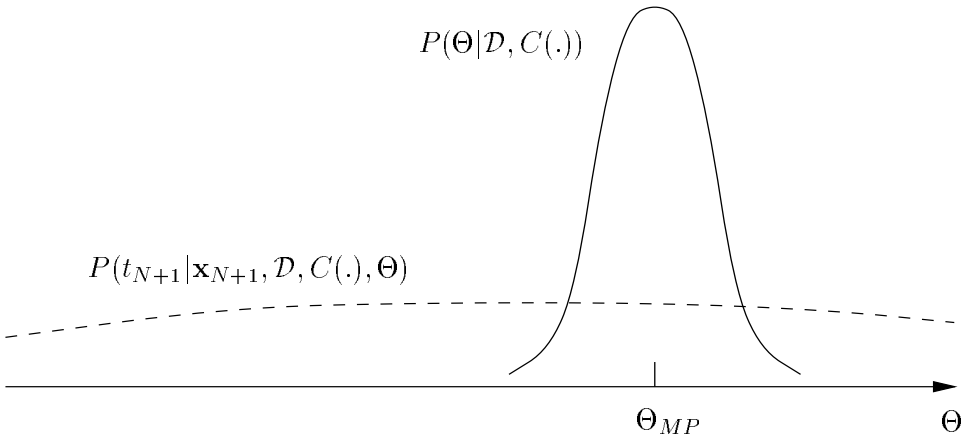


Figure 2.4: The Evidence Approximation : This figure illustrates the assumption made when using Evidence maximization. We assume that the posterior distribution over Θ, $P(\Theta|\mathcal{D}, C(.))$, is sharply peaked around Θ_{MP} relative to the variation in the predictive distribution $P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, C(.), \Theta)$.

In order to make use of Evidence maximization we need to find the most probable hyperparameters. If we can find the derivatives of the posterior distribution of Θ then we can do this using a standard gradient based optimisation routine[1] such as conjugate gradients. The posterior distribution of Θ can be written

$$P(\Theta|\mathcal{D}, C(.)) = \frac{P(\mathbf{t}_N|\{\mathbf{x}_n\}, C(.), \Theta)P(\Theta)}{P(\mathbf{t}_N|\{\mathbf{x}_n\}, C(.))} \tag{2.28}$$

The denominator is independent of Θ and shall be ignored for the time being. The two remaining terms, the likelihood of Θ and the prior on Θ, shall be considered in terms of their logs. The log likelihood $\mathcal{L}$ for a Gaussian process is

$$\mathcal{L} = -\frac{1}{2}\log\det\mathbf{C}_N - \frac{1}{2}\mathbf{t}_N^T\mathbf{C}_N^{-1}\mathbf{t}_N - \frac{N}{2}\log 2\pi \tag{2.29}$$

and its derivative with respect to a generic hyperparameter θ is

$$\frac{\partial\mathcal{L}}{\partial\theta} = -\frac{1}{2}\mathrm{tr}\left(\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta}\right) + \frac{1}{2}\mathbf{t}_N^T\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta}\mathbf{C}_N^{-1}\mathbf{t}_N \tag{2.30}$$

It is common practice to ignore the log prior term and perform a maximum likelihood optimization of the hyperparameters. However by doing so we are often allowing solutions to our problem which we do not believe are correct. The likelihood function is generically multi-modal with respect to Θ (see example given in Figure 2.5) and some of the modes often correspond to "ridiculous" solutions which sensible priors on the hyperparameters would remove.

Many of the hyperparameters of a Gaussian process are constrained to be positive. Gamma and Inverse Gamma distributions are therefore useful as priors on these hyperparameters. The Gamma distribution can be written

$$\mathrm{Ga}(\theta|m,\alpha) = \frac{(\alpha/2m)^{\alpha/2}}{?\ (\alpha/2)}\theta^{\alpha/2-1}\exp\left(-\alpha\theta/2m\right) \tag{2.31}$$

[1] A gradient based optimization routine is considered to be one that optimizes a function using only the derivatives of the function. No evaluation of the function itself need take place.

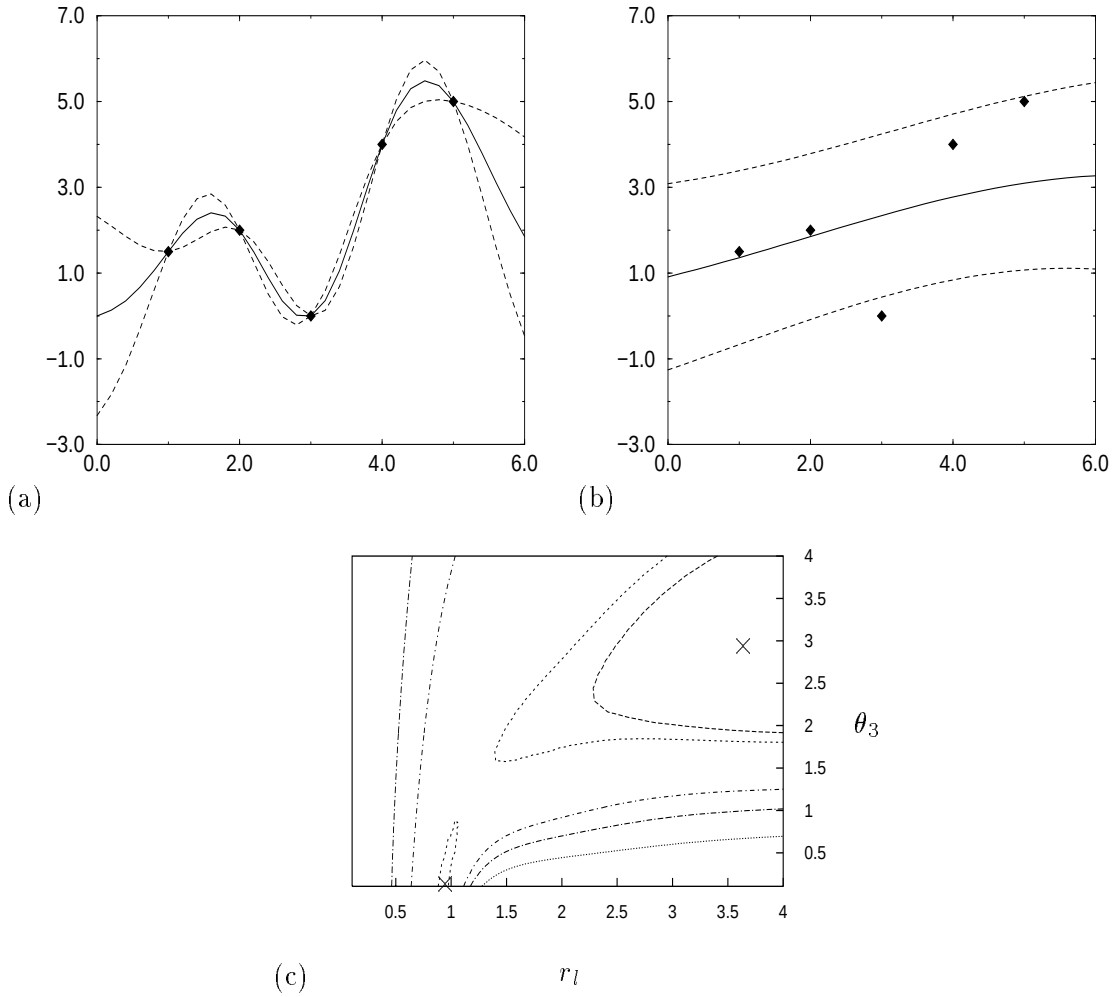


Figure 2.5: Multi-modal Likelihood Functions : This figure highlights the problems that can be encountered when using a Gaussian process with no priors on the hyperparameters. (a) shows the most probable interpolant and its 1σ error bars obtained when a Gaussian process was used to model the data shown as black diamonds on the graph. The covariance function used is given in equation 2.20. The hyperparameters of the Gaussian process were initialized with the noise level and length scale both set to low values ($\theta_3 = 0.01, r_1 = 1.0$). Following the Evidence maximization approach, the hyperparameters were then optimized using conjugate gradients. (b) shows the most probable interpolant and its 1σ error bars obtained when an identical Gaussian process with identical training data and learning procedure was given different initial conditions (high noise level and large length scale ($\theta_3 = 2.0, r_1 = 4.0$)). (c) gives a contour plot of the likelihood of Θ. The two distinct modes (one at $r_l = 0.95, \theta_3 = 0.0$ and one at $r_l = 3.5, \theta_3 = 3.0$) which give rise to the two different solutions are shown by crosses.

where m is the mean and α is a parameter which controls the shape of the distribution. The smaller the value of α, the more vague the prior. Neal (1996) describes a system for scaling the mean of the Gamma distribution with the shape parameter α in order to produce sensible priors as the number of inputs to the Gaussian process becomes large. This reflects a belief that even with a large number of inputs, only a small number of those inputs are actually relevant to the output.

The Inverse Gamma distribution can be written

$$\mathrm{Ig}(\theta|m,\alpha) = \frac{(\alpha m/2)^{\alpha/2+1}}{?\,(\alpha/2+1)}\theta^{-(\alpha/2+2)}\exp\left(-\alpha m/2\theta\right) \tag{2.32}$$

Again, m is the mean and α controls the shape of the distribution. In this case, large values of α give rise to vague priors.

Assuming that finding the derivatives of the priors is straightforward, we can now search for Θ_{MP}. However there are two problems that we need to be aware of. Firstly the posterior distribution over Θ can be multi-modal. This can mean that the Θ_{MP} that is found by the optimization routine is dependent on the initial Θ used. Secondly and perhaps most importantly each evaluation of the gradient of the log likelihood requires the evaluation of $\mathbf{C}_N^{-1}$. Any exact inversion method has an associated computational cost that is $\mathcal{O}(N^3)$ and so calculating gradients exactly becomes time consuming for large training data sets.

2.4.2 Monte Carlo Approach

The Monte Carlo approach uses sampling methods to calculate an approximation to the predictive distribution. By approximating the integral in equation 2.26 using a Markov chain, we can write

$$P(t_{N+1}|\mathbf{x}_{N+1},\mathcal{D},C(.)) \simeq \frac{1}{T}\sum_{t=1}^{T} P(t_{N+1}|\mathbf{x}_{N+1},\mathcal{D},C(.),\Theta_t) \tag{2.33}$$

where the Θ_t are samples drawn from $P(\Theta|\mathcal{D},C(.))$, the posterior distribution over Θ. Each term in the sum in the above equation is a Gaussian and so the Monte Carlo approximation to the predictive distribution is a mixture of Gaussians. The accuracy of the approximation increases as we take more samples from the posterior over Θ. Note that as we are sampling from the posterior over Θ we shall need priors on our hyperparameters $P(\Theta)$ as we did for the Evidence maximization approach.

The method that we use to sample from the posterior distribution over Θ will affect the efficiency of our approach considerably because we want the samples that we take to represent the underlying distribution from which they are taken. If we do not sample at all from a particular region of Θ space that has a high associated probability then our approximation to the integral in equation 2.26 will be poor. Hybrid Monte Carlo (Duane *et al.* 1987) is a stochastic dynamics sampling algorithm that has been used by Williams and Rasmussen (1996), Rasmussen (1996) and Neal (1997) to successfully implement Gaussian processes. It uses gradient information about the posterior distribution over Θ in order to efficiently sample from it, avoiding random walk behaviour that can affect Gibbs sampling and Metropolis approaches. Details of the algorithm can be found in Neal (1996) and Duane *et al.* (1987).

2.4.3 Evidence vs. Monte Carlo

In this thesis we shall follow the Evidence maximization approach to Gaussian processes. The problem of multi-modality that can sometimes plague an approach based on optimization of hyperparameters is, in my experience, not acute for Gaussian processes. We can express priors on

our hyperparameters that can remove multi-modality while still capturing our prior beliefs accurately. Obtaining a set of most probable hyperparameters is also an attractive feature of Evidence maximization. The interpretability of the Gaussian process model means that such a set can tell us a great deal about the data we are modelling. Also having one final set of hyperparameters means that we have only one covariance matrix to invert and store in order to make predictions. The addition of extra training data using the partitioned inverse (see Appendix A) requires us to update only one inverse covariance matrix.

Using the Monte Carlo approach, we approximate our posterior probabilities using averages over a series of samples. As long as we have asked for all the information we require before the start of the Monte Carlo simulation, we can build up the averages step by step, inverting a covariance matrix every time but without the need to store this inverse. However if we need to go back and calculate any new statistics about the data or need to incorporate some new training data into the model then, unless we have retained all the inverses, we must re-compute them at considerable expense. This being said, the Monte Carlo approach does offer us significantly more flexibility than Evidence maximization. Non-Gaussian noise models can be incorporated into Gaussian processes using the Monte Carlo approach and the performance of some of the more complex covariance functions (particularly the input-dependent noise models) is much better than when using Evidence maximization. The problem of classification using Gaussian processes (see chapter 4 for a variational approach) can be tackled using Monte Carlo methods and the extension from binary to multi-class (> 2) problems is less problematic than when using variational methods based on Evidence maximization (see chapter 5). See Neal (1997) for a review of implementing Gaussian processes using Monte Carlo methods.

Our use of Evidence maximization here in no way suggests that it is superior to the Monte Carlo approach for all problems. For smaller data sets where matrix storage is not an important issue, Rasmussen (1996) found that the Monte Carlo approach gives better results for a fixed amount of CPU time. For the larger data sets, Evidence based Gaussian processes are faster and Rasmussen (1996) found their performance to be better.

2.5 Relationship to other Models

The study of Gaussian processes for regression is far from new. Within the geostatistics field, Matheron (1963) proposed a framework for regression using optimal linear estimators which he called 'kriging' after D.G. Krige, a South African mining engineer. This framework is identical to the Gaussian process approach to regression. 'Kriging' has been developed considerably in the last thirty years (see Cressie (1993) for an excellent review) including several Bayesian treatments (Omre 1987; Kitanidis 1986). However the geostatistics approach to the Gaussian process model has concentrated mainly on low-dimensional input spaces and has has largely ignored any probabilistic interpretation of the model and any interpretation of the individual parameters of the covariance function.

The Gaussian process framework encompasses a wide range of different regression models. O'Hagan (1978) introduced an approach which is essentially similar to Gaussian processes. Generalized radial basis functions (Poggio and Girosi 1989), ARMA models (Wahba 1990) and variable metric kernel methods (Lowe 1995) are all closely related to Gaussian processes.

The present interest in the area has been initiated by the work of Neal (1996) on priors for infinite networks. Neal showed that the prior over functions defined by a neural network with one hidden layer converges to a Gaussian process as the number of hidden neurons tends to infinity for certain priors on the weights. The Bayesian interpretation of Gaussian processes was extended in Williams and Rasmussen (1996) and Neal (1997) and a comparison of Gaussian processes with other methods such as neural networks and MARS was done by Rasmussen (1996).

Relationship to Fixed Basis Function Model

Let us consider how Gaussian processes relate to fixed basis function networks. As before, let us have some noisy data $\mathcal{D} = \{\mathbf{x}_n, t_n\}_{n=1}^N$. Let

$$t_n = y(\mathbf{x}_n) + \nu_n \tag{2.34}$$

where ν_n is random Gaussian noise and let us define a fixed basis function model

$$y(\mathbf{x}_n) = \sum_{i=1}^{I} w_i \phi_i(\mathbf{x}_n), \tag{2.35}$$

where $\{\phi_i\}$ is a set of basis functions. We can write down a prior on the weights $\mathbf{w}$ and a prior on the noise

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_w} \exp\left(-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right) \tag{2.36}$$

$$P(\nu\,|\beta) = \frac{1}{Z_\nu} \exp\left(-\frac{\beta}{2}\,|\nu|^2\right) \tag{2.37}$$

where $\nu = (\nu_1, \cdots, \nu_N)$ and α and β are appropriate hyperparameters. We can then show that the joint probability of the data is given by

$$P(\mathbf{t}_N|\{\mathbf{x}_n\}, \alpha, \beta) = \frac{1}{Z_t} \exp\left(-\frac{1}{2}\mathbf{t}_N^T\mathbf{G}^{-1}\mathbf{t}_\mathrm{N}\right) \tag{2.38}$$

where

$$\mathbf{G} = \alpha^{-1}\mathbf{\Phi}\mathbf{\Phi}^\mathrm{T} + \beta^{-1}\mathbf{I} \tag{2.39}$$

with $\Phi_{ij} = \phi_j(\mathbf{x}_i)$. Comparing equation 2.38 with equation 2.6 we can see that the basis function model is equivalent to a Gaussian process. Using radial basis functions centred on the points $\{\mathbf{a}_k\}$

$$\phi_k(\mathbf{x}_i) = \exp\left(-\sum_{l=1}^{L} \frac{(x_i^{(l)} - a_k^{(l)})^2}{r_l^2}\right) \tag{2.40}$$

with set of length scales $\{r_l\}$, we obtain a covariance function which has the following data dependent part:

$$\mathcal{G}_f(\mathbf{x}_i, \mathbf{x}_j) = \sum_k \frac{1}{\alpha} \exp\left(-\sum_{l=1}^{L}\left\{\frac{(x_i^{(l)} - a_k^{(l)})^2}{r_l^2} + \frac{(x_j^{(l)} - a_k^{(l)})^2}{r_l^2}\right\}\right) \tag{2.41}$$

Note that this is not equivalent to the covariance function given in equation 2.20. However, consider the case where we have an infinite number of basis functions that are centred on an infinite number of distinct points. The summation in equation 2.41 becomes an integral and so the covariance function becomes

$$\mathcal{G}_f(\mathbf{x}_i, \mathbf{x}_j) = D \exp\left(-\frac{1}{2}\sum_{l=1}^{L} \frac{(x_i^{(l)} - x_j^{(l)})^2}{r_l^2}\right) \tag{2.42}$$

where D is a constant. This *is* equivalent to equation 2.20, showing that this form of the covariance function defines a regression model that is equivalent to a radial basis function model with an infinite number of basis functions. We do not however have to invert an infinite matrix in order to

make predictions using the Gaussian process model. The rank of the matrix that we must invert is equal to the number of data points. Also the computational cost of evaluating the elements of the matrix to be inverted is finite. Hence the Gaussian process offers us greater resolution and flexibility than a fixed basis function network without greater computational cost (assuming we do not have a very large amount of training data). The analogy to fixed basis function networks provides a good method to construct new forms of the covariance function which have certain desired properties, e.g. periodicity.

Comparison to Neural Networks

As shown by Neal (1996), there is a strong relationship between Gaussian processes and neural networks. Let us consider a feed-forward neural network with one hidden layer. The activation function for the hidden neurons is a tanh function and that for the output neuron is linear. There are J input nodes, H hidden neurons and I output neurons. We can describe the network mathematically as follows:

$$p_h = \tanh\left(\sum_{j=1}^{J} w_{hj}^{(1)} x_j\right) \tag{2.43}$$

$$y_i = \sum_{h=1}^{H} w_{ih}^{(2)} p_h + b_i \tag{2.44}$$

where b_i is the bias, $w_{ij}^{(1)}$ is a weight joining input neuron j to hidden neuron i and $w_{ih}^{(2)}$ is a weight joining hidden neuron h to output neuron i.

Following MacKay (1992d), we shall place Gaussian priors on all the weights. Each Gaussian has zero mean and standard deviation σ_1, σ_2 and σ_b for the input-hidden weights, the hidden-output weights and bias respectively. For an arbitrary input, $\mathbf{x}_n$, the expectation of the hidden units is zero as we are using tanh activation functions. Consequently, the expectation of the i^{th} output $y_i(\mathbf{x}_n)$ is also zero. The variance of $y_i(\mathbf{x}_n)$ can be written

$$E[y_i^2(\mathbf{x}_n)] = E\left[(\sum_{h=1}^{H} w_{ih}^{(2)} p_h(\mathbf{x}_n) + b_i)(\sum_{k=1}^{H} w_{ik}^{(2)} p_k(\mathbf{x}_n) + b_i)\right] \tag{2.45}$$

$$= E\left[\sum_{h,k} w_{ih}^{(2)} w_{ik}^{(2)} p_h(\mathbf{x}_n) p_k(\mathbf{x}_n)\right] + \sigma_b^2 \tag{2.46}$$

$$= E\left[\sum_{h} (w_{ih}^{(2)})^2 p_h^2(\mathbf{x}_n)\right] + \sigma_b^2 \tag{2.47}$$

as the weights are independent. As the number of hidden neurons tends to infinity we can use the Central Limit Theorem and the fact that $E\left[p_h^2(\mathbf{x}_n)\right]$ is the same for any h to obtain

$$E[y_i^2(\mathbf{x}_n)] = H\sigma_2^2 E\left[p^2(\mathbf{x}_n)\right] + \sigma_b^2 \tag{2.48}$$

We can use a similar argument to find the covariance of $y_i(\mathbf{x}_m)$ and $y_i(\mathbf{x}_n)$ for arbitrary inputs $\mathbf{x}_m$ and $\mathbf{x}_n$:

$$E[y_i(\mathbf{x}_m) y_i(\mathbf{x}_n)] = E\left[(\sum_{h=1}^{H} w_{ih}^{(2)} p_h(\mathbf{x}_m) + b_i)(\sum_{k=1}^{H} w_{ik}^{(2)} p_k(\mathbf{x}_n) + b_i)\right] \tag{2.49}$$

$$= E\left[\sum_{h,k} w_{ih}^{(2)} w_{ik}^{(2)} p_h(\mathbf{x}_m) p_k(\mathbf{x}_n)\right] + \sigma_b^2 \tag{2.50}$$

$$= H\sigma_2^2 E\left[p(\mathbf{x}_m)p(\mathbf{x}_n)\right] + \sigma_b^2 \tag{2.51}$$

We would like the variance and covariance to be finite even for an infinite number of hidden neurons. $E\left[p(\mathbf{x}_m)p(\mathbf{x}_n)\right]$ must be finite as tanh(x) is bounded above and below. If $\sigma_2 = \alpha H^{-\frac{1}{2}}$ where α is a constant then the covariance (and variance) becomes

$$E[y_i(\mathbf{x}_m)y_i(\mathbf{x}_n)] = \alpha^2 E\left[p(\mathbf{x}_m)p(\mathbf{x}_n)\right] + \sigma_b^2. \tag{2.52}$$

Thus, as the number of hidden neurons tends to infinity, the prior on $y_i(\mathbf{x}_n)$ implied by a series of Gaussian priors on the weights of a neural network is a Gaussian with zero mean and covariance

$$E[y_i(\mathbf{x}_m)y_i(\mathbf{x}_n)] = \alpha^2 C(\mathbf{x}_m, \mathbf{x}_n) + \sigma_b^2 \tag{2.53}$$

where $C(\mathbf{x}_m, \mathbf{x}_n)$ is some function dependent on σ_1 and the functional form of tanh(x). As $\mathbf{x}_m$ and $\mathbf{x}_n$ are arbitrary, equation 2.53 defines a Gaussian process. While we have assumed Gaussian priors on the weights, this is a fairly general result. It holds for any prior distributions on the input-hidden weights and the bias, for any activation function that is bounded and for any prior on the hidden to output weights with zero mean and finite variance.

Despite the derivation above, it is still not entirely clear what happens as the number of hidden units approaches infinity. At first glance we do not appear to have eliminated any of the weights and so we are still left with an infinite computation. However due to the scaling required of σ_2 in order to produce a sensible prior over functions, the magnitude of all the hidden-output weights falls to zero as the number of hidden neurons approaches infinity. The dependence of the outputs on the inputs is propagated by the sum of infinitely many infinitesimal fluctuations of the hidden-output weights. These fluctuations and hence the weights themselves are not individually significant but their joint behaviour is significant and can be characterized by equation 2.53.

Chapter 3

Implementation of Gaussian Processes

Previously we discussed the form of the Gaussian process regression model and looked at the possible methods for determining the hyperparameters of this model. Now we must carefully consider how to go about implementing the model and what the associated computational cost might be.

The most significant cost of implementing a Gaussian process model when dealing with a significant number of data points ($N > 100$) and a small number of input dimensions ($L < 10$) is the inversion of the covariance matrix $\mathbf{C}_N$. Such an inversion is required to make any predictions and, most significantly, for every evaluation of the gradient of the log posterior distribution over Θ. We must therefore think carefully about how such inversions should be performed.

Another potentially high cost of implementation is the evaluation of the elements of the covariance matrix and their derivatives. If there are many inputs ($L > 10$) or if a particularly complex form of the covariance function is being used then evaluating these elements can take significant amounts of time. We must be aware of this difficulty and should construct our covariance routines to avoid unnecessary computational expense. We should also temper any desire to incorporate a huge number of hyperparameters into our covariance function.

In this chapter we shall consider two classes of implementations. Firstly, exact implementations where inversions of $\mathbf{C}_N$ are performed to maximum precision and, secondly, implementations where we calculate approximations to the application of $\mathbf{C}_N^{-1}$ to an arbitrary vector. We shall then go on to consider several toy and real world examples of the application of Gaussian processes.

3.1 Exact Implementations

The most obvious implementation of equations 2.10, 2.11 and 2.30 is one which uses exact matrix inversion methods (such as Gauss-Jordan Elimination, LU Decomposition, Cholesky Decomposition or Singular Value Decomposition). We can obtain an explicit representation for $\mathbf{C}_N^{-1}$ and hence calculate the gradient of the log posterior with respect to Θ. A gradient based optimization algorithm can then be used to find the most probable hyperparameters or a Monte Carlo sampling method (such as Hybrid Monte Carlo (Neal 1993)) can be used to integrate over the hyperparameters. I shall refer to this as a direct implementation.

Direct implementations can run into numerical problems when the ratio of the highest and lowest eigenvalues of $\mathbf{C}_N$ becomes large (we shall discuss the eigenvalue structure of $\mathbf{C}_N$ in detail in Section 3.4). The matrix $\mathbf{C}_N$ is then said to be *ill-conditioned*. We often find, for ill-conditioned $\mathbf{C}_N$, that the elements of $\mathbf{v} = \mathbf{C}_N^{-1}\mathbf{t}_N$ are constructed from very small differences between very large numbers. Thus the likelihood of numerical inaccuracies is considerable. There are two possible solutions to this problem. Please note that both of the methods described below still have $\mathcal{O}(N^3)$ scaling of computational cost.

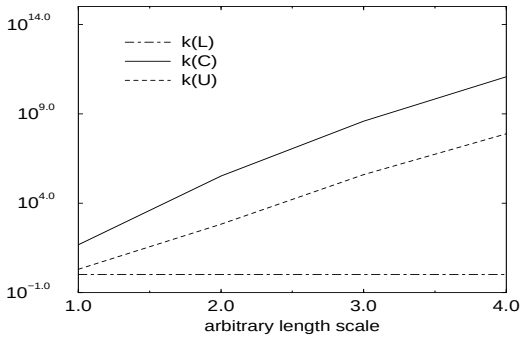


Figure 3.1: Indirect LU Implementation : A 10×10 covariance matrix $\mathbf{C}$ was generated using one-dimensional input vectors $x_j = j$ and a covariance function similar to that in equation 2.20. For a range of values of the length scale r_1, the LU-decomposition of $\mathbf{C} = \mathbf{LU}$ was calculated. The variation of the condition number $k(.)$, the ratio of the highest eigenvalue to the lowest eigenvalue, is shown in the figure for $\mathbf{C}, \mathbf{L}$ and $\mathbf{U}$. Note that the condition number of $\mathbf{C}$ is always significantly larger than that of either $\mathbf{L}$ or $\mathbf{U}$ (the condition number of $\mathbf{L}$ is always one).

3.1.1 Indirect LU Implementation

Instead of calculating $\mathbf{v} = \mathbf{C}_N^{-1}\mathbf{t}_N$ and then finding the dot product $\mathbf{k}_{N+1}^T\mathbf{v}$, we can use the LU decomposition of $\mathbf{C}_N$ to define two vectors, $\mathbf{d}_1$ and $\mathbf{d}_2$;

$$\mathbf{d}_1 = \mathbf{L}^{-1}\mathbf{t}_N \tag{3.1}$$

$$\mathbf{d}_2 = \left(\mathbf{U}^{-1}\right)^T \mathbf{k}_{N+1} \tag{3.2}$$

and then calculate

$$\hat{t}_{N+1} = \mathbf{d}_2^T\mathbf{d}_1 \tag{3.3}$$

The LU decomposition of $\mathbf{C}$ is defined as $\mathbf{C} = \mathbf{LU}$ where $\mathbf{L}$ and $\mathbf{U}$ are lower triangular and upper triangular matrices respectively. Figure 3.1 shows the variation in the ratio of the highest and lowest eigenvalues of the matrices $\mathbf{C}, \mathbf{L}$ and $\mathbf{U}$. Here we can see that $\mathbf{L}$ and $\mathbf{U}$ are less poorly conditioned than $\mathbf{C}$. Thus computations involving the application of $\mathbf{U}$ and $\mathbf{L}$ to a vector are liable to be less susceptible to numerical errors than computations involving $\mathbf{C}$. We must pay a price for this improved accuracy. Whenever we wish to calculate the value of the interpolant for a new $\mathbf{x}_{N+1}$, we must apply a matrix to a vector in contrast to the simple dot product required for the direct method.

3.1.2 Indirect SVD Implementation

Singular Value Decomposition (Press *et al.* 1988) is a common technique used for dealing with matrices which are singular or very close to singular. We will briefly discuss the method and highlight its significance in the present context.

SVD relies on a theorem of linear algebra which states that any $M \times N$ matrix $\mathbf{C}_N$, where number of rows M is greater than or equal to N, can be described in terms of an $M \times N$ column-

orthogonal matrix $\mathbf{U}$, the transpose of an $N \times N$ orthogonal matrix $\mathbf{V}$ and an $N \times N$ diagonal matrix $\mathbf{W}$,

$$\mathbf{C}_N = \mathbf{U}\mathbf{W}\mathbf{V}^T, \tag{3.4}$$

where the diagonal elements w_i of $\mathbf{W}$ are zero or positive. For the case $M = N$, $\mathbf{U}$ is square and hence is fully orthogonal. Consequently, we can write the inverse of $\mathbf{C}_N$ as

$$\mathbf{C}_N^{-1} = \mathbf{V}\mathbf{W}^{-1}\mathbf{U}^T \tag{3.5}$$

If $\mathbf{C}_N$ is ill-conditioned then some of the diagonal elements of $\mathbf{W}$ are close to zero. Hence the corresponding diagonal elements of $\mathbf{W}^{-1}$ will be very large and numerical problems can easily occur when attempting to apply $\mathbf{C}_N^{-1}$ to a vector $\mathbf{t}_N$.

To avoid these problems, let us take the following approach. If the diagonal element w_i of $\mathbf{W}$ is very close to zero then we shall replace the corresponding diagonal element in $\mathbf{W}^{-1}$ with a zero. We then use the modified inverse of $\mathbf{C}$ to calculate $\mathbf{C}_N^{-1}\mathbf{t}_N$. Replacing what should be a large number by zero seems illogical. However by zeroing the element in $\mathbf{W}^{-1}$ we are discarding a subspace of the vector space of $\mathbf{C}_N^{-1}\mathbf{t}_N$ that is significantly corrupted by round-off error. In practice, we need a criterion for deciding a threshold for elements of $\mathbf{W}$ below which we will zero the corresponding elements of $\mathbf{W}^{-1}$. As $\mathbf{C}_N$ is symmetric, the elements of $\mathbf{W}$ are the eigenvalues of the covariance matrix. If we have an idea of the minimum possible noise variance we might expect for our problem then we can zero a diagonal element of $\mathbf{W}^{-1}$ when the corresponding w_i falls below this value. Picking this minimum noise variance cannot be done blindly and requires a certain amount of knowledge about the precision of the algorithm used to calculate the elements of $\mathbf{W}$.

3.2 Approximate Implementation

All of the approaches to matrix inversion described above invert the matrix with maximum possible precision and therefore have $\mathcal{O}(N^3)$ scaling where N is the rank of the matrix. When applying a Gaussian process to a data set with a large number of data points, such scaling leads to a prohibitive computational cost. We would like to find a way to invert a matrix which does not exhibit such poor scaling with N.

Many algorithms exist for the approximate inversion of matrices. Most of these are based on conjugate gradient techniques (Hestenes and Stiefel 1952) in the form of the Lanczos algorithm (Lanczos 1950). The goal of these algorithms is to find good approximations to a function of the inverse of a matrix (often the result of applying the inverse to an arbitrary vector) while having a computational cost that scales better than $\mathcal{O}(N^3)$.

We shall concentrate on a variant of the Lanczos algorithm based upon the ideas of Skilling (1993). The basic rationale behind these ideas is that we shall restrict ourselves in the computational cost of the matrix and vector operations that we can perform. We shall only allow the application of a matrix to a vector or the calculation of a dot product. No operations that scale greater than $\mathcal{O}(N^2)$ will be allowed. Thus multiplying a matrix by a matrix and exact matrix inversion are forbidden. Using the techniques described below, we can evaluate approximations to the inverse of a matrix applied to an arbitrary vector and the trace of the inverse of a matrix while not breaking these restrictions. Furthermore we can quantify the accuracy of our approximations also within the restrictions. This then provides us with the basic tools to implement the Gaussian process framework described in Section 2.2.

3.3 Conjugate Gradient Inversion

Say we have a matrix $\mathbf{C}_N$ and a vector $\mathbf{u}$ and that we wish to calculate $\mathbf{C}_N^{-1}\mathbf{u}$. Let us define the function

$$Q(\mathbf{y}) = \mathbf{y}^T\mathbf{u} - \frac{1}{2}\mathbf{y}^T\mathbf{C}_N\mathbf{y} \tag{3.6}$$

so that the matrix $\mathbf{C}_N$ is the effective Hessian matrix of Q. At the maximum of Q the gradient with respect to $\mathbf{y}$ satisfies

$$\nabla Q_{max} = \mathbf{u} - \mathbf{C}_N\mathbf{y}_{max} = \mathbf{0} \tag{3.7}$$

and hence

$$\mathbf{y}_{max} = \mathbf{C}_N^{-1}\mathbf{u}. \tag{3.8}$$

Thus finding the maximum of Q gives us the inverse of $\mathbf{C}_N$ applied to $\mathbf{u}$. Let us consider the following method for maximizing Q. Taking an initial vector $\mathbf{y}_1 = \mathbf{0}$, we define $\mathbf{g}_1 = \nabla Q(\mathbf{y}_1)$ and $\mathbf{h}_1 = \mathbf{g}_1$. We then find the point $\mathbf{y}_2$ along the direction $\mathbf{h}_1$ which maximizes Q. It is straightforward to show that

$$\mathbf{y}_2 = \mathbf{y}_1 + \lambda_1\mathbf{h}_1 \tag{3.9}$$

where

$$\lambda_k = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{h}_k^T\mathbf{C}_N\mathbf{h}_k}. \tag{3.10}$$

We can write down $\mathbf{g}_2 = \nabla Q(\mathbf{y}_2)$ in terms of $\mathbf{g}_1$ using the recurrence relation

$$\mathbf{g}_{k+1} = \mathbf{g}_k - \lambda_k\mathbf{C}_N\mathbf{h}_k. \tag{3.11}$$

We now define $\mathbf{h}_2$ using the recurrence

$$\mathbf{h}_{k+1} = \mathbf{g}_{k+1} + \gamma_k\mathbf{h}_k, \tag{3.12}$$

where the scalar γ_k is given by

$$\gamma_k = \frac{\mathbf{g}_{k+1}.\mathbf{g}_{k+1}}{\mathbf{g}_k.\mathbf{g}_k}. \tag{3.13}$$

This ensures that the new maximization direction $\mathbf{h}_2$ is conjugate to the old gradient $\mathbf{g}_1$.

We continue this process of conjugate line searches, approaching the maximum of Q, until we have generated two sets of K vectors $\{\mathbf{g}_k\}$ and $\{\mathbf{h}_k\}$. The corresponding set of $\{\mathbf{y}_k\}$ can be calculated using the recurrence

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \lambda_k\mathbf{h}_k. \tag{3.14}$$

We now have the approximation $\mathbf{y}_K \simeq \mathbf{C}_N^{-1}\mathbf{u}$. Note that $\mathbf{C}_N\mathbf{h}_k$ is the only matrix-applied-to-vector quantity that we need to calculate for any given iteration (line search). Note also that if $K = N$ then we are guaranteed to reach the maximum of Q assuming perfect precision. The set of K orthogonal basis vectors $\{\mathbf{g}_k\}$ gives a series of successive lower bounds on $Q(\mathbf{y}_{max})$

$$0 = Q(\mathbf{y}_1) \leq Q(\mathbf{y}_2) \leq \cdots \leq Q(\mathbf{y}_K) \leq Q(\mathbf{y}_{max}) = \frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u}. \tag{3.15}$$

3.4 Convergence of the Algorithm

If we allow the conjugate gradient algorithm to continue for N iterations then we are guaranteed, machine precision allowing, to find $\mathbf{C}_N^{-1}\mathbf{u}$ exactly. Such a procedure is no more efficient than exact matrix inversion techniques as it scales $\mathcal{O}(N^3)$. However, the convergence of the conjugate gradient algorithm can be rapid, providing a good estimate $\mathbf{y}_K$ for $\mathbf{C}_N^{-1}\mathbf{u}$ in K iterations where K is significantly less than N. In this section we will discuss the rate of convergence of the conjugate gradient algorithm and how this relates to the eigenvalue structure of $\mathbf{C}_N$ (Jennings 1977). In the next section we will go on to derive a set of bounds which will allow us to monitor the convergence of the algorithm without detailed knowledge of the eigenstructure.

Let $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_N]$ be the matrix of the eigenvectors of $\mathbf{C}_N$ where $\{p_n\}_{n=1}^N$ is the set of corresponding eigenvalues. As $\mathbf{C}_N$ is a real symmetric matrix, the $\{\mathbf{q}_n\}$ form an orthogonal basis set. The gradient vectors $\mathbf{g}_k$ generated by the conjugate gradient algorithm can be transformed into this basis

$$\mathbf{s}_k = \mathbf{Q}^T\mathbf{g}_k. \tag{3.16}$$

We can also transform the recursion relations used by the algorithm into the eigenvector basis. Equation 3.11 for $k = 1$ can be written

$$\mathbf{g}_2 = (\mathbf{I} - \lambda_1\mathbf{C}_N)\mathbf{g}_1 \tag{3.17}$$

remembering that $\mathbf{g}_1 = \mathbf{h}_1$ by definition. Transforming into the eigenvector basis we obtain

$$\mathbf{s}_2 = (\mathbf{I} - \lambda_1\mathbf{P})\mathbf{s}_1 \tag{3.18}$$

where $\mathbf{P}$ is the diagonal matrix of eigenvalues. In terms of the components of $\mathbf{s}_k = (s_k^{(1)}, \cdots, s_k^{(N)})$,

$$s_2^{(n)} = (1 - \lambda_1 p_n)s_1^{(n)}. \tag{3.19}$$

Multiplying both sides of equation 3.11 by λ_{k-1}:

$$\lambda_{k-1}\mathbf{g}_{k+1} = \lambda_{k-1}\mathbf{g}_k - \lambda_{k-1}\lambda_k\mathbf{C}_N\mathbf{h}_k. \tag{3.20}$$

Using equation 3.12, we can substitute in for $\mathbf{h}_k$:

$$\lambda_{k-1}\mathbf{g}_{k+1} = \lambda_{k-1}\mathbf{g}_k - \lambda_{k-1}\lambda_k\mathbf{C}_N(\mathbf{g}_k + \gamma_{k-1}\mathbf{h}_{k-1}). \tag{3.21}$$

Hence, using equation 3.11 to substitute in for the $\mathbf{C}_N\mathbf{h}_{k-1}$ term, we find

$$\lambda_{k-1}\mathbf{g}_{k+1} + (\lambda_k\lambda_{k+1}\mathbf{C}_N - \lambda_{k-1} - \lambda_k\gamma_{k-1})\mathbf{g}_k + \lambda_k\gamma_{k-1}\mathbf{g}_{k-1} = 0. \tag{3.22}$$

Thus, in the eigenvector basis,

$$\lambda_{k-1}s_{k+1}^{(n)} + (\lambda_k\lambda_{k+1}p_n - \lambda_{k-1} - \lambda_k\gamma_{k-1})s_k^{(n)} + \lambda_k\gamma_{k-1}s_{k-1}^{(n)} = 0. \tag{3.23}$$

Considering equations 3.19 and 3.23, we can see that the elements of $\mathbf{s}_{k+1}$ can be described in terms of their initial values $\{s_1^{(n)}\}$ and a k^{th} order polynomial in p_n, i.e.,

$$s_{k+1}^{(n)} = f_k(p_n)s_1^{(n)} \tag{3.24}$$

Thinking about the $p_n = 0$ case gives us the boundary condition that $f_k(0) = 1$.

Does expressing the conjugate gradient algorithm in terms of the components of the gradient vectors $\mathbf{g}_k$ along the eigenvector directions help us understand its convergence? Let us pause to

reconsider our interpretation of the gradient vector $\mathbf{g}_k$. We can think of $\mathbf{g}_k$ not only as a gradient but also as a residual, $\mathbf{g}_k = \mathbf{u} - \mathbf{C}_N\mathbf{y}_k$, and so $\mathbf{g}_k$ gives us a measure of the distance of $\mathbf{y}_k$ from its optimal value of $\mathbf{C}_N^{-1}\mathbf{u}$. Equation 3.24 tells us that when the residual becomes zero, the polynomial $f_N(p)$ must be such that it has zeros at all the eigenvalues, i.e., $f_N(p_n) = 0$ for all n. To understand the nature of $f_k(p)$ for $k < N$, let us express $Q(\mathbf{y}_k)$ in terms of $\mathbf{s}_1, \{p_n\}$ and $f_k(p)$.

$$Q(\mathbf{y}_k) = \frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u} - \frac{1}{2}\sum_{n=1}^{N} w_n(f_k(p_n))^2 \tag{3.25}$$

where $w_n = (s_1^{(n)})^2/p_n$. At the $(k+1)^{th}$ iteration of the conjugate gradient algorithm, $f_k(p)$ is the k^{th} order polynomial which has the weighted least squares fit to a set of zeros at each of the eigenvalues with weights w_n.

Viewing the conjugate gradient algorithm as a process of polynomial fitting of zeros can help us understand how the convergence of the algorithm is affected by the eigenvalue structure of the underlying matrix. Figure 3.2 shows three different eigenstructures and the weighted least squares polynomial corresponding to the fourth iteration ($k = 5$) of the conjugate gradient algorithm. The first eigenstructure is evenly spaced and we obtain a polynomial that gives moderate error at most of the eigenvalues. The second eigenstructure contains a series of clusters of eigenvalues. There are four distinct cluster of eigenvalues and the fourth order polynomial has a zero within each cluster. Consequently the error associated with the polynomial is appreciably less than in the first case. The third case has the eigenstructure of an ill-conditioned matrix (the condition number $k(\mathbf{C}_N) = 5 \times 10^6$). Here the polynomial has an much larger associated error. It should be noted that a conjugate gradient inversion based on an ill-conditioned matrix might still have very rapid convergence if its eigenvalues are strongly clustered.

To summarize, the convergence of the conjugate gradient routine is dependent on the eigenstructure of $\mathbf{C}_N$. Tightly clustered eigenvalues of $\mathbf{C}_N$ lead to fast convergence where as eigenvalues that are spread out and that have a large condition number cause the algorithm to converge slowly.

3.4.1 Eigenstructure of the Covariance Matrix

To decide whether the Skilling approach will be suitable for the efficient inversion of covariance matrices, we need to investigate the eigenstructure of covariance matrices and how the values of the hyperparameters affect the eigenstructure. We shall restrict our discussion to covariance functions of the form of equation 2.20.

The effect of the diagonal noise term θ_3 is the simplest to analyse. Let us write $\mathbf{C}_N$ as

$$\mathbf{C}_N = \mathbf{A} + \theta_3\mathbf{I} \tag{3.26}$$

where $\mathbf{A}$ is generated using a covariance function with the noise model removed. If $\mathbf{A}$ is very poorly conditioned, adding $\theta_3\mathbf{I}$ (where θ_3 is assumed to represent a significant amount of noise, i.e., $\theta_3 = \mathcal{O}(\theta_1)$) will decrease the condition number and modify the weights $\{w_n\}$ such that the error associated with the polynomial $f_k(p)$ is significantly reduced. Neal (1997) recommends (in the context of inversion using Cholesky decomposition) always having a small noise level or 'jitter' term in the covariance function to prevent numerical problems even when the data is known to be noise free. Such an approach will also benefit approximate inversion using conjugate gradients.

θ_1 is a multiplicative scaling factor for all the eigenvalues of $\mathbf{A}$ and so has no influence on convergence rates. The θ_2 term has little effect on all but the largest eigenvalue. For large θ_2 ($\gg \theta_1$), the relationship between the largest eigenvalue and θ_2 is approximately linear. Thus the condition number of $\mathbf{A}$ increases with θ_2 although the distribution of all but the largest eigenvalue is not dramatically altered. Increasing θ_2 will therefore decrease the rate of convergence of the conjugate

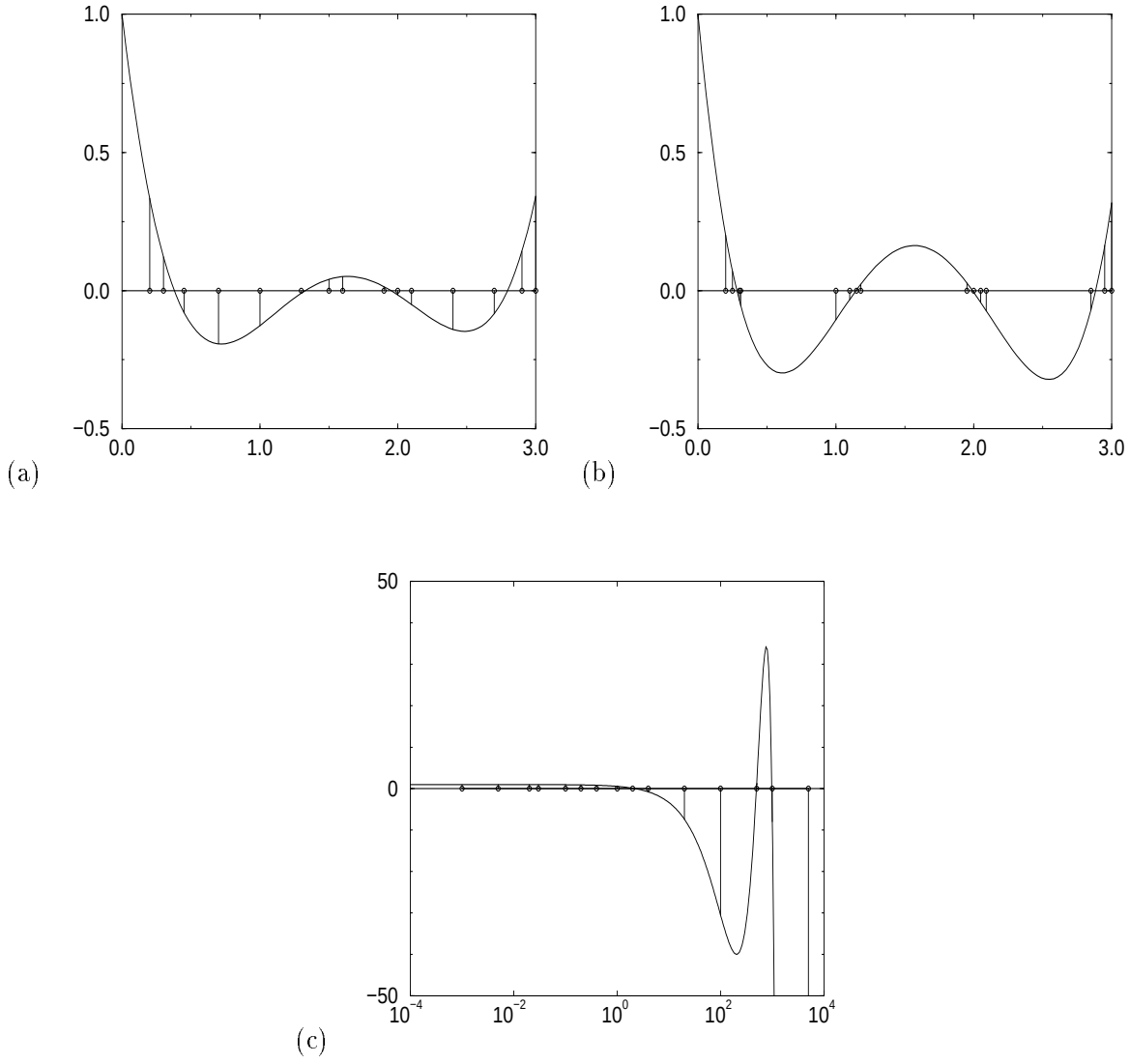


Figure 3.2: Polynomial Fitting of Zeros at Eigenvalues : (a) shows the polynomial associated with the fourth iteration of the conjugate gradient algorithm for a 15×15 matrix. Here the eigenvalues shown by the open circles along the horizontal axis are fairly equally spaced. (b) shows the same polynomial for eigenvalues that are bunched into four groups. The fit is much better than in (a). (c) shows the polynomial for a matrix which is ill-conditioned ($k(\mathbf{C}_N) = 5 \times 10^6$). Here the polynomial has a much greater error. Note that the horizontal scale in (c) is logarithmic and that the range of the vertical scale is much larger than on the other two graphs.

gradient inversion algorithm. A thorough discussion of the effect of θ_2 on the eigenstructure of $\mathbf{A}$ can be found in MacKay and Miller (1989).

To see how the length scales $\{r_l\}$ affect the eigenstructure of $\mathbf{A}$, let us consider a continuous approximation to estimate the eigenvalues of $\mathbf{A}$ for the covariance function given in equation 2.20 with $\theta_2 = 0$. For a one dimensional input space, we assume a constant spacing of the data, i.e., $x_n = \alpha n$. We also assume that N is sufficiently large that the eigenvalue equation is equivalent to the integral equation

$$\int_{-\infty}^{\infty} \theta_1 \exp\left(-\frac{\alpha^2(x-y)^2}{2r^2}\right) \tilde{q}^{(A)}(x)\mathrm{d}x = \tilde{p}^{(A)}\, \tilde{q}^{(A)}(y). \tag{3.27}$$

where $\tilde{p}^{(A)}$ and $\tilde{q}^{(A)}$ are approximations to the eigenvalues $p^{(A)}$ and eigenvectors $\mathbf{q}^{(A)}$ of $\mathbf{A}$ respectively. A solution of the above equation is

$$\tilde{q}^{(A)}(x) = \cos(k_n x) \tag{3.28}$$

$$\tilde{p}_n^{(A)} = \sqrt{2\pi}\theta_1(r/\alpha)\exp\left(-\frac{r^2 k_n^2}{2\alpha^2}\right) \tag{3.29}$$

For $k_1 = 0$, we obtain an approximation $\tilde{p}_1^{(A)}$ to the true largest eigenvalue of $\mathbf{A}$, $p_1^{(A)}$. Approximations to successively decreasing eigenvalues are given by $k_n = \pi n/N$ for $n = 0$ to $N-1$. As n increases, the continuous approximation becomes worse, making the estimates of the lower eigenvalues less good. Figure 3.3 shows these approximations for a 100×100 matrix.

All of the above results have been derived for regularly spaced one dimensional data. What can we say about the eigenspectrum associated with an arbitrary distribution of points in input space? Let us compare the eigenvalue spectrum of a 100×100 matrix with randomly spaced data with that of a matrix with regularly spaced data where the regular spacing is equal to the average nearest neighbour distance in the randomly spaced case. The two eigenspectra are broadly similar although the fractional error increases as the eigenvalues become smaller (see Figure 3.3(e) and (f)). Thus we shall use equation 3.29 to approximate the dependence of the eigenspectrum on the length scales. It should be noted that equation 3.29 is intended only to give a general feel for dependence of the eigenspectrum on the length scales and not to provide a basis for further calculations.

Thinking back to equation 3.29, we might well expect the length scale r to be $\mathcal{O}(\alpha N)$. Thus our Gaussian eigenspectrum would have a standard deviation of $\mathcal{O}(1)$. This is saying that we would only expect to obtain a couple of large eigenvalues, independent of the size of the matrix. Thus the eigenspectrum of $\mathbf{A}$ is likely to be sharply peaked. For a sharply peaked eigenspectrum of $\mathbf{A}$ and a non-zero noise level, we expect to obtain an eigenspectrum for $\mathbf{C}_N$ which has a few large eigenvalues and then many eigenvalues very close to the noise variance θ_3. Recalling the discussion of convergence of the conjugate gradient routine, we can see that covariance matrices $\mathbf{C}_N$ with covariance functions of the form of equation 2.20 and with non-zero noise will give rise to a rapid convergence for the conjugate gradient algorithm provided that the length scales are significantly larger than the average spacing between the data points.

However, when implementing a Gaussian process we often have to perform some form of search or simulation in hyperparameter space. It is highly likely that we will encounter a matrix whose condition number is very large (due to a low noise level). Such matrices may require many more iterations of the conjugate gradient algorithm for an accurate inversion than would normally be performed. So using a fixed number of iterations K to invert every matrix will either result in some wildly inaccurate estimates of $\mathbf{C}_N^{-1}\mathbf{u}$ or in far too many iterations being used for the well-conditioned matrices leading to significant computational cost. So it is vital that we have a method

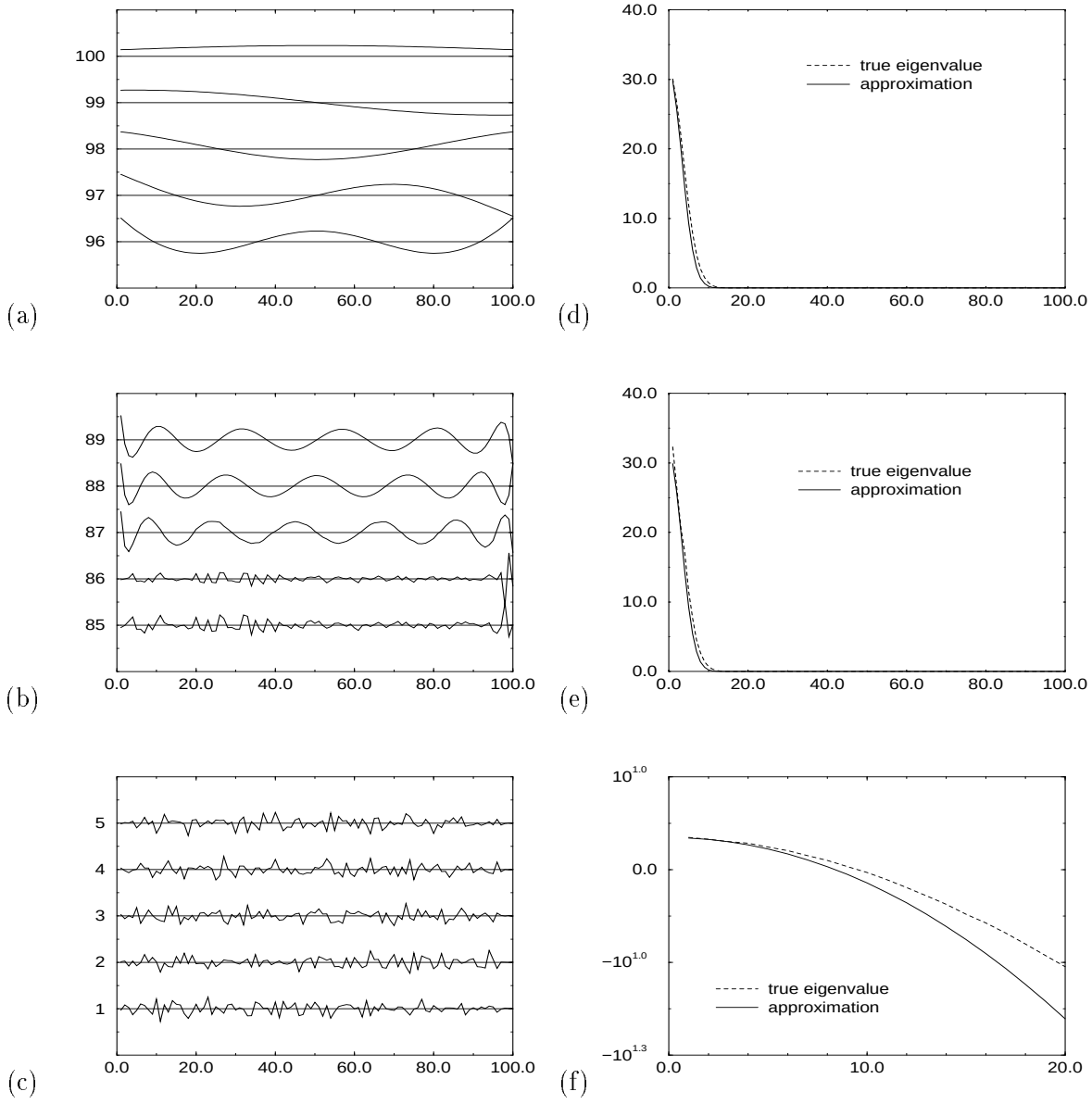


Figure 3.3: Approximations to the Eigenspectrum : (a),(b) and (c) show the eigenvectors associated with three different ranges of eigenvalues from a 100×100 matrix with regularly spaced data. Each eigenvector is labelled on the vertical axis, 1 being associated with the lowest eigenvalue and 100 being associated with the highest. The horizontal scale is the continuous approximation to position in the eigenvector. The true eigenvalue spectrum and the predicted spectrum using the continuous approximation can be seen in (d). (e) shows the true eigenvalue spectrum of a matrix with randomly spaced data compared with the predicted spectrum using the continuous approximation with a spacing equal to the mean nearest neighbour distance in the randomly spaced case. The larger eigenvalues are predicted well but, as the log plot of continuous approximation to true eigenvalue spectrum in (f) shows, the predictions for the lower eigenvalues are increasingly poor.

for identifying when the conjugate gradient algorithm is having difficulties converging so that we can adapt K accordingly.

3.5 Upper Bound on $Q_{\max}$

Using the Skilling method we already have a sequence of increasing lower bounds, $Q(\mathbf{y}_k)$, on $Q_{\max} \equiv Q(\mathbf{y}_{\max})$ (see equation 3.15). We can find a similar sequence of decreasing upper bounds for certain forms of $\mathbf{C}_N$. Given upper and lower bounds on $Q_{\max}$, we can then monitor the convergence of the inversion algorithm and prevent inaccurate estimates of $\mathbf{C}_N^{-1}\mathbf{u}$.

Let us assume $\mathbf{C}_N$ can be written

$$\mathbf{C}_N = \mathbf{A} + \theta_3 \mathbf{I}, \tag{3.30}$$

where $\mathbf{A}$ is symmetric and positive semi-definite and $\theta_3 \geq 0$. Let us also construct the function

$$Q^*(\mathbf{y}^*) = (\mathbf{y}^*)^T \mathbf{A}\,\mathbf{u} - \frac{1}{2}(\mathbf{y}^*)^T \mathbf{C}_N \mathbf{A}\mathbf{y}^*. \tag{3.31}$$

Consider the maximization of Q^* with respect to $\mathbf{A}^{\frac{1}{2}}\mathbf{y}^*$, i.e., the maximization of

$$Q^*(\mathbf{x}) = \mathbf{x}^T\mathbf{v} - \frac{1}{2}\mathbf{x}^T\mathbf{C}_N\mathbf{x} \tag{3.32}$$

with respect to $\mathbf{x} = \mathbf{A}^{\frac{1}{2}}\mathbf{y}^*$ where $\mathbf{v} = \mathbf{A}^{\frac{1}{2}}\mathbf{u}$. This mimics the maximization of Q except that $\mathbf{u}$ is replaced by $\mathbf{v}$. Consequently we can describe the maximization of Q^* with respect to $\mathbf{y}^*$ in terms of the conjugate gradient equations with modified λ_k and γ_k:

$$\begin{aligned}
\mathbf{h}^*_{k+1} &= \mathbf{g}^*_{k+1} + \gamma^*_k \mathbf{h}^*_k & (3.33)\\
\mathbf{g}^*_{k+1} &= \mathbf{g}^*_k - \lambda^*_k \mathbf{C}_N \mathbf{h}^*_k & (3.34)\\
\gamma^*_k &= \frac{(\mathbf{g}^*_{k+1})^T \mathbf{A}\mathbf{g}^*_{k+1}}{(\mathbf{g}^*_k)^T \mathbf{A}\mathbf{g}^*_k} & (3.35)\\
\lambda^*_k &= \frac{(\mathbf{g}^*_k)^T \mathbf{A}\mathbf{g}^*_k}{(\mathbf{g}^*_k)^T \mathbf{A}\mathbf{C}_N\mathbf{h}^*_k} & (3.36)
\end{aligned}$$

and hence

$$\mathbf{y}^*_{k+1} = \mathbf{y}^*_k + \lambda^*_k \mathbf{h}^*_k. \tag{3.37}$$

The above equations are identical to those derived for Q except that λ^*_k and γ^*_k contain extra factors of $\mathbf{A}$. Note that $\mathbf{A}\mathbf{g}^*_k$ and $\mathbf{C}_N\mathbf{h}^*_k$ are the only matrix-applied-to-vector quantities that we need to calculate for any given iteration. This has been achieved by using a slightly different form for λ^*_k than that used for λ_k. However, using equations 3.12 and 3.11, we can show that

$$\lambda_k = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{h}_k^T\mathbf{C}_N\mathbf{h}_k} = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T\mathbf{C}_N\mathbf{h}_k} \tag{3.38}$$

Starting with $\mathbf{y}^*_1 = \mathbf{0}$ and $\mathbf{g}^*_1 = \mathbf{h}^*_1 = \mathbf{u}$, we can now derive a new sequence of lower bounds

$$0 = Q^*(\mathbf{y}^*_1) \leq Q^*(\mathbf{y}^*_2) \leq \cdots \leq Q^*(\mathbf{y}^*_K) \leq Q^*(\mathbf{y}^*_{\max}) = \frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{A}\,\mathbf{u} \tag{3.39}$$

Hence, as

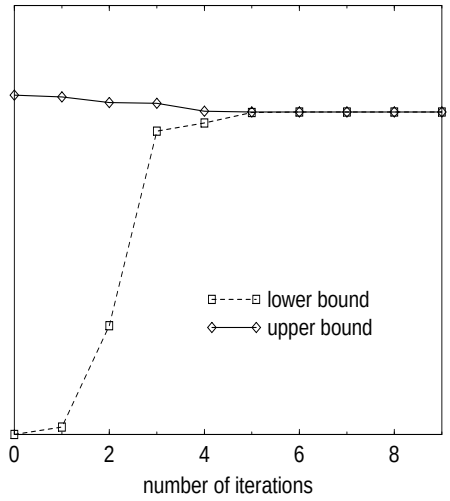


Figure 3.4: Variation of lower and upper bounds on $Q_{\max}$: This figure shows a typical variation of the upper and lower bounds on $Q_{\max}$ with the number of iterations of the conjugate gradient inversion algorithm. Notice that the plot is not symmetrical about $Q_{\max}$. However the relative convergence rates for the upper and lower bounds are the same as subspaces for the optimization of Q and Q^* are identical.

$$Q^*_{\max} + \theta_3 Q_{\max} = \frac{1}{2}\mathbf{u}^T \mathbf{C}_N^{-1} \left(\mathbf{A} + \theta_3 \mathbf{I}\right) \mathbf{u} = \frac{1}{2}\mathbf{u}^T \mathbf{u} \tag{3.40}$$

and $\mathbf{u}$ is known before we start the maximization, we can construct progressively decreasing upper bounds for $Q_{\max}$:

$$Q(\mathbf{y}_K) \;\leq\; Q_{\max} \;\leq\; \frac{1}{\theta_3}\left(\frac{1}{2}\mathbf{u}^T\mathbf{u} - Q^*(\mathbf{y}^*_K)\right) \tag{3.41}$$

An example of the variation of the bounds can be seen in Figure 3.4.

To calculate the bounds, we perform two parallel conjugate gradient optimizations to maximize Q and Q^*, keeping track of $\mathbf{y}_k$ and $\mathbf{y}^*_K$ as we go along. Thus each iteration requires $\mathcal{O}(3N^2)$ operations. As the computational cost of evaluating the bounds is quite significant ($\mathcal{O}(3N^2)$ operations), we only evaluate the bounds after K iterations of the conjugate gradient algorithm. Then we calculate the ratio $\Delta Q_K / Q(\mathbf{y}_K)$ where ΔQ_K is the difference between the upper and lower bounds (not between Q and Q^*) at the K^{th} iteration. If this ratio has fallen below some desired level then we terminate the algorithm. If the ratio has not dropped below the desired level then we perform a further K iterations and continue until the level has been reached.

3.6 Tridiagonal Representation of Conjugate Gradients

As I mentioned in the previous section, there is a fundamental relationship between the optimization of Q and the optimization of Q^*. Is it, therefore, necessary to perform two parallel conjugate gradient optimization? Can we not maximize both Q and Q^* using only one optimization? In

order to answer these questions, let us express the inversion problem in a different manner by considering the transformation of $\mathbf{C}_N$ into the basis defined by

$$\mathbf{e}_k = \frac{\mathbf{g}_k}{|\mathbf{g}_k|} \text{ for } k = 1 \cdots K \tag{3.42}$$

In order to see the advantages of this change of basis, recall equation 3.11

$$\mathbf{g}_{k+1} = \mathbf{g}_k - \lambda_k \mathbf{C}_N \mathbf{h}_k$$

Taking the dot product of this expression with $\mathbf{g}_{k+q}$ where q is an integer,

$$\mathbf{g}_{k+q}^T \mathbf{g}_{k+1} = \mathbf{g}_{k+q}^T \mathbf{g}_k - \lambda_k \mathbf{g}_{k+q}^T \mathbf{C}_N \mathbf{h}_k \tag{3.43}$$

Using equation 3.12,

$$\lambda_k \mathbf{g}_{k+q}^T \mathbf{C}_N \mathbf{g}_k = \mathbf{g}_{k+q}^T \mathbf{g}_k - \mathbf{g}_{k+q}^T \mathbf{g}_{k+1} - \lambda_k \gamma_{k-1} \mathbf{g}_{k+q}^T \mathbf{C}_N \mathbf{h}_{k-1} \tag{3.44}$$

Combining equation 3.43 and 3.44 and using the fact that all gradient vectors are orthogonal, we have

$$\mathbf{e}_{k+q}^T \mathbf{C}_N \mathbf{e}_k = \begin{cases} \frac{1}{\lambda_k} + \frac{\gamma_{k-1}}{\lambda_{k-1}} & \text{if } q = 0 \\ -\frac{1}{\lambda_k} \frac{|\mathbf{g}_{k+1}|}{|\mathbf{g}_k|} & \text{if } q = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.45}$$

Hence at the K^{th} iteration of the conjugate gradient maximization of $Q(\mathbf{y})$, $\mathbf{C}_N$ can be written $\mathbf{C}_N = \mathbf{E}_K \mathbf{T}_K^{-1} \mathbf{E}_K^T$ where $\mathbf{T}_K$ is a $K \times K$ tridiagonal matrix with elements given by equation 3.45 and $\mathbf{E}_K$ is the $N \times K$ matrix of vectors $\{\mathbf{e}_i\}$. We can use the tridiagonal representation to calculate $\mathbf{y}_K$ and the lower bound on $Q_{\max}$:

$$\mathbf{y}_K = \mathbf{E}_K \mathbf{T}_K^{-1} \mathbf{E}_K^T \mathbf{u} \tag{3.46}$$

$$Q(\mathbf{y}_K) = \frac{1}{2} \mathbf{u}^T \mathbf{E}_K \mathbf{T}_K^{-1} \mathbf{E}_K^T \mathbf{u} \tag{3.47}$$

Using the gradient vector basis, we no longer have to keep track of $\mathbf{y}_K$ at every iteration. We simply have to calculate the new element of $\hat{\mathbf{u}}_K = \mathbf{E}_K^T \mathbf{u}$ and the new elements of $\mathbf{T}_K$. The new element of $\hat{\mathbf{u}}_K$ will be zero for $K \neq 1$ as $\mathbf{g}_1 = \mathbf{u}$ and all the gradient vectors are orthogonal. Evaluating the lower bound involves the inversion of $\mathbf{T}_K$ which, as it is tridiagonal, takes only $\mathcal{O}(2N)$ operations.

How can we use this new tridiagonal representation to calculate the upper bound ? To answer this question we need to examine each iteration of the conjugate gradient algorithm more closely. Every iteration is equivalent to optimizing Q with respect to $\mathbf{y}$ where $\mathbf{y}$ is constrained in lie in a subspace $\mathcal{S}_K$. The subspaces can be defined in terms of the gradient vectors which, being orthogonal, form a convenient basis. $\mathbf{y}_K$ must lie in subspace $\mathcal{S}_K$ which has dimension K. The next gradient vector $\mathbf{g}_{K+1}$, being orthogonal to all previous gradient vectors, cannot lie in $\mathcal{S}_K$ and hence $\mathcal{S}_{K+1}$ must have dimension $K+1$. The matrix $\mathbf{E}_K$ can be thought of as transforming a K dimensional vector representing a point in $\mathcal{S}_K$ into the N dimensional space occupied by $\mathbf{u}$. Equation 3.46 shows that $\mathbf{y}_K$ can be described in terms of a K dimensional vector $\hat{\mathbf{y}}_K = \mathbf{T}_K^{-1} \mathbf{E}_K^T \mathbf{u}$ transformed into N dimensions by $\mathbf{E}_K$.

The subspaces $\mathcal{S}_K$ and $\mathcal{S}_K^*$ created by the optimizations of Q and Q^* respectively are identical. We can therefore use the gradient vectors created by the optimization of Q to find the maximum of Q^*. Consider the K^{th} iteration of the optimization of Q^*. We are attempting to maximize $Q^*(\mathbf{y}_K^*)$ where $\mathbf{y}_K^*$ is chosen from the K dimensional subspace $\mathcal{S}_K$ such that $\mathbf{y}_K^* = \mathbf{E}_K \hat{\mathbf{y}}_K^*$. We can write

$$Q^*(\mathbf{y}_K) = (\mathbf{y}_K^*)^T \mathbf{A}\mathbf{u} - \frac{1}{2}(\mathbf{y}_K^*)^T \mathbf{C}_N \mathbf{A}\mathbf{y}_K^* \tag{3.48}$$

$$= (\hat{\mathbf{y}}_K^*)^T \mathbf{E}_K^T \mathbf{A}\mathbf{E}_K \hat{\mathbf{u}}_K - \frac{1}{2}(\hat{\mathbf{y}}_K^*)^T \mathbf{E}_K^T \mathbf{C}_N \mathbf{A}\mathbf{E}_K \hat{\mathbf{y}}_K^*. \tag{3.49}$$

At the maximum in $\mathcal{S}_K$, the gradient of Q^* with respect to $\hat{\mathbf{y}}_K^*$ must be zero. Therefore

$$\left(\mathbf{E}_K^T \mathbf{C}_N \mathbf{C}_N \mathbf{E}_K - \theta_3 \mathbf{E}_K^T \mathbf{C}_N \mathbf{E}_K\right) \hat{\mathbf{y}}_K^* = \left(\mathbf{E}_K^T \mathbf{C}_N \mathbf{E}_K - \theta_3 \mathbf{I}\right) \hat{\mathbf{u}}_K \tag{3.50}$$

The $\mathbf{E}_K^T \mathbf{C}_N \mathbf{E}_K$ terms in the above equation give rise to the tridiagonal matrix $\mathbf{T}_K$. Only the $\mathbf{E}_K^T \mathbf{C}_N \mathbf{C}_N \mathbf{E}_K$ term in the above equation poses us any difficulties. To evaluate this term, we will use the following equation;

$$\mathbf{C}_N \mathbf{E}_K = \mathbf{E}_K \mathbf{T}_K + \left\{ \frac{\mathbf{h}_{K+1}^T \mathbf{C}_N \mathbf{h}_{K+1}}{|\mathbf{g}_{K+1}|^{\frac{3}{2}}} \right\} \mathbf{g}_{K+1} \left(\mathbf{E}_K^T \mathbf{g}_K\right)^T \tag{3.51}$$

When first considered by Hestenes and Stiefel (1952), the conjugate gradient algorithm was viewed as repeated applications of the matrix $\mathbf{C}_N$ to $\mathbf{u}$ in order to find $\mathbf{C}_N^{-1}\mathbf{u}$. Starting from the initial vector $\mathbf{u}$, the subspaces $\mathcal{S}_K$ expand by application of $\mathbf{C}_N$ to $\mathbf{u}$, i.e.,

$$\mathcal{S}_K = \text{span}\{\mathbf{u}, \mathbf{C}_N\mathbf{u}, \mathbf{C}_N^2\mathbf{u}, \cdots, \mathbf{C}_N^{K-1}\mathbf{u}\} \tag{3.52}$$

The relationship between the above equation and the previous definition of $\mathcal{S}_K$ as the subspace spanned by the gradient vectors can be seen by considering the construction of $\mathbf{y}_K$ in equation 3.14. Equation 3.51 is describing the transition from $\mathcal{S}_K$ to $\mathcal{S}_{K+1}$ caused by the application of $\mathbf{C}_N$. The second term on the right hand side of the equation denotes the expansion of the subspace necessary if the new gradient $\mathbf{g}_{K+1}$ is not contained within $\mathcal{S}_K$. To explain further, consider a vector $\mathbf{y} = \mathbf{E}_K\hat{\mathbf{y}}$ that is contained in $\mathcal{S}_K$. Applying $\mathbf{C}_N$ to $\mathbf{y}$ will either leave $\mathbf{y}$ in $\mathcal{S}_K$ ($\mathbf{g}_K^T\mathbf{y} = 0$) or will project $\mathbf{y}$ out of $\mathcal{S}_K$ ($\mathbf{g}_K^T\mathbf{y} \neq 0$). In the latter case, we must construct a larger subspace, $\mathcal{S}_{K+1}$, to contain $\mathbf{C}_N\mathbf{y}$. The difference between $\mathcal{S}_K$ and $\mathcal{S}_{K+1}$ is $\mathbf{g}_{K+1}$ and hence equation 3.51 follows.

Using equation 3.51 to give us an expression for $\mathbf{C}_N\mathbf{E}_K$, equation 3.50 becomes

$$\mathbf{D}_K^T \overline{\mathbf{T}}_K \mathbf{D}_K \hat{\mathbf{y}}_K^* = (\mathbf{T}_K - \theta_3 \mathbf{I})\hat{\mathbf{u}}_K \tag{3.53}$$

where $\mathbf{D}_K$ is an upper bi-diagonal matrix

$$\mathbf{D}_K = \begin{pmatrix} 1/|\mathbf{g}_1| & -|\mathbf{g}_2|/|\mathbf{g}_1|^2 & \cdots & \cdots & 0 \\ 0 & 1/|\mathbf{g}_2| & -|\mathbf{g}_3|/|\mathbf{g}_2|^2 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1/|\mathbf{g}_{K-1}| & -|\mathbf{g}_K|/|\mathbf{g}_{K-1}|^2 \\ 0 & 0 & \cdots & 0 & 1/|\mathbf{g}_K| \end{pmatrix} \tag{3.54}$$

and $\overline{\mathbf{T}}_K$ is a diagonal symmetric matrix with elements

$$(\overline{\mathbf{T}}_K)_{kk} = \frac{1}{\lambda_k^2}(1+\gamma_k) - \theta_3 \frac{|\mathbf{g}_k|^2}{\lambda_k}$$

$$(\overline{\mathbf{T}}_K)_{k(k+1)} = -\frac{|\mathbf{g}_{k+1}|^2}{\lambda_k \lambda_{k+1}}$$

Note that to evaluate the elements of $\overline{\mathbf{T}}_K$, we need the first $K+1$ gradient vectors.

We can now write down an expression for $\hat{\mathbf{y}}_K^*$ and hence, by substituting this into equation 3.49, find the following expression for $Q^*(\mathbf{y}_K^*)$:

$$Q^*(\mathbf{y}_K^*) = \frac{1}{2}\hat{\mathbf{v}}_K^T \overline{\mathbf{T}}_K^{-1} \hat{\mathbf{v}}_K \tag{3.55}$$

where

$$\hat{\mathbf{v}}_K = (\mathbf{D}_K^T)^{-1}(\mathbf{T}_K - \theta_3 \mathbf{I})\,\hat{\mathbf{u}}_K. \tag{3.56}$$

While the above expression looks formidable, the elements of $\hat{\mathbf{v}}_K$ have a simple form.

$$\hat{\mathbf{v}}_K^T = (|\mathbf{g}_1|^2(1/\lambda_1 - \theta_3), -\theta_3|\mathbf{g}_2|^2, \cdots, -\theta_3|\mathbf{g}_K|^2) \tag{3.57}$$

We do not need to keep track of $\mathbf{y}_K^*$ in order to calculate $Q^*(\mathbf{y}_K^*)$. At each iteration we simply need to calculate the new elements of $\overline{\mathbf{T}}_K$ and the new element of $\hat{\mathbf{v}}_K$ and invert one tridiagonal matrix ($\mathcal{O}(2N)$ operations). We can then use equation 3.55 to calculate the upper bound on $\frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u}$.

We have now derived a tridiagonal approach to calculating $Q(\mathbf{y}_K)$ and $Q^*(\mathbf{y}_K^*)$ and hence the upper and lower bounds on $\frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u}$. This procedure requires the inversion of two tridiagonal matrices taking $\mathcal{O}(4N)$ operations. We only need to perform one conjugate gradient algorithm (for $Q(\mathbf{y})$) which requires only one application of a matrix to a vector per iteration. We can also calculate the bounds at every iteration for minimal cost and then use the ratio $\Delta Q_K/Q(\mathbf{y}_K)$, as before, to decide when to terminate the algorithm. If we do decide to terminate the algorithm, we can then use equation 3.46 to find our best estimate of $\mathbf{C}_N^{-1}\mathbf{u}$. Thus the tridiagonal approach not only is less expensive per iteration than the double conjugate gradient algorithm approach but it also provides us with cheap bounds at every iteration so that we perform no more iterations than are strictly necessary.

3.7 Imperfect Orthogonalization and Optimizations based on A

There is a problem that traditionally occurs with Lanczos algorithms that we have not addressed yet. As the number of iterations increases, the gradient vectors cease to be perfectly orthogonal due to roundoff errors. Initially such errors are a small factor. However, when the algorithm comes close to the true solution and the magnitude of the gradient becomes small the departure from perfect orthogonalization becomes far more pronounced.

The error in the orthogonalization can be expressed in terms of the dot product of the normalized vectors $\mathbf{e}_k$ and $\mathbf{e}_{k+1}$ (Golub and Loan 1990):

$$|\mathbf{e}_{k+1}^T\mathbf{e}_k| = \frac{|\mathbf{g}_{k+1}^T\mathbf{e}_k| + 10^{1-p}||\mathbf{C}_N||_2}{|\mathbf{g}_{k+1}|} \tag{3.58}$$

where p is the precision. $||\mathbf{C}_N||_2$ is the 2-norm of $\mathbf{C}_N$ which is equal to the largest eigenvalue of $\mathbf{C}_N$. Thus the error in orthogonalization can be large for small $|\mathbf{g}_{k+1}|$ even when $|\mathbf{g}_{k+1}^T\mathbf{e}_k| = 0$. For matrices with a few large eigenvalues and then numerous small eigenvalues, all tightly bunch together, convergence of the optimization is rapid. The magnitude of the gradient becomes small very quickly and significant departures from perfect orthogonalization can be introduced in a single iteration.

When explicitly keeping track of $\mathbf{y}_K$ and $\mathbf{y}_K^*$, the roundoff errors have little effect on the results. However, the tridiagonal method relies much more heavily on perfect orthogonalization and the algorithm can converge to the correct answer and then drift significantly away from it. This is due not to the accumulation of roundoff errors but to small values of the denominator in equation 3.13.

The termination criterion defined above using the upper and lower bounds is effective at stopping the algorithm before it drifts away from the correct solution.

However there is a further improvement we can make to our algorithm to prevent problems caused by imperfect orthogonalization. Previously we have concentrated on optimizations based on $\mathbf{C}_N$. The eigenvalue structure of $\mathbf{C}_N$ is such that, for a non-zero noise level, we expect to get a few large eigenvalues and then many eigenvalues bunched around θ_3. Let us instead consider optimizations based on $\mathbf{A}$, i.e., the optimization of

$$Q^{(A)}(\mathbf{y}) = \mathbf{y}^T\mathbf{u} - \frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} \tag{3.59}$$

with respect to $\mathbf{y}$. This might, at first, sound like a remarkably bad idea. $\mathbf{A}$ is likely to be very poorly conditioned and so the convergence of the conjugate gradient algorithm will be very slow. However, the subspaces defined by the optimization of $\mathbf{A}$ are the same as those for the optimization based on $\mathbf{C}_N$. Thus we can calculate bounds on $\frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u}$ using the gradients derived from the optimization based on $\mathbf{A}$. The convergence of these bounds will be exactly the same as if we had based the optimization on $\mathbf{C}_N$. However, as the optimization of $\mathbf{A}$ converges very slowly, the magnitude of the gradient vectors does not become small and round off errors do not cause problems.

Let us discuss in detail how to use the gradient vectors from the optimization based on $\mathbf{A}$ to calculate lower and upper bounds on $\frac{1}{2}\mathbf{u}\mathbf{C}_N^{-1}\mathbf{u}$. Following the procedure described in Section 3.6 for $\mathbf{A}$ instead of $\mathbf{C}_N$, we obtain a tridiagonal representation of $\mathbf{A}$, $\mathbf{T}_K^{(A)}$. The estimate of $\mathbf{C}_N^{-1}\mathbf{u}$ and the lower bound on $\frac{1}{2}\mathbf{u}\mathbf{C}_N^{-1}\mathbf{u}$ can be found by simply adding $\theta_3\mathbf{I}$ to $\mathbf{T}_K^{(A)}$:

$$\mathbf{y}_K = \mathbf{E}_K^{(A)}(\mathbf{T}_K^{(A)} + \theta_3\mathbf{I})^{-1}(\mathbf{E}_K^{(A)})^T\mathbf{u} \tag{3.60}$$

$$Q(\mathbf{y}_K) = \frac{1}{2}\mathbf{u}^T\mathbf{E}_K^{(A)}(\mathbf{T}_K^{(A)} + \theta_3\mathbf{I})^{-1}(\mathbf{E}_K^{(A)})^T\mathbf{u} \tag{3.61}$$

where $\mathbf{E}_K^{(A)}$ is the matrix of gradient vectors generated using $\mathbf{A}$.

In order to calculate Q^*, consider equation 3.50. We can re-write this equation in terms of $\mathbf{A}$ and $\mathbf{T}_K^{(A)}$. Then, using the version of equation 3.51 appropriate for optimizations based on $\mathbf{A}$, we can write down an expression for Q^*. Explicitly,

$$Q^*(\mathbf{y}_K) = \frac{1}{2}(\hat{\mathbf{v}}_K^{(A)})^T(\overline{\mathbf{T}}_K^{(A)})^{-1}\hat{\mathbf{v}}_K^{(A)} \tag{3.62}$$

where

$$(\overline{\mathbf{T}}_K^{(A)})_{kk} = \frac{1}{(\lambda_k^{(A)})^2}\left(1+\gamma_k^{(A)}\right) + \theta_3\frac{|\mathbf{g}_k^{(A)}|^2}{\lambda_k^{(A)}}$$

$$(\overline{\mathbf{T}}_K^{(A)})_{k(k+1)} = -\frac{|\mathbf{g}_{k+1}^{(A)}|^2}{\lambda_k^{(A)}\lambda_{k+1}^{(A)}}$$

and

$$\hat{\mathbf{v}}_K^{(A)} = \left((|\mathbf{g}_1^{(A)}|^2/\lambda_1^{(A)}), 0, 0, \cdots\right) \tag{3.63}$$

Note the only difference between $\overline{\mathbf{T}}_K^{(A)}$ and $\overline{\mathbf{T}}_K$ is a minus sign in the expression for the diagonal elements. Thus this method of calculating an approximation to $\mathbf{C}_N^{-1}\mathbf{u}$ has the same computational cost per iteration as the tridiagonal method based on $\mathbf{C}_N$ ($\mathcal{O}(N^2)$).

3.8 The Trace

As well as calculating the inverse of a matrix applied to a vector, we need to calculate the trace of the inverse of a matrix under the same restrictions on computational cost. Let us construct

$$\tau = \mathbf{r}^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta} \mathbf{r} \tag{3.64}$$

where $\mathbf{r}$ is a random vector with its elements r_n having a Gaussian distribution with zero mean and unit variance. Taking the expectation with respect to the distribution over $\mathbf{r}$

$$E[\tau] = \text{tr}\left[\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta}\right] \tag{3.65}$$

and

$$\sigma_\tau^2 = 2 \text{ tr}\left[\left(\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta}\right)^2\right] \tag{3.66}$$

Hence we may approximate the trace by averaging several different τ's, i.e., over several $\mathbf{r}$'s. In fact, for large matrices, the number of τ's needed to obtain a good estimate of the trace is surprisingly small. Also as we increase the size of the matrix, the number of random samples we require to achieve a given level of accuracy on our estimation of the trace decreases making the method extremely efficient for large matrices.

Calculating the individual τ's used to find the trace requires us to evaluate $\mathbf{r}^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta} \mathbf{r}$. This can be done using the methods described in Section 3.3 to calculate $\mathbf{y}_K \simeq \mathbf{C}_N^{-1} \mathbf{r}$ and hence give $\tau \simeq \mathbf{y}_K^T \frac{\partial \mathbf{C}_N}{\partial \theta} \mathbf{r}$.

3.9 Counting the Cost

Having now described different methods for implementing a Gaussian process let us look more closely at their computational cost. Table 3.1 gives the cost associated with the implementation of a Gaussian process using three different methods - the direct inversion method, the indirect LU decomposition method and the tridiagonal approximate inversion method. Only leading order terms are included. The number of gradient vectors $\mathbf{g}$ that need to be evaluated by the approximate inversion method is also given for each task. Table 3.1 first gives the initialization cost which is primarily the cost of inverting $\mathbf{C}_N$ (whether explicitly or otherwise). This initialization is necessary before we are able to make any predictions or to evaluate any gradients. The extra cost of making a prediction at a new point $\mathbf{x}_{N+1}$ and calculating the error bars is then given. The table also gives the extra cost of calculating the gradient of the log likelihood $\mathcal{L}$ given that we have already paid the initialization cost.

In order to produce meaningful statistics for these tasks, we have made several assumptions. Firstly the computational cost of evaluating $\mathbf{C}_N$ and any differentials of $\mathbf{C}_N$ has been assumed to be $\mathcal{O}(LN^2)$ where L is the dimensionality of the input space. This may not be the case for very complicated noise models. Secondly we have assumed that only $\mathcal{O}(L)$ derivatives need to be calculated, i.e., the number of hyperparameters is not significantly more than L. Thirdly we have assumed for the approximate inversion method that the number of gradient vectors required to achieve a specified $\Delta Q_K / Q(\mathbf{y}_K)$ is approximately constant. Finally, any cost associated with the priors on hyperparameters has been ignored.

There are two important points to note from Table 3.1. Firstly, the scaling of the initialization cost with the number of data points N for the direct methods is $\mathcal{O}(N^3)$ but for the approximate methods it is $\mathcal{O}(KN^2)$. Thus as the amount of data increases, the approximate methods are

	Direct	Indirect	Approximate	
			operations	**g** calcs
Initialization	$2N^3 + LN^2$	$2N^3/3 + LN^2$	$KN^2 + LN^2$	K
Prediction :				
mean	$N + LN$	$N^2 + LN$	$N + LN$	0
error bars	N^2	$2N^2$	KN^2	K
Gradient	$4N^2 + L(L+2)N^2$	$4N^2 + L(L+2)N^2$	$\rho KN^2 + L(L+2)N^2$	ρK

Table 3.1: Computational Cost of Methods. This tables gives, to leading order, the number of floating point operations required to perform four tasks for three different methods: the initialization cost and the extra cost to make a prediction, calculate error bars and evaluate the gradient of the log likelihood. The table also gives the number of **g** vectors needed for each task using the tridiagonal approximate inversion method. N is the dimension of the matrix $\mathbf{C}_N$, K is the average number of **g** vectors generated in order to estimate $\mathbf{C}_N^{-1}\mathbf{u}$ using the approximate approach. ρ is the number of random samples τ used to estimate the trace.

significantly faster than the direct ones as, for moderately well conditioned matrices, K can be significantly smaller than N and ρ can be as low as 2. Secondly for both the direct and the approximate inversion methods, once we have paid the initial cost to invert $\mathbf{C}_N$ we can make predictions at new points very efficiently. The extra accuracy of the indirect method requires slightly more computation for this task.

3.10 Examples

We shall now consider a brief set of toy examples to demonstrate the Gaussian process regression model and highlight the differences between the exact and approximate implementations. We shall then look at two real world examples - modelling the spike signal of a zebra finch neuron and modelling potential energy surfaces of weakly bound molecules.

3.10.1 1D Toy Example

In order to appreciate some of the basic properties of Gaussian process regression, let us consider the following simple example. We have a set of 37 noisy training data points which are shown in Figure 3.5(a). We shall use the basic covariance function given in equation 2.20 and a input independent noise model to provide a regression model.

10 different runs with different initial values of the hyperparameters were performed to guard against the possibility of multiple minima in $P(\Theta|\mathcal{D})$. For each run, the hyperparameters were randomly sampled from their priors and then a conjugate gradient optimization routine was used to find the most probable Θ given the data. The optimization of the hyperparameters was terminated when the magnitude of the gradient of the log posterior with respect to the hyperparameters fell below a threshold value. The matrix inversions involved in the optimization were performed using exact LU-decomposition methods. Broad priors were placed on all the hyperparameters

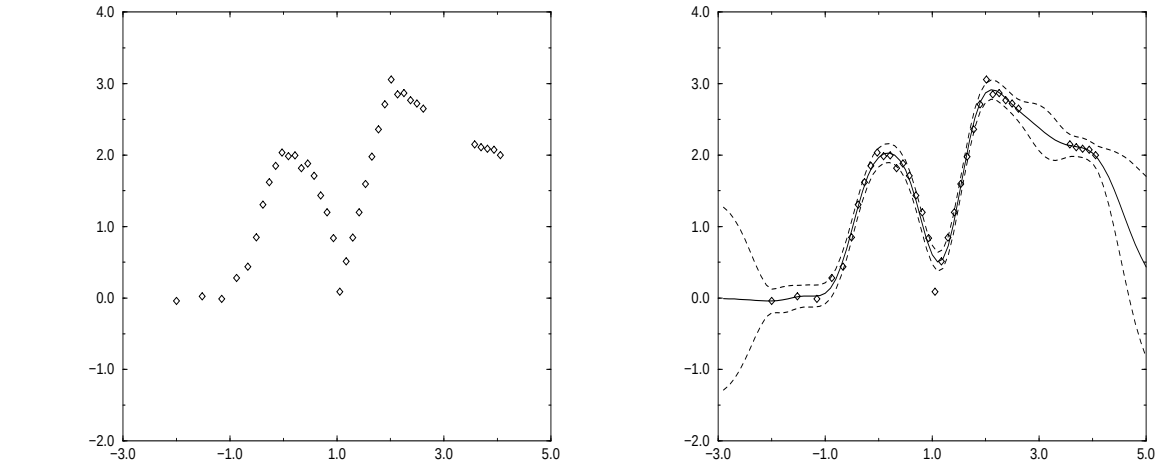


Figure 3.5: 1d Toy Example : (a) shows the 37 noisy data points used as the training data. Note the lack of data in the region $2.5 \leq x \leq 3.5$. In (b) we see the interpolant generated by a Gaussian process and its 1σ error bars. The error bars represent how uncertain we are about the interpolant at each point assuming that the model is correct. They take into account both the noise level and the uncertainty due to the finite amount of data. Notice how the error bars increase as the data point density decreases. Also note that the point near $(1, 0)$ is outside the error bars because the priors specify that the interpolant is moderately smooth. Hence this point is treated as an improbable outlier.

$(Ga(r_l|2.0, 2.0)$, $Ig(\theta_1|4.0, 3.0)$, $Ig(\theta_2|4.0, 3.0)$, $Ig(\theta_3|1.0, 3.0))$ as it was assumed that there was little prior knowledge other than a belief that the interpolant should be relatively smooth.

The results obtained in each of the 10 runs were almost identical and the results from the first of the runs can be seen in be seen in Figure 3.5(b). One can compare these results with those of MacKay (1992a) using a radial basis function model which is equivalent to a Gaussian process (as shown in section 2.5). It should be noted that a finite radial basis function model is not equivalent to a Gaussian process defined by a covariance function of the form of equation 2.20. This explains the slight discrepancies between the results in Figure 3.5(b) and those in MacKay (1992a).

3.10.2 2D Toy Example using Approximate Inversion Techniques

For this example 400 noisy data points were generated by taking random samples from a 2D function (see Figure 3.6(a)) and adding Gaussian noise ($\sigma = 0.03$). Broad priors were placed on all the hyperparameters of the Gaussian process used to model the data $(Ga(r_l|0.5, 2.0), Ig(\theta_1|1.0, 3.0)$, $Ig(\theta_2|1.0, 3.0), Ig(\theta_3|1.0, 3.0))$ and ten optimizations of the hyperparameters were performed, each with initial hyperparameters sampled from their priors. In this case the approximate techniques of Section 3.2 were used to evaluate inverses and traces. Two random vectors were used to evaluate traces ($\rho = 2$) and an accuracy of $\Delta Q_K / Q_K = 0.01$ was required for the approximate inversions. The results from each run were very similar and the interpolant obtained from the first run is shown in Figure 3.6(b).

Let us consider the time required to perform a single optimization of the hyperparameters. Each training run took about 5 minutes using the approximate inversion techniques. Training runs performed using direct methods took approximately 39 minutes. This shows the advantage of the approximate methods when dealing with large amounts of training data. Figure 3.7 shows

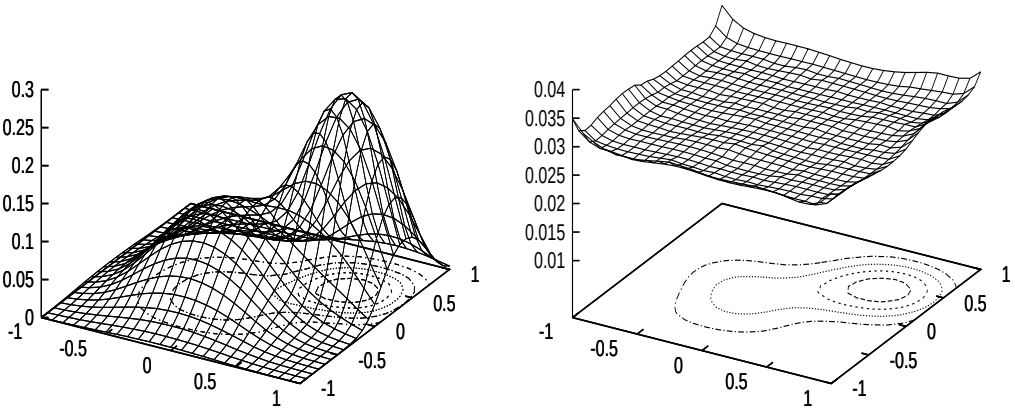


Figure 3.6: 2d Toy Example : (a) shows the function from which 400 noisy data points were generated. The contours of function are shown on the base of the plot. In (b) we see the contours of the most probable interpolant generated using an approximate implementation of a Gaussian process. The magnitude of the 1σ error bar is shown by the surface.

the training times for data sets of varying size generated using the same function. As for the 1D example, the optimization of the hyperparameters was terminated when the magnitude of the gradient of the log posterior with respect to the hyperparameters fell below a fixed threshold value. The threshold for this particular example was set to quite a high value ($|\mathbf{g}_k|^2 \leq 0.1$) to avoid the training times for the direct method being several weeks.

3.10.3 Zebra Finch Neuron Spike

Traditional parameterized interpolation models allow us to express our beliefs about the smoothness of an interpolant through the choice of regularizer $\mathcal{R}$ and the choice of noise model $\mathcal{N}$. Often the regularizer and noise model have quadratic forms and associated hyperparameters α and β respectively. For example, consider the Bayesian treatment of a neural network (MacKay 1992d). Here we express a Gaussian prior on the weights $\mathbf{w}$ of the network

$$\log P(\mathbf{w}|\alpha) = -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + \text{const} \tag{3.67}$$

and define the noise model in terms of a sum squared error function between the output of the network $y(\mathbf{x}_n;\mathbf{w})$ and the target output t_n for a given input $\mathbf{x}_n$.

$$\log P(\mathbf{t}|\mathbf{w},\{\mathbf{x}_n\}) = -\frac{\beta}{2}\sum_n (y(\mathbf{x}_n;\mathbf{w}) - t_n)^2 + \text{const.} \tag{3.68}$$

The ratio of the two hyperparameters α/β determines the smoothness of the interpolant: large values of α/β result in smooth interpolants and, conversely, small values of α/β lead to jagged interpolants.

Such a regularization system is fine when the smoothness of the function does not vary significantly over the region covered by the data. However if the data contains a sharp peak but is

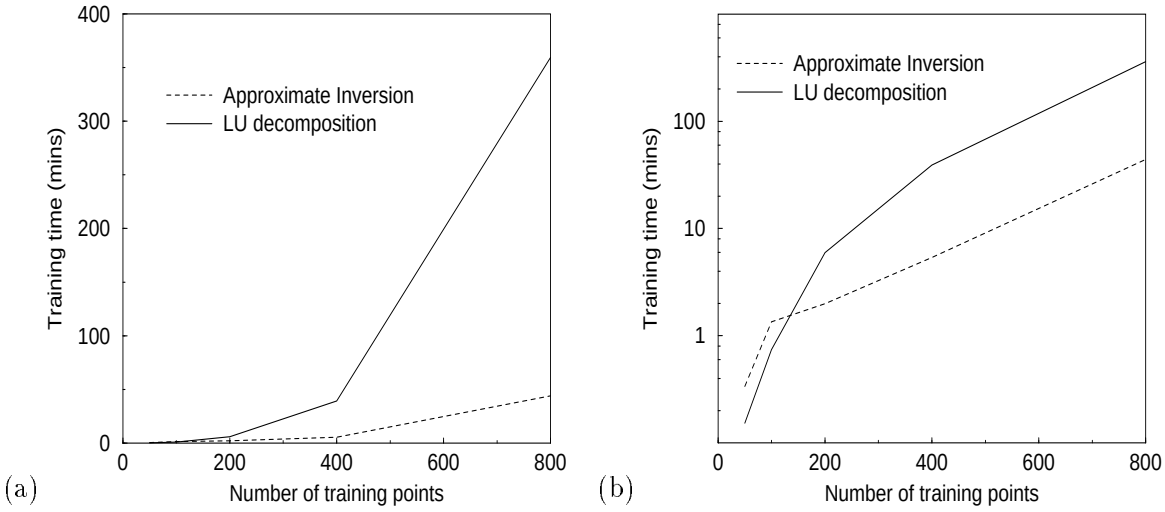


Figure 3.7: Scaling of training time with amount of data : (a) shows the training times for noisy data sets of different sizes generated from the function shown in Figure 3.6. Times are shown for both the direct LU decomposition method and the approximate method. (b) shows the same graph with a logarithmic vertical scale. For small data sets the LU method is significantly quicker than the approximate approach. At over 100 data points we can see that the $\mathcal{O}(N^2)$ scaling of the approximate method begins to yield benefits. For 400 and 800 data points the approximate method is significantly faster than the LU decomposition method. It should, however, be noted that the problem is only two dimensional and the benefits of the approximate method for higher dimensional problems will only be obvious for larger data sets.

otherwise very smooth then this can cause problem: setting α/β to a large value to describe the majority of the data will cause the model to smooth out the peak; setting α/β to a small value will cause the peak to be modelled well but may cause over-fitting away from the peak, especially if the data is noisy. It would therefore be useful to have a model capable of capturing spatially varying smoothness.

The Gaussian process defined by the covariance function in equation 2.20 suffers from the same problem as a standard neural network. The quantity $r_l^2\theta_3$ behaves in the same manner as α/β, providing no way to deal with spatial variation in interpolant smoothness. The stationary nature of the covariance function makes it difficult for the Gaussian process to model what is a manifestly non-stationary phenomenon effectively.

Let us consider an example to highlight the failings of the "simple" stationary Gaussian process. Figure 3.8 shows the action potential of a neuron deduced from recordings of 40 distinct events (Lewicki 1994). The function has one spike with large characteristic amplitude but elsewhere the true function is smooth. Following the treatment of MacKay and Takeuchi (1994), we shall add Gaussian noise of standard deviation 1.0 to the data depicted in Figure 3.8(a) in order to create 120 noisy training data points[1]. The predictive mean of stationary Gaussian processes trained on this data can be seen in Figure 3.8(b) and (c). The Gaussian process with a prior favouring large length scales smooths out the peak and a Gaussian process with a prior favouring small length scales fits the peak well, as expected, but produces over-fitting away from the peak.

[1]In order to embody the fact that the signal is smooth and has a value of zero before the spike, 20 data points with associated output 0.0 were added to the data set used by MacKay and Takeuchi (1994) in the input region -0.2 to 0.0.

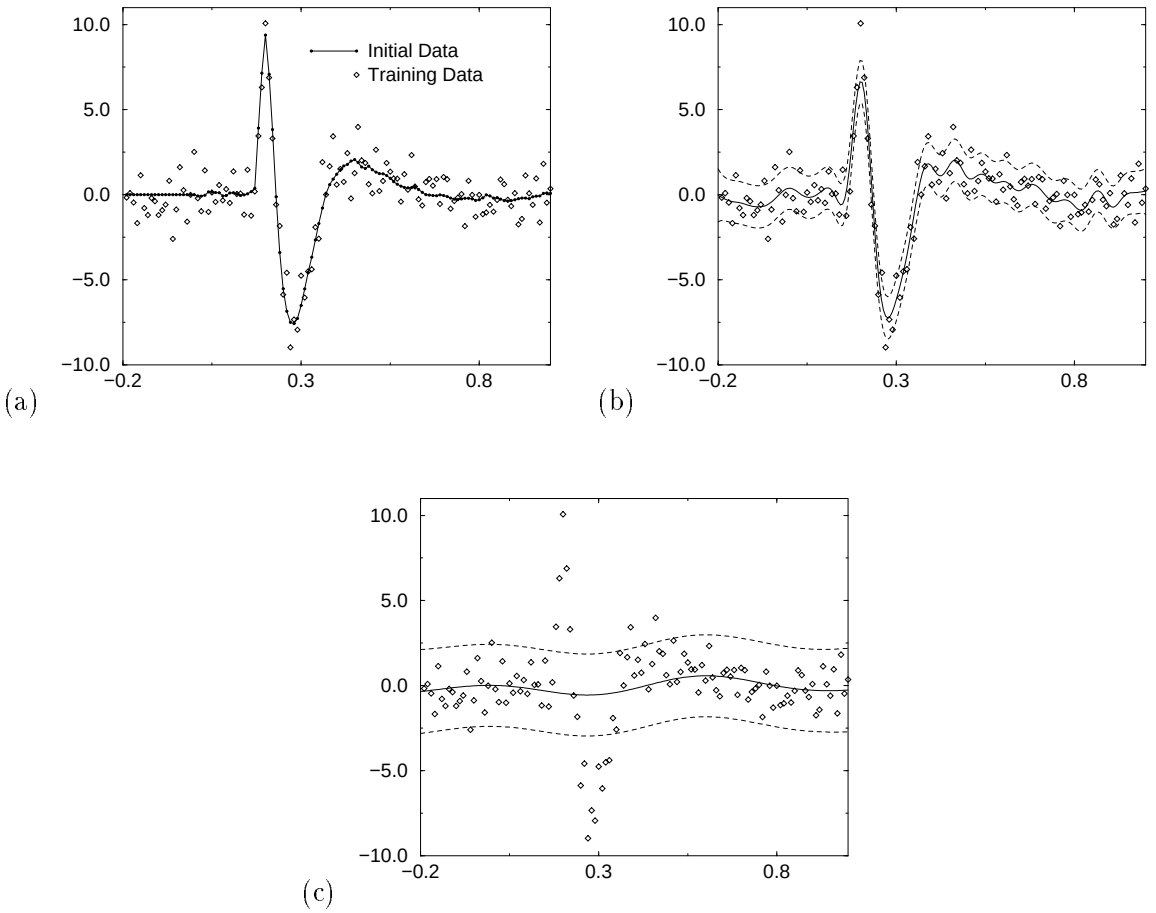


Figure 3.8: Zebra Finch Neuron Spike Interpolation : (a) shows the inferred signal spike from a zebra finch neuron. Gaussian noise with unit variance was added to this data set to create the training data, shown by the diamond symbols. The predictive mean of a stationary Gaussian process with a prior favouring a small length scale ($Ga(r_l|0.1, 0.6)$) and favouring a large length scale ($Ga(r_l|0.5, 0.6)$) is shown in (b) and (c) respectively.

Non-stationary Covariance function Approach

Let us consider the possibility of a non-stationary covariance function that can embody spatially varying length scales. Simply letting the length scale in equation 2.20 be dependent on $\mathbf{x}$ is not sufficient as this does not produce a positive definite function. As mentioned in Section 2.5, a possible way to derive a covariance function is to consider the analogous fixed basis function network. We wish to derive a covariance function that allows us to have spatially varying length scales for all the inputs of our Gaussian process. Hence let us consider the set of basis functions $\{\phi_k(\mathbf{x})\}$ where

$$\phi_k(\mathbf{x}) = \prod_{l=1}^{L} \sqrt{\frac{\sqrt{2}}{r_l(\mathbf{x};\Theta_l)}} \exp\left[-\sum_{l=1}^{L} \frac{(x^{(l)} - a_k^{(l)})^2}{r_l^2(\mathbf{x};\Theta_l)}\right] \tag{3.69}$$

and $r_l(\mathbf{x};\Theta_l)$ is a parameterized positive function of $\mathbf{x}$. The hyperparameters $\Theta = \{\Theta_l\}_{l=1}^{L}$ are common to all the basis functions and each basis function has a distinct centre $\mathbf{a}_k$.

There are two things to note about these basis functions. Firstly the length scales here are functions of $\mathbf{x}$ and not $\mathbf{x}_k$. Using length scales that are functions of $\mathbf{x}_k$ would be preferable but would make the integrals required to produce a covariance function intractable. Secondly, the prefactor of the exponential term might appear rather arbitrary. It is introduced to ensure that the resulting covariance function has a variance which is constant with respect to $\mathbf{x}$ and equal to θ_1.

We shall now rewrite the above equation in terms of a diagonal matrix $\mathbf{M}(\mathbf{x};\Theta)$ where $M_{ll}(\mathbf{x};\Theta) = 1/r_l^2(\mathbf{x};\Theta_l)$:

$$\phi_k(\mathbf{x}) = \prod_{l=1}^{L} \sqrt{\frac{\sqrt{2}}{r_l(\mathbf{x};\Theta_l)}} \exp\left[-(\mathbf{x} - \mathbf{a}_k)^T \mathbf{M}(\mathbf{x} - \mathbf{a}_k)\right] \tag{3.70}$$

For a network of K such fixed basis functions, the covariance function can be written (see Section 2.5)

$$\begin{aligned} \mathcal{G}_f(\mathbf{x}_m, \mathbf{x}_n) &= \prod_{l=1}^{L} \sqrt{\frac{2}{r_l(\mathbf{x}_m;\Theta_l) r_l(\mathbf{x}_n;\Theta_l)}} \sum_{k=1}^{K} \exp\Big(-(\mathbf{x}_m - \mathbf{a}_k)^T \mathbf{M}_m (\mathbf{x}_m - \mathbf{a}_k) \\ &\quad -(\mathbf{x}_n - \mathbf{a}_k)^T \mathbf{M}_n (\mathbf{x}_n - \mathbf{a}_k)\Big) \end{aligned} \tag{3.71}$$

with $\mathbf{M}_n = \mathbf{M}(\mathbf{x}_n;\Theta)$. Taking the limit as $K \to \infty$, the covariance function becomes

$$\begin{aligned} \mathcal{G}_f(\mathbf{x}_m, \mathbf{x}_n) &= \prod_{l=1}^{L} \sqrt{\frac{2}{r_l(\mathbf{x}_m;\Theta_l) r_l(\mathbf{x}_n;\Theta_l)}} \exp\left[-\mathbf{x}_m \mathbf{M}_m \mathbf{x}_m - \mathbf{x}_n \mathbf{M}_n \mathbf{x}_n\right] \\ &\quad \times \int \exp\left[-\mathbf{a}_k^T \left(\mathbf{M}_m + \mathbf{M}_n\right) \mathbf{a}_k + 2\mathbf{a}_k^T \left(\mathbf{M}_m \mathbf{x}_m + \mathbf{M}_n \mathbf{a}_n\right)\right] d\mathbf{a}_k \end{aligned} \tag{3.72}$$

Now using the identity

$$\int \exp\left(-\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{h}^T \mathbf{x}\right) d\mathbf{x} = (2\pi)^{L/2} |\mathbf{A}|^{-1/2} \exp\left(\frac{1}{2}\mathbf{h}^T \mathbf{A}^{-1} \mathbf{h}\right) \tag{3.73}$$

and writing

$$\mathbf{A} = 2\mathbf{M}_m + 2\mathbf{M}_n \tag{3.74}$$

$$\mathbf{h} = 2\mathbf{M}_m \mathbf{x}_m + 2\mathbf{M}_n \mathbf{x}_n \tag{3.75}$$

we find

$$\begin{aligned}\mathcal{G}_f(\mathbf{x}_m,\mathbf{x}_n) &= \mathcal{G}_0\frac{2^{L/2}|\mathbf{M}_m+\mathbf{M}_n|^{-1/2}}{\prod_{l=1}^{L}\sqrt{r_l(\mathbf{x}_m;\Theta_l)r_l(\mathbf{x}_n;\Theta_l)}}\exp\left[-\mathbf{x}_m\left(\mathbf{M}_m-\mathbf{M}_m(\mathbf{M}_m+\mathbf{M}_n)^{-1}\mathbf{M}_m\right)\mathbf{x}_m\right.\\ &\left.-\mathbf{x}_n\left(\mathbf{M}_n-\mathbf{M}_n(\mathbf{M}_m+\mathbf{M}_n)^{-1}\mathbf{M}_n\right)\mathbf{x}_n+2\mathbf{x}_m\left(\mathbf{M}_m(\mathbf{M}_m+\mathbf{M}_n)^{-1}\mathbf{M}_n\right)\mathbf{x}_n\right] \qquad (3.76)\end{aligned}$$

where $\mathcal{G}_0$ is a constant independent of $\mathbf{x}$ and $\{r_l(\mathbf{x};\Theta_l)\}$. Because $\mathbf{M}_m$ and $\mathbf{M}_n$ commute,

$$\begin{aligned}\mathbf{M}_m\mathbf{M}_n &= (\mathbf{M}_m+\mathbf{M}_n)\left[\mathbf{M}_n-\mathbf{M}_n(\mathbf{M}_m+\mathbf{M}_n)^{-1}\mathbf{M}_n\right] \qquad (3.77)\\ &= (\mathbf{M}_m+\mathbf{M}_n)\left[\mathbf{M}_m-\mathbf{M}_m(\mathbf{M}_m+\mathbf{M}_n)^{-1}\mathbf{M}_m\right] \qquad (3.78)\end{aligned}$$

Hence

$$\mathcal{G}_f(\mathbf{x}_m,\mathbf{x}_n)=\mathcal{G}_0\frac{2^{L/2}|\mathbf{M}_m+\mathbf{M}_n|^{-1/2}}{\prod_{l=1}^{L}\sqrt{r_l(\mathbf{x}_m;\Theta_l)r_l(\mathbf{x}_n;\Theta_l)}}\exp\left[-(\mathbf{x}_m-\mathbf{x}_n)^T\underbrace{\mathbf{M}_m\mathbf{M}_n(\mathbf{M}_m+\mathbf{M}_n)^{-1}}_{\mathbf{K}^{mn}}(\mathbf{x}_m-\mathbf{x}_n)\right] \qquad (3.79)$$

By substituting back in $\mathbf{M}_m=\mathrm{diag}(1/r_1^2(\mathbf{x}_m;\Theta_1),\cdots,1/r_L^2(\mathbf{x}_m;\Theta_L))$ we can write

$$\begin{aligned}|\mathbf{M}_m+\mathbf{M}_n|^{-1/2} &= \prod_{l=1}^{L}\left\{\frac{r_l^2(\mathbf{x}_m;\Theta_l)r_l^2(\mathbf{x}_n;\Theta_l)}{r_l^2(\mathbf{x}_m;\Theta_l)+r_l^2(\mathbf{x}_n;\Theta_l)}\right\}^{1/2} \qquad (3.80)\\ K_{lm}^{mn} &= \frac{\delta_{lm}}{r_l^2(\mathbf{x}_m;\Theta_l)+r_m^2(\mathbf{x}_n;\Theta_m)} \qquad (3.81)\end{aligned}$$

Hence we obtain the covariance function

$$\mathcal{C}_f^{ns}(\mathbf{x}_m,\mathbf{x}_n;\Theta)=\theta_1\prod_l\left\{\frac{2r_l(\mathbf{x}_m;\Theta)r_l(\mathbf{x}_n;\Theta)}{r_l^2(\mathbf{x}_m;\Theta)+r_l^2(\mathbf{x}_n;\Theta)}\right\}^{1/2}\exp\left(-\sum_l\frac{\left(x_m^{(l)}-x_n^{(l)}\right)^2}{r_l^2(\mathbf{x}_m;\Theta)+r_l^2(\mathbf{x}_n;\Theta)}\right). \qquad (3.82)$$

Assuming the length scales are parameterized using some set of fixed basis functions,

$$r_l(\mathbf{x};\Theta_l)=\exp\left(\sum_j\beta_j^{(l)}\psi_j(\mathbf{x})\right), \qquad (3.83)$$

then the derivative of the covariance function with respect to one of the hyperparameters $\beta_b^{(a)}$ is given by

$$\begin{aligned}\frac{\partial\mathcal{C}_f^{ns}(\mathbf{x}_m,\mathbf{x}_n)}{\partial\beta_b^{(a)}} &= \mathcal{C}_f^{ns}(\mathbf{x}_m,\mathbf{x}_n)\left[\frac{r_a^2(\mathbf{x}_m;\Theta_a)-r_a^2(\mathbf{x}_n;\Theta_a)}{r_a^2(\mathbf{x}_m;\Theta_a)+r_a^2(\mathbf{x}_n;\Theta_a)}\left(\psi_b(\mathbf{x}_n)-\psi_b(\mathbf{x}_m)\right)\right.\\ &\left.+\frac{2(x_m^{(a)}-x_n^{(a)})^2}{\left(r_a^2(\mathbf{x}_m;\Theta_a)+r_a^2(\mathbf{x}_n;\Theta_a)\right)^2}\left(r_a^2(\mathbf{x}_m;\Theta_a)\psi_b(\mathbf{x}_m)+r_a^2(\mathbf{x}_n;\Theta_a)\psi_b(\mathbf{x}_n)\right)\right] \qquad (3.84)\end{aligned}$$

While such a covariance function is positive definite, we must check that varying the length scales produces the required results. The crucial part of the covariance function is the prefactor to the exponential

$$f(\mathbf{x}_m, \mathbf{x}_n; \Theta) = \prod_l \left\{ \frac{2 r_l(\mathbf{x}_m; \Theta_l) r_l(\mathbf{x}_n; \Theta_l)}{r_l^2(\mathbf{x}_m; \Theta_l) + r_l^2(\mathbf{x}_n; \Theta_l)} \right\}^{1/2}. \tag{3.85}$$

Ideally we would like this to be constant for all values of $r_l(\mathbf{x})$. However, as previously mentioned, this would not produced a positive definite function. $f(\mathbf{x}_m, \mathbf{x}_n; \Theta)$ is constructed to have constant variance but what about its behaviour at $\mathbf{x}_m \neq \mathbf{x}_n$? Let us consider two examples. Firstly a case where the length scale changes from a large value to a small value and then back again and, secondly, the converse where it changes from a small scale to a large one and back again. Plots of the variation of the covariance and $f(\mathbf{x}_m, \mathbf{x}_n; \Theta)$ are shown in figure 3.9 and 3.10. For the first case, the covariance behaves as we would like being sharply peaked in the region of small length scale and broad in the region of large length scale. Another interesting feature is that the large length scale sections on either side of the small length scale region are quite strongly correlated. For the second case, we obtain some less pleasing behaviour. The covariance initially drops off faster in the area of large length scale than in the areas of a small length scale. The long range decrease of the covariance in the area of a large length scale is slower. Samples drawn from a Gaussian process using the covariance function in equation 3.82 with length scales as shown in figure 3.10(a) appear to be more jagged in the region where the length scale increases (see figure 3.10(d)).

The cause of the problems with the covariance function in equation 3.82 are rapid increases in the length scales. Rapid in this context means the variation of r_l itself has a length scale that is significantly smaller than r_l. The prefactor $f(2.0, x; \Theta)$ varies at a similar rate to the length scale and so, for a length scale that is varying rapidly, the prefactor will introduce undesirable behaviour for increasing length scales. The covariance function is useful for modelling neuron spike data (which we will discuss in section 3.10.3) as the length scale for a spike feature behaves in a similar manner to the length scale shown in Figure 3.9. However because of the influence of the prefactor, the general interpretation of $r_l(\mathbf{x}; \Theta)$ is less clear than for the stationary model.

Non-linear map Approach

Instead of having the length scales vary across the input space, another approach to modelling non-stationary behaviour is to transform the data into an input space in which it is stationary and then use a stationary Gaussian process. The map from the original input vectors $\mathbf{x}_n$ to the new input vectors $\tilde{\mathbf{x}}_n$ has to have several properties. Firstly it needs to be a one-to-one map. Secondly, the map needs to be continuous. Thus the order of any number of points on any non-intersecting line in $\mathbf{x}$-space is preserved by the map and the line is distorted by the map but never crosses itself.

We define the map $\mathbf{x} \to \tilde{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_L(\mathbf{x}))$. In order to ensure that the map is one-to-one, we define it in terms integrals over a set of positive densities. We define the densities:

$$g_l(\mathbf{x}) = \sum_{j=1}^{J} e^{\beta_j^{(l)}} \psi_j(\mathbf{x}), \tag{3.86}$$

where $\beta_j^{(l)} \in \Theta$ and the $\{\psi_j(\mathbf{x})\}$ are a set of basis functions, common to all the densities, which are positive for all $\mathbf{x}$. Let

$$f_l(\mathbf{x}_n) = x_0^{(l)} + \int_{S_{0n}} g_l(\mathbf{x}) \, \mathrm{d}s \tag{3.87}$$

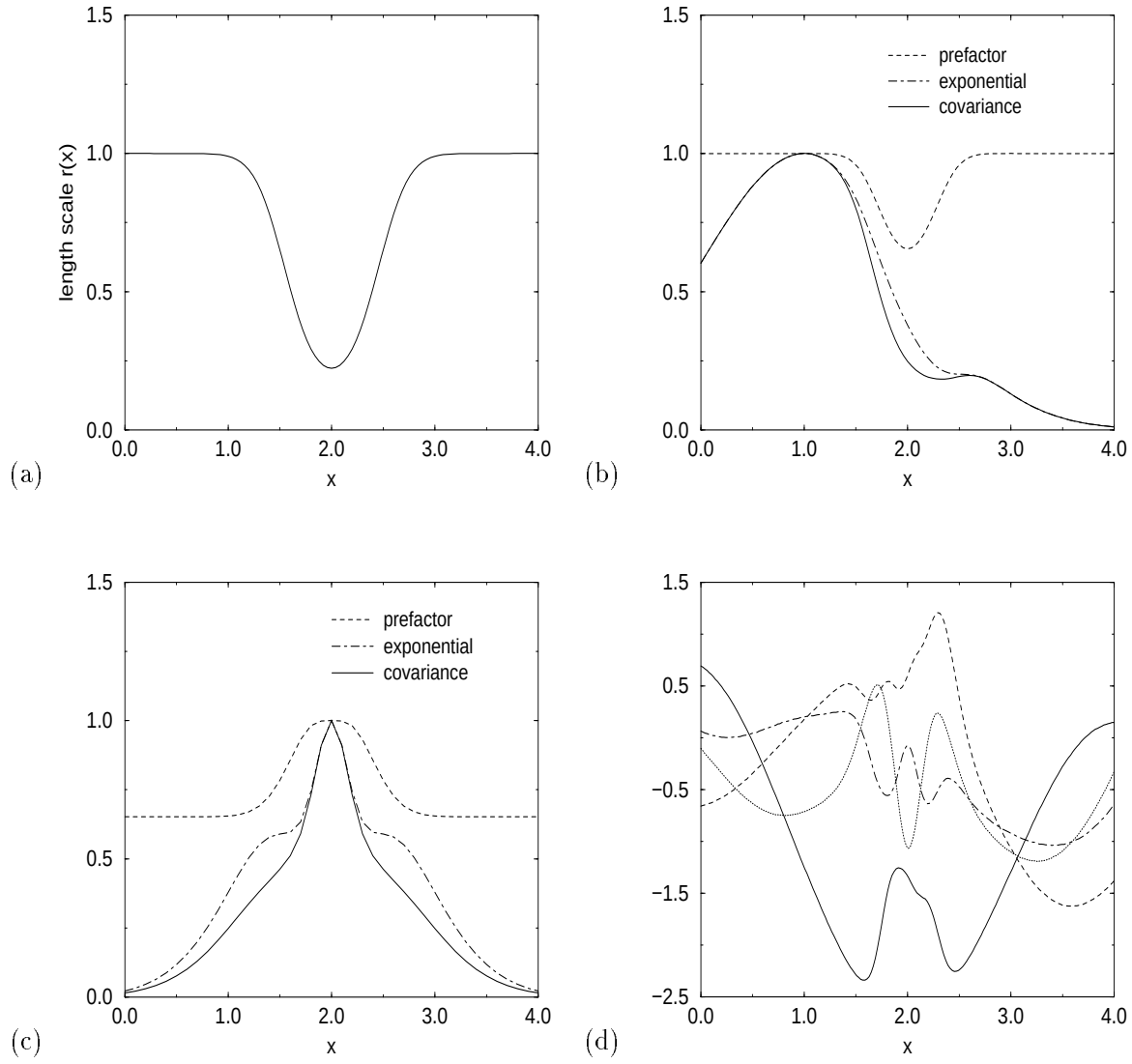


Figure 3.9: Behaviour of the Covariance Function : (a) shows an example variation in length scale across a one dimensional region. (b) shows the variation of the covariance function $\mathcal{C}_f^{ns}(1.0, x; \Theta)$ as defined in equation 3.82 and the two factors (the prefactor and the exponential part) which make up this covariance function. One argument of the covariance function is fixed at 1.0 and the other, x, is allowed to vary. (c) is analogous to (b) but for a fixed argument of 2.0. (d) shows four samples from the a Gaussian process with a covariance function as defined in equation 3.82. Note how the covariance in (c) falls away much more sharply than that in (b) and hence the random samples in (d) are more jagged in the region with smaller length scales, as we would want.

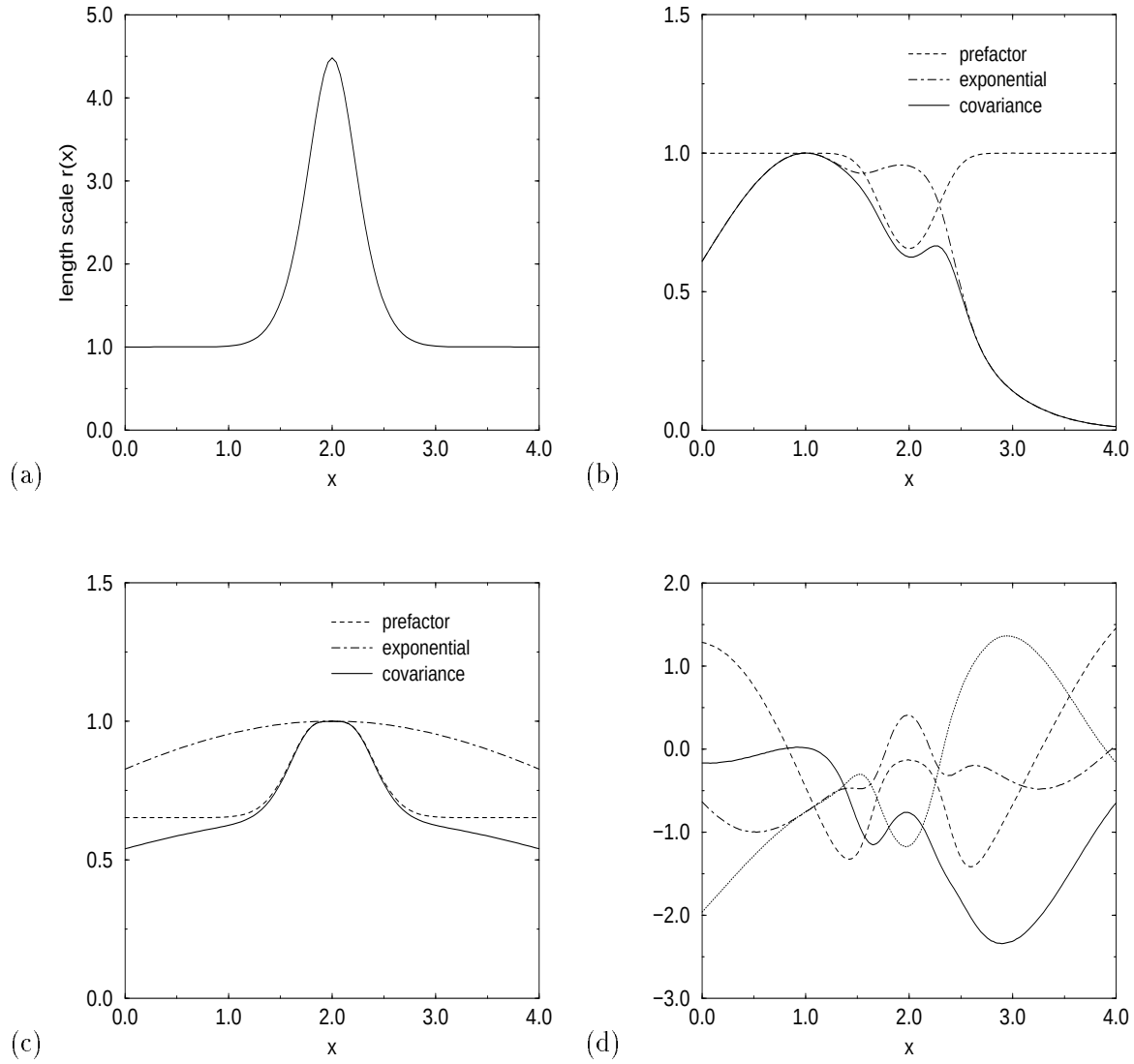


Figure 3.10: Behaviour of the Covariance Function : (a) shows an example variation in length scale across a one dimensional region. (b) shows the variation of the covariance function $\mathcal{C}_f^{ns}(1.0, x; \Theta)$ as defined in equation 3.82 and the two factors (the prefactor and the exponential part) which make up this covariance function. One argument of the covariance function is fixed at 1.0 and the other, x, is allowed to vary. (c) is analogous to (b) but for a fixed argument of 2.0. (d) shows four samples from the covariance function. Note how the covariance in (c) falls away much more sharply than that in (b) and hence the random samples in (d) show a greater variation in the region with higher length scales. This is due to the unfortunate behaviour of the prefactor $f(2.0, x; \Theta)$ in that region.

where $\mathbf{x}_0 = (x_0^{(1)}, \cdots, x_0^{(L)})$ is a reference vector (preferably chosen to be somewhere in the centre of the data) and S_{0n} is a path between $\mathbf{x}_0$ and $\mathbf{x}_n$. As long as the integral of $g_l(\mathbf{x})$ exists for any $\mathbf{x}_0$ and $\mathbf{x}_n$, the integral in equation 3.87 is independent of the path S_{0n}. We therefore wish to choose the $\{\psi_j(\mathbf{x})\}$ such that their integral between any two points exists and can be evaluated simply, with minimal computational cost.

Let us consider using Gaussian basis functions,

$$\psi_j(\mathbf{x}) = \exp\left(-\frac{1}{2}\sum_{l=1}^{L}\frac{(x^{(l)} - a_j^{(l)})^2}{\sigma_l^2}\right) \tag{3.88}$$

where $\mathbf{a}_j$ is the centre of each of the functions and σ_l is an appropriate width. We can evaluate the integral of an uncorrelated Gaussian between any two points by performing a series of integrals along the axial directions. Each integral is an error function which can be approximated efficiently and accurately using Chebyshev polynomials (see Press *et al.* (1988) page 176).

Using the map, we re-define our covariance function:

$$\mathcal{C}_f^{map}(\mathbf{x}_m, \mathbf{x}_n; \Theta) = \theta_1 \exp\left(-\frac{1}{2}\sum_l (f_l(\mathbf{x}_m) - f_l(\mathbf{x}_n))^2\right). \tag{3.89}$$

The length scales $r_l(\mathbf{x}) \equiv 1/g_l(\mathbf{x})$.

Applying Gaussian processes to the Data

We begin by attempting to model the neuron spike data using the non-stationary covariance function in equation 3.82. The log of the length scales was defined in terms of a set of twelve basis functions, one constant term and eleven Gaussians which are shown in Figure 3.8(a), each with an associated hyperparameter $\beta_j^{(1)} \in \Theta_l$:

$$r_l(\mathbf{x}; \Theta_l) = \exp\left(\beta_0^{(1)} + \sum_{j=1}^{11}\beta_j^{(1)}\psi_j(\mathbf{x})\right). \tag{3.90}$$

There was a prior belief that the length scale varied moderately smoothly with position in input space and so a regularizing Gaussian prior was placed on each of the $\beta_j^{(1)}$ $(1 < j < 11)$ with mean of zero and variance of 5.0. A Gaussian prior $(\mu = \log(0.2), \sigma^2 = 50)$ was placed on $\beta_0^{(1)}$. We also had the prior knowledge that the interpolant should tend to zero at the extremes of the data. However this information was difficult to embody in terms of a prior on the $\{\beta_j^{(1)}\}$ and was ignored in the training.

The predictive mean obtained using the non-stationary covariance function can be seen in Figure 3.8(b). The interpolant fits the peak well but does not exhibit the over-fitting found using the stationary covariance function. Figure 3.8(c) shows the variation of the length scale with $\mathbf{x}$. As expected the length scale is small in the region of the spike and becomes larger on either side.

The data was then modelled using the non-linear map approach. The density was defined in terms of the eleven Gaussian basis functions shown in Figure 3.11(a). A broad Gaussian prior (mean 0.0, standard deviation 10.0) was placed on the density parameters $\beta_j^{(1)}$ as defined in equation 3.86. The prior belief that the interpolant is zero at the edges of the data was again ignored. The predictive mean and the variation of length scale are shown in Figure 3.11(e) and (f). The peak is modelled well without over-fitting and the length scale is small near the peak. The error bars on the length scale bulge out in the region of the peak. This is due to the parameterization of the density and the inverse relationship between the density and the length scale. Small uncertainties in $\beta_j^{(1)}$ for a Gaussian centred on the peak will cause large uncertainties in the length scale.

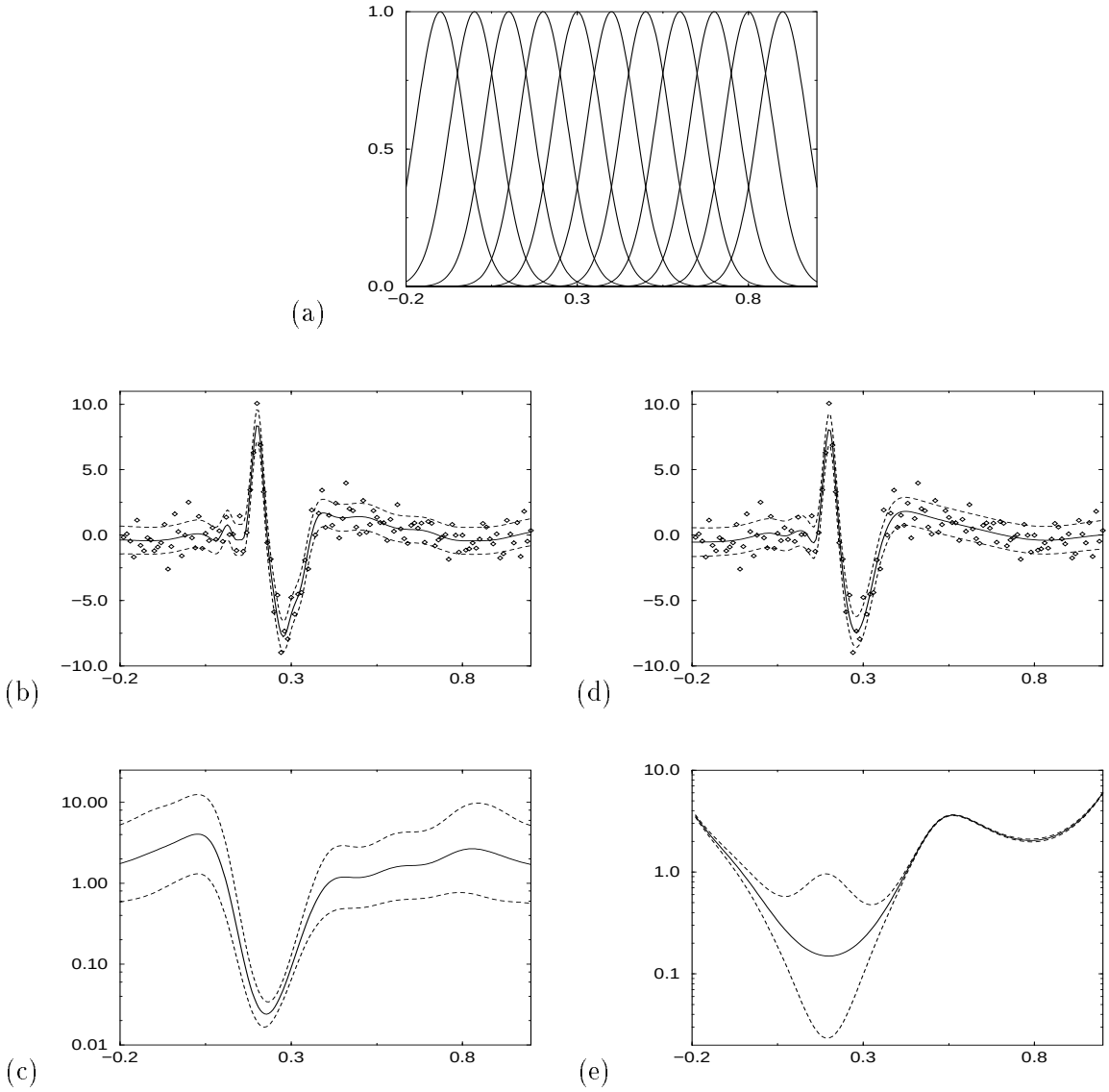


Figure 3.11: Zebra Finch Neuron Spike Results : The noisy data in figure 3.8(a) was modelled using the covariance function in equation 3.82. (a) shows the set of Gaussian basis functions used to model the variation of the length scale. The predictive mean is shown in (b) and the variation of the length scale is shown in (c). A non-linear map approach was then applied to the data. The predictive mean and variation of the length scale are shown in (d) and (e) respectively. The errors bars shown in these figures are all 1σ error bars. A Hessian approximation was used to calculate the error bars in (c) and (e).

Method	log Evidence	RMS error	Average RMS
Stationary	-227.9	0.5286	0.5176
Non-stat	-198.5	0.3585	0.3881
Non-linear	-204.7	0.3187	0.3369

Table 3.2: Comparison of Models for Neuron Spike Data : The table shows the comparison of the evidence and RMS error for a standard stationary Gaussian process, a non-stationary Gaussian process and a Gaussian process using a non-linear map on the input space for fitting the noisy neuron spike data shown in Figure 3.8. The average RMS error was calculated over four data sets generated in exactly the same way as the data in Figure 3.8 but using different seeds for the random noise.

MacKay and Takeuchi (1994) encountered difficulties with multiple optima in their hyperparameter space when tackling this problem. Such multiple optima also occur in both the non-stationary and non-linear map approaches. The results produced by the different optima were very similar, any variation usually only occurring at the edges of the data. The variation in solutions for different initial conditions was slightly more pronounced for the non-linear approach than for the non-stationary covariance function approach. The results shown in Figure 3.11 were generated using the set of hyperparameters with the highest posterior probability that were found in ten runs with varying initial conditions for each method.

In order to compare the stationary, non-stationary and non-linear map approaches, the evidence and the RMS error for the initial 120 "noise-free" data points were calculated. The stationary Gaussian process had a prior on the length scale giving the solution in Figure 3.8(b). The results can be seen in Table 3.2. The evidence for the stationary model is significantly lower and the RMS error is significantly higher than for the other two approaches. There is little to choose between the performance of the non-stationary and non-linear map approaches.

Conclusions

The non-stationary and non-linear Gaussian processes are useful models for data with spatially varying length scales. Both gave evidence values and RMS errors significantly less than the stationary Gaussian process. The non-linear map approach might be favoured as the length scales are more directly interpretable. Due to the behaviour of the prefactor in equation 3.82, the interpretation of the $\{r_l(\mathbf{x})\}$ is not quite so obvious. The non-linear map approach also does not encounter difficulties with rapidly varying length scales although, in both non-linear map and non-stationary cases, the model assigns a small length scale to regions close to the peak as well as to the peak itself, suggesting that the basis functions used were too broad.

3.10.4 Vibrational ground state properties of weakly bound molecular systems

The Schrödinger equation has long been used to calculate the behaviour of quantum mechanical systems. However in order to perform such calculations, a certain amount of knowledge is required concerning the potential energy surface of the system in question. Unfortunately there is a lack of accurate multi-dimensional surfaces. Individual points on the surfaces can be calculated using spectroscopy or *ab initio* techniques and approximations can be made to the surfaces, for example simple harmonic oscillator approximations. However weakly bound systems, e.g. van der Waals clusters (see Figure 3.12), are strongly anharmonic and the simple harmonic approximation gives poor results. Thus the standard approach in the weakly bound case is to calculate the potential at

many points and then use this data to fit some interpolation model to approximate the potential energy surface as a whole.

Once we have obtained a model for the potential energy surface, we then need to find a method to determine the properties of the system. Diffusion Monte Carlo (DMC) (Anderson 1975; Suhm and Watts 1991) is a general and exact method for determining the ground state of the time-independent Schrödinger equation. It takes into account the couplings and anharmonicities embodied by the potential energy surface and hence is well suited to weakly bound calculations. DMC also has very favourable scaling with cluster size, making it a useful tool for studying large systems. However, typical DMC simulations require of the order of 10^7 evaluations of the potential. Consequently some easily calculable expression for the potential energy surface is required. This expression must also be accurate as small discrepancies can significantly bias final results.

A variety of different schemes have been used in the past to model the potential energy surface of weakly bound systems. However most of these schemes have either been specific to certain groups of systems (Tully 1980; Bowman and Kuppermann 1975; Connor *et al.* 1975), been viable only for a limited number of dimensions (Ho and Rabitz 1996) or required a large amount of *ab initio* data (Gregory and Clary 1995). We are principally interested in modelling the potential energy surface of large weakly bound systems. Large systems have many degrees of freedom and hence a high dimensional potential energy surface. Also the intermolecular interactions of weakly bound systems are awkward to describe and so general models are difficult to come up with. Finally, high level *ab initio* theory is required for weakly bound systems and so data is computationally expensive.

The constraints on size of training data set, the possibly high dimensionality of the input space, the lack of noise present in the problem and the low cost of making predictions make Gaussian process models suitable for a problem such as this. To begin with, we will discuss the physics behind the problem and briefly describe the Diffusion Monte Carlo algorithm. We will then consider a Bayesian neural network and a Gaussian processes treatment of the problem.

Diffusion Monte Carlo

The time-dependent N-body Schrödinger equation can be written

$$-i\hbar\frac{\partial\psi(\mathbf{x},t)}{\partial t} = \sum_{n=1}^{N}\frac{\hbar^2}{2m_n}\nabla_n^2\psi(\mathbf{x},t) - \left[V(\mathbf{x}) - V_{ref}\right]\psi(\mathbf{x},t), \tag{3.91}$$

where $\psi(\mathbf{x},t)$ is the wavefunction, $V(\mathbf{x})$ is the potential energy, $\mathbf{x}$ is the position co-ordinates of the bodies and m_n is the mass of the n^{th} body. V_{ref} is a reference energy introduced to stabilize the ground state. This is similar in form to the Diffusion equation with a first-order rate term,

$$\frac{\partial C}{\partial t} = \mathcal{D}\frac{\partial^2 C}{\partial x^2} - kC. \tag{3.92}$$

Transforming into imaginary time $\tau = it/\hbar$, equation 3.91 becomes

$$\frac{\partial\psi(\mathbf{x},\tau)}{\partial\tau} = \sum_{n}\frac{\hbar^2}{2m_n}\nabla_n^2\psi(\mathbf{x},\tau) - \left[V(\mathbf{x}) - V_{ref}\right]\psi(\mathbf{x},\tau). \tag{3.93}$$

This equation has solutions which can be written as a spatially varying part multiplied by a exponentially varying time part,

$$\psi(\mathbf{x},\tau) = \sum_{k} A_k\phi_k(\mathbf{x})e^{-E_k\tau}, \tag{3.94}$$

where $\phi_k(\mathbf{x})$ and E_k are eigenfunctions and eigenvalues of the time-independent Schrödinger equation. As it is analogous to a diffusion equation, we can simulate the time-dependent Schrödinger

equation using a random walk. For large τ, the dominant term in equation 3.94 corresponds to the lowest eigenstate as the higher energy states will have decayed to zero.

The DMC algorithm begins with a population of non-interacting random walkers. The linearity of the Schrödinger equation allows us to use more than one in each simulation. Each walker or "replica" has weight 1.0 and is located randomly on the potential energy surface. Each iteration of the algorithm corresponds to an increase in imaginary time of $\Delta\tau$ and proceeds in three steps. Firstly each replica makes a random movement ($\Delta\mathbf{x}$) to simulate the kinetic energy term in equation 3.91. The atoms of the replica are displaced by a random three dimensional vector drawn from a spherical Gaussian distribution with zero mean and variance $\Delta\tau/m_n$. The second step is to update the weight of each replica using the formula

$$w_i = w_i \exp\left[-(V(\mathbf{x}_i) - V_{ref})\Delta\tau\right]. \tag{3.95}$$

This is known as continuous weighting. In the third step, when the weight of any replica falls beneath a certain critical threshold it is removed and the replica with the highest weight is duplicated. This duplicate is then allowed to evolve independently. The reference potential V_{ref} is also adjusted in order to maintain an approximately constant population of replicas.

Properties of the ground state of the system can be found by allowing the simulation to reach an equilibrium distribution of replicas and then averaging over a large number of iterations. The ground state energy can be obtained from the average of the potential energy, V($\mathbf{x}$), and the wavefunction is equivalent to the average of a histogram of the positions of the replicas at each iteration.

Modelling the Potential Energy Surface

Initially, we attempted to model potential energy surfaces using a Bayesian neural network based on the ideas of MacKay (1992d). The software package `BACKPROB`[2] was used. We will briefly describe the methods used but a full discussion of the work can be found in Brown *et al.* (1996).

The 4 dimensional $(HF)_2$ potential (with the monomer bonds held rigid) due to Barton and Howard (1982) is well known and so served as a good test case. Figure 3.12 shows the co-ordinate system used to describe the molecule. A training set of 240 *ab-initio* data points was used initially, drawn almost randomly from the input space. There was a slight bias towards the region containing the bottom of the potential well as greater accuracy is required in this region for the DMC simulation. All the *ab-initio* data was generated using the CADPAC package (Amos 1982). In the investigation performed by Brown, an MLP with 32 hidden nodes (the number chosen using Evidence maximization (MacKay 1992d)) was trained on the data set using `BACKPROB`. In order to improve accuracy of the solution, Brown added a further 60 points to the data set and re-trained the network. The whole procedure took about 2.5 hours on a Pentium P75 workstation.

Brown found that the mean error over 50000 random points from the chemically interesting parts of the surface ranged from $\pm 15cm^{-1}$ to $\pm 25cm^{-1}$. This is comparable with the error expected from even high level *ab initio* calculations and so we can conclude that the neural network's model of the potential energy surface is sufficiently accurate. DMC simulations were then performed using the neural network surface and the actual Barton and Howard potential. Brown then calculated a range of observable quantities for both simulations and found no statistical difference between the results (see Table 3.3). Hence we can conclude that the neural network performs well at the task of modelling the $(HF)_2$ potential.

Having achieved high quality results using neural networks, we then attempted to apply Gaussian processes to the problem. We used the covariance function given in equation 2.20 and focused

[2] `BACKPROB` was written by Martin Oldfield and David MacKay and is available by anonymous `ftp` at wol.ra.phy.cam.ac.uk or 131.111.48.24 from the pub/backprob directory.

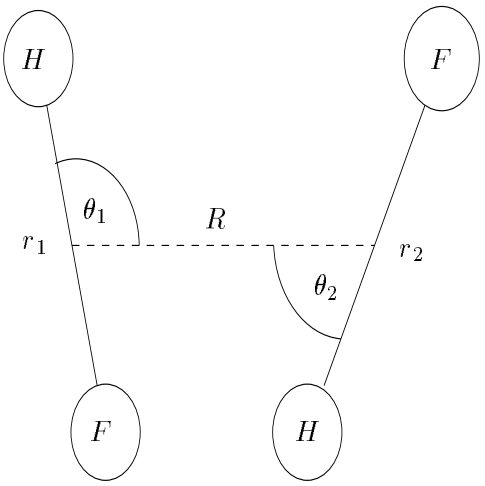


Figure 3.12: Co-ordinate system of a van der Waals molecule: The figure shows the co-ordinate system for the $(HF)_2$ van der Waals molecule. For the rigid body form of DMC, r_1 and r_2 are fixed. θ_1, θ_2, R and ψ are allowed to vary where ψ is the torsional angle in the plane of the paper.

Property	Barton Howard	Neural Net	Gaussian process
D_0 (cm^{-1})	1088(3)	1084(4)	1081(5)
$\langle R \rangle$ (Å)	2.78(2)	2.79(2)	2.78(2)
$\langle \theta_1 \rangle$ (deg)	33(3)	34(3)	38(8)
$\langle \theta_2 \rangle$ (deg)	59(4)	64(6)	67(7)
$0.5(B+C)$ cm^{-1}	0.215(2)	0.214(2)	0.214(2)

Table 3.3: Comparison of Neural Network, Gaussian process and Barton Howard Potentials The table shows the theoretical dissociation energy, D_0, the dimer bond length, R, the two bond angles, θ_1 and θ_2, and two of the moments of inertia of the molecule, A and 0.5 (B+C), of two isotopomers of $(HF)_2$ calculated using the original Barton Howard PES and the Barton Howard surface re-fitted using a neural network and a Gaussian process. The neural network and Gaussian process were trained using 307 and 520 data points taken from the Barton Howard potential respectively. The uncertainty in the last digit is shown in parentheses. These results are taken from Brown *et al.* (1996).

again on the four dimensional potential energy surface of rigid $(HF)_2$. All of the hyperparameters of the Gaussian process including the noise variance were optimized using conjugate gradients. The results obtained (see Table 3.3) are very similar to the neural network results. However, Brown found that more training data was required for the Gaussian processes than was needed for the neural networks to achieve similar accuracy (520 points compared with 307). The training of the Gaussian process took about 1.5 hours on a Pentium P75 workstation.

An important extension of any analysis of weakly bound molecules is to molecules with a large number of degrees of freedom and hence a high dimensional potential energy surface. In non-rigid $(HF)_2$ the monomer bond lengths, r_1 and r_2, are allowed to vary and so the potential energy surface is six dimensional. An analytic potential for 6D $(HF)_2$ exists due to Bunker *et al.* (1988). Neural networks found it extremely difficult to model the six dimensional surface accurately, consistently producing unacceptably high deviations from the true potential even using large quantities of data. Gaussian processes produced significantly better results, achieving a mean of error of $\pm 20 cm^{-1}$ using 3000 data points.

Conclusions

The application of Gaussian processes (and neural networks) is a significant advance in the area of modelling weakly bound molecules. Calculations that previously required 10^7 costly *ab initio* potential evaluations and which typically took several months can now be performed using a few hundred data points in under a day without appreciable loss of accuracy. The accuracy of the neural networks to low dimensional problems is greater than that of the Gaussian processes for a fixed amount of data. However the ability of Gaussian processes to deal accurately with higher dimensional surfaces than neural networks suggests that Gaussian processes offer a useful technique for the continuing study of more complicated systems.

Chapter 4

Classification using Gaussian processes

Previously we have discussed the application of Gaussian processes to regression. We now turn to classification problems. We shall consider only supervised classification problems which can be described as follows; Assume that we have some data $\mathcal{D}$ which consists of inputs $\{\mathbf{x}_n\}_{n=1}^N$ in some space, real or discrete, and corresponding targets t_n which are categorical variables which can take a finite and known number of, say, integer values. We wish to model the data and hence find the predictive distribution over classes for a new point $\mathbf{x}_{N+1}$. We shall begin by discussing the case of binary (two class) classification. The multiple class case will be discussed in chapter five.

4.1 Binary Classifiers

While the application of Gaussian processes to regression is fairly straightforward, it is less obvious how to proceed for classification. The standard Bayesian approach is to model this data using a Bayesian conditional classifier which predicts t conditional on $\mathbf{x}$. To do this we assume the existence of a function $a(\mathbf{x})$ which models the 'logit' $\log \frac{P(t=1|\mathbf{x})}{P(t=0|\mathbf{x})}$ as a function of $\mathbf{x}$. Thus

$$P(t=1|\mathbf{x}, a(\mathbf{x})) = \frac{1}{1+\exp(-a(\mathbf{x}))} \tag{4.1}$$

To complete the model we place a prior distribution over the unknown function $a(\mathbf{x})$. There are two approaches to this. In the standard parametric approach, $a(\mathbf{x})$ is a parameterized function $a(\mathbf{x};\mathbf{w})$ where the parameters $\mathbf{w}$ might be, say, the weights of a neural network or the coefficients of a linear expansion $a(\mathbf{x};\mathbf{w}) = \sum_h w_h \phi_h(\mathbf{x})$. We place a prior probability distribution $P(\mathbf{w})$ over the parameters which is traditionally taken to be Gaussian (MacKay 1995b).

Alternatively we can model $a(\mathbf{x})$ directly using a Gaussian process. This involves modelling the joint distribution of $\{a(\mathbf{x}_n)\}$ with a Gaussian distribution.

$$P(\mathbf{a}_N|\Theta) = \frac{1}{Z_{gp}} \exp\left(-\frac{1}{2}\mathbf{a}_N^T \mathbf{C}_N^{-1} \mathbf{a}_N\right) \tag{4.2}$$

where $\mathbf{a}_N = (a(\mathbf{x}_1), a(\mathbf{x}_2), \cdots, a(\mathbf{x}_N))$ and $\mathbf{C}_N$ is an appropriate positive definite covariance matrix that is a function of the inputs $\{\mathbf{x}_n\}$ and a set of hyperparameters Θ.

For classification models there are two well-established approaches to Bayesian inference: Gaussian approximations centred on the posterior modes (MacKay 1992b) and Monte Carlo methods (Neal 1996). Barber and Williams (1996) have implemented classifiers based on Gaussian process priors using Laplace approximations and Neal (1997) has used a Monte Carlo approach to implement a Gaussian process classifier.

We shall consider another approach based on the methods of Jaakkola and Jordan (1996). This involves finding tractable upper and lower bounds for the unnormalized posterior density

$P(\{t_N\}|\mathbf{a}_N)P(\mathbf{a}_N)$. These bounds are parameterized by variational parameters which are adjusted in order to obtain the tightest possible fit. Using the normalized versions of the optimized bounds we then compute approximations to the predictive distributions. This is similar to an ensemble learning approach (Hinton and van Camp 1993; MacKay 1995a) in that we are adapting approximations to the posterior distribution of unknown variables.

4.2 Variational Gaussian Process Model

Let us look at the Gaussian process approach in more detail. We wish to make predictions at new points given the data, i.e., we want to find the predictive distribution $P(t_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ where $\mathbf{x}_{N+1} \notin \mathcal{D}$. This can be written

$$P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D}) = \int P(t_{N+1} = 1|a(\mathbf{x}_{N+1}))\, P(a(\mathbf{x}_{N+1})|\mathbf{x}_{N+1}, \mathcal{D})\, \mathrm{d}a(\mathbf{x}_{N+1}) \tag{4.3}$$

The first term in the integrand is a sigmoid (see equation 4.1). The second term can found by integrating out the dependence on $\{a(\mathbf{x}_n)\}$ (for $n = 1 \cdots N$) in the joint posterior distribution $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$:

$$P(a(\mathbf{x}_{N+1})|\mathbf{x}_{N+1}, \mathcal{D}) = \int P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}) d^N\mathbf{a}_N. \tag{4.4}$$

where $\mathbf{a}_{N+1} = (\mathbf{a}_N, a(\mathbf{x}_{N+1}))$. We can write the joint posterior distribution as a product over the data points and a prior on the function $a(\mathbf{x})$.

$$P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}) = \frac{1}{Z} P(\mathbf{a}_{N+1}) \prod_{n=1}^{N} P(t_n|a(\mathbf{x}_n)) \tag{4.5}$$

where Z is an appropriate normalizing constant. The prior on $\mathbf{a}_{N+1}$ shall be assumed to be a Gaussian process prior of the form

$$P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \{\mathbf{x}_n\}, \Theta) = \frac{1}{Z_{gp}} \exp\left(-\frac{1}{2}\mathbf{a}_{N+1}^T \mathbf{C}_{N+1}^{-1} \mathbf{a}_{N+1}\right) \tag{4.6}$$

where $\mathbf{C}_{N+1}$ is the $(N+1) \times (N+1)$ covariance matrix for $\mathbf{a}_{N+1}$ and $(\mathbf{C}_{N+1})_{mn} = \mathcal{C}_f(\mathbf{x}_m, \mathbf{x}_n; \Theta)$. The covariance function $\mathcal{C}_f$ has the form

$$\mathcal{C}_f(\mathbf{x}_m, \mathbf{x}_n; \Theta) = \theta_1 \exp\left\{-\frac{1}{2}\sum_{l=1}^{L} \frac{\left(x_m^{(l)} - x_n^{(l)}\right)^2}{r_l^2}\right\} + \theta_2 + \delta_{mn} J \tag{4.7}$$

where $\Theta = (\theta_1, \theta_2, J, \{r_l\})$ are appropriate hyperparameters and $x_m^{(l)}$ is the l^{th} component of the vector $\mathbf{x}_m$. Unlike the regression covariance function, we assume $a(\mathbf{x})$ to be noise free. However we do introduce a "jitter" term $\delta_{mn} J$ (Neal 1997) to make the matrix computations well-conditioned. We choose the magnitude of J to be small in comparison to θ_1.

The product of sigmoid functions in equation 4.5 generally makes the integral in equation 4.4 analytically intractable. Barber and Williams (1996) use a Laplace approximation in order to evaluate this integral. Neal (1997) uses Monte Carlo techniques to approximate the integral. Instead we introduce an upper and lower bound to the sigmoid function in order to find analytic approximations to the posterior $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ and therefore find approximations to the posterior probability $P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D})$.

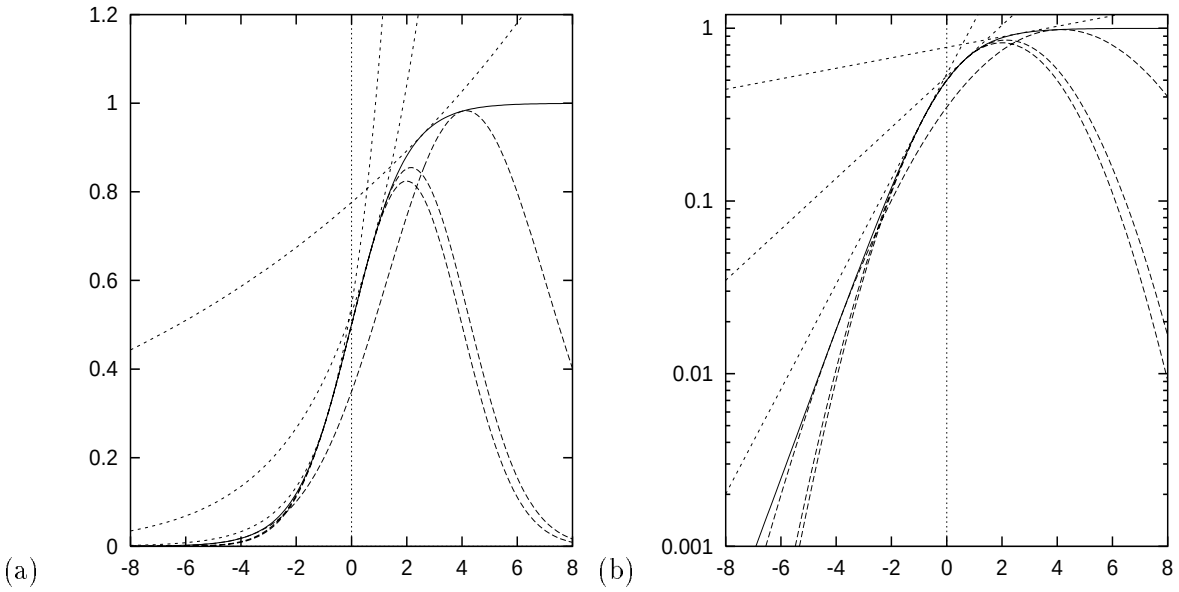


Figure 4.1: Bounds on sigmoid function : (a) shows the sigmoid function (solid black line) and upper and lower bounds as defined in equations 4.8 and 4.9 for $\nu = 0.05, 1.0, 4.0$ and $\mu = 0.07, 0.34, 0.7$. (b) shows the same bounds on a log scale, showing how the log upper bound is the tangent of the log sigmoid function at one point and how the log lower bound is a quadratic which touches the log sigmoid function at two points.

We can define upper and lower bounds on the sigmoid, i.e., on $P(t = 1|a(\mathbf{x}))$, as follows (Jaakkola and Jordan 1996):

$$P(t_n = 1|a(\mathbf{x}_n)) \geq Q(t_n = 1|a_n, \nu_n) = g(\nu_n) \exp\left[(a_n - \nu_n)/2 - \lambda(\nu_n)(a_n^2 - \nu_n^2)\right] \quad (4.8)$$

$$P(t_n = 1|a(\mathbf{x}_n)) \leq R(t_n = 1|a_n, \mu_n) = \exp\left(\mu_n a_n - \mathcal{H}_2(\mu_n)\right) \quad (4.9)$$

where $a_n = a(\mathbf{x}_n)$ and

$$g(a_n) = \frac{1}{1 + \exp(-a_n)} \quad (4.10)$$

$$\lambda(\nu_n) = [g(\nu_n) - 1/2]/2\nu_n. \quad (4.11)$$

μ_n and ν_n are variational parameters with μ_n in the interval $[0, 1]$ and $\mathcal{H}_2(x)$ is the binary entropy function

$$\mathcal{H}_2(x) = -x \log x - (1 - x) \log(1 - x). \quad (4.12)$$

Figure 4.1 shows some examples of the upper and lower bounds for a range of values of the variational parameters. The bounds on $P(t_n = 0|a(\mathbf{x}_n))$ follow from the above by substituting $-a_n$ for a_n as

$$P(t_n = 0|a(\mathbf{x}_n)) = 1 - P(t_n = 1|a(\mathbf{x}_n)) = 1 - \frac{1}{1 + \exp(-a(\mathbf{x}_n))} = \frac{1}{1 + \exp(a(\mathbf{x}_n))} \quad (4.13)$$

We can use the two distributions Q and R to bound each factor $P(t_n|a(\mathbf{x}_n))$ in equation 4.5 by introducing two variational parameters μ_n and ν_n for each data point. Note we do not introduce

any variational parameters for the point $\mathbf{x}_{N+1}$ (we shall introduce a separate approximation when extracting the prediction later). We can then use these bounds on the sigmoid to approximate the posterior probability $P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D})$.

In the example in Figure 4.2, we have one ($N = 1$) training data point $\mathbf{x}_1 = (1, 0)$ which belongs to class 1, i.e., $t_1 = 1$. We want to make inferences at a new point $\mathbf{x}_{N+1} = (2, 0)$. To do this we use the upper and lower variational bounds defined in equations 4.8 and 4.9 to approximate the posterior distribution $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ where $\mathbf{a}_{N+1} = (a(\mathbf{x}_1), a(\mathbf{x}_{N+1}))$. We place a Gaussian process prior ($\theta_1 = 10.0$, $r_1 = 1$, $\theta_2 = 0$ and $\theta_3 = 10^{-3}$) on $a(\mathbf{x})$ (see Figure 4.2(a)). Figure 4.2(b) shows the true posterior distribution $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ for the training set $\mathcal{D} = (\mathbf{x}_1, t_1)$. Notice that the posterior is no longer Gaussian and hence difficult to integrate over. We can then calculate an approximation to the posterior distribution using the Gaussian lower bound on $P(t_1|a(\mathbf{x}_1))$, a sigmoid function (see Figure 4.2(c)). We choose a value for the variational parameter $\nu_1 = 1.0$. The approximation is a Gaussian which fits the true posterior well near the origin but drops off too quickly as we move away from the origin. This is because the Gaussian lower bound to the sigmoid drops off to zero as $a(\mathbf{x}_1)$ tends to ∞ whereas the sigmoid tends asymptotically to 1.0. The approximation to the posterior is actually a different shape than the prior distribution on $\mathbf{a}_{N+1}$. We can also calculate an approximation to the posterior using the exponential upper bound with the variational parameter $\mu_1 = 0.34$ (see Figure 4.2(d)). This approximation is also Gaussian but it is the same shape as the prior; only the position of the centre has been shifted. The approximation has the majority of its probability mass in the right place but the tail of the Gaussian furthest from the origin is too long. This is because the exponential bound to the sigmoid tends to ∞ as $a(\mathbf{x}_1)$ tends to ∞ rather than to 1.0. Neither ν_1 nor μ_1 have been optimized for this example in order to highlight the differences between the approximating distributions and the true posterior.

4.3 Making Predictions

4.3.1 Predictions based on the lower bound

We can write down the following approximation to $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$:

$$P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}) \simeq \mathcal{P}_Q(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\}, \Theta) = \frac{1}{Z'} P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \{\mathbf{x}_n\}, \Theta) \prod_{n=1}^{N} Q(t_n|a(\mathbf{x}_n), \nu_n) \tag{4.14}$$

where Z' is the appropriate normalizing constant. Now using the previous definition of $Q(t = 1|a, \nu)$ we can write

$$\prod_{n=1}^{N} Q(t_n|a(\mathbf{x}_n), \nu_n) \propto \exp\left[-\mathbf{a}_N^T \Lambda_N \mathbf{a}_N + \mathbf{d}^T \mathbf{a}_N\right] \tag{4.15}$$

where Λ_N is a diagonal matrix with elements $(\lambda(\nu_1), \lambda(\nu_2), \cdots, \lambda(\nu_N))$, $\lambda(.)$ being defined in equation 4.11. The summation over n is from 1 to N as there are no variational parameters associated with the input vector $\mathbf{x}_{N+1}$. The vector $\mathbf{d}$ reflects whether the training input vector $\mathbf{x}_n$ is a member of class 0 or class 1 and has components $d_n = \frac{1}{2}(-1)^{t_n+1}$. In the above equation we have ignored any terms that are independent of $\mathbf{a}_N$ and $a(\mathbf{x}_{N+1})$. Such terms simply contribute to a normalising constant which we shall not need to evaluate when making predictions.

Introducing the Gaussian process prior we can write

$$\mathcal{P}_Q(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\}, \Theta) \propto \exp\left[-\frac{1}{2}\mathbf{a}_{N+1}^T (\mathbf{C}_{N+1}^{-1} + 2\Lambda_{N+1})\mathbf{a}_{N+1} + \mathbf{d}^T \mathbf{a}_N\right] \tag{4.16}$$

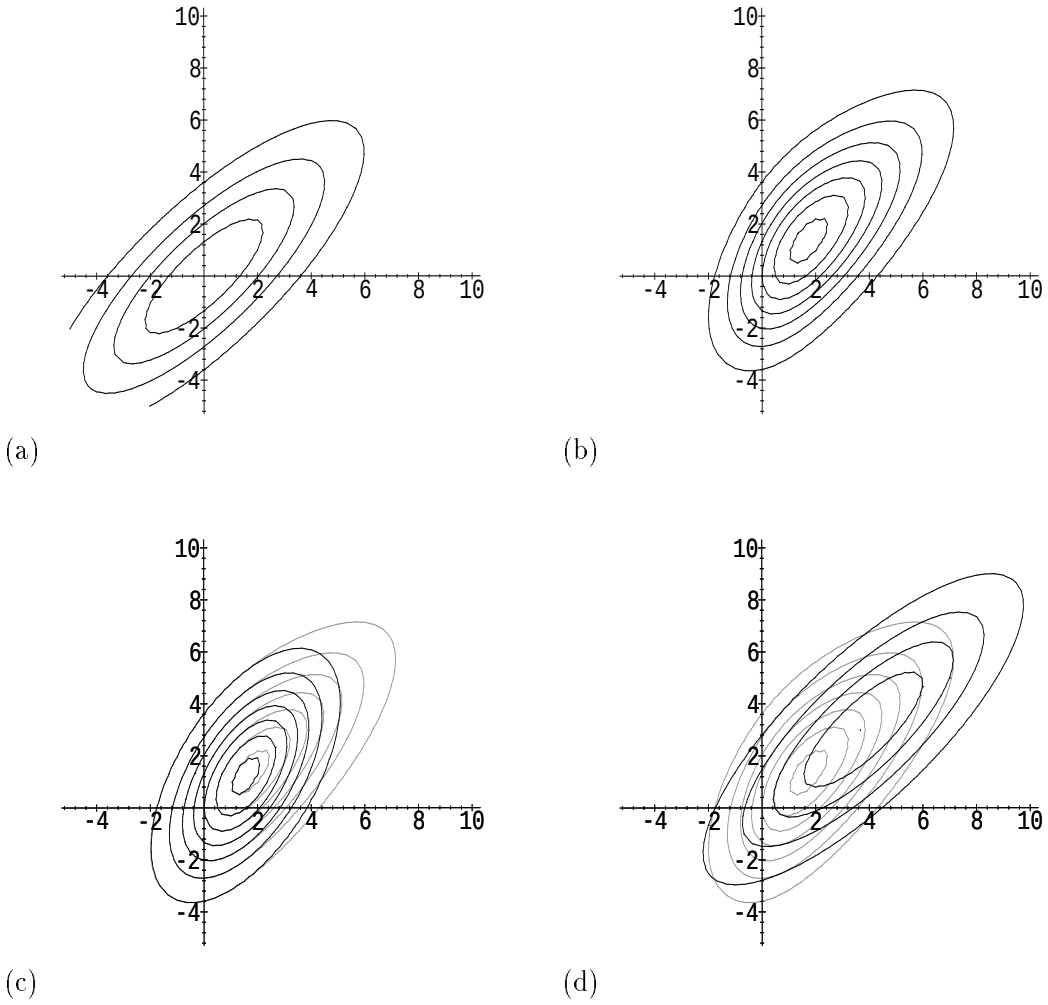


Figure 4.2: Approximating $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ using Variational Bounds : (a) shows the Gaussian process prior on $\mathbf{a}_{N+1} = (a(\mathbf{x}_1), a(\mathbf{x}_{N+1}))$. In all the plots, $a(\mathbf{x}_1)$ is along the horizontal axis and $a(\mathbf{x}_{N+1})$ is along the vertical. (b) shows the true posterior distribution $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ for a training set consisting of one data point $\mathbf{x}_1$ with $t_1 = 1$. The solid black lines in (c) show the contours of the approximation to the posterior distribution obtained using the Gaussian lower bound on $P(t_1|a(\mathbf{x}_1))$, a sigmoid function, for $\nu_1 = 1.0$, superimposed over the true posterior, shown in grey. (d) shows the approximation to the posterior using the exponential upper bound with $\mu_1 = 0.34$, superimposed over the true posterior, shown in grey.

where $\Lambda_{N+1} = \mathrm{diag}(\lambda(\nu_1), \lambda(\nu_2), \cdots, \lambda(\nu_N), 0)$ We can see that the marginal distribution over $a(\mathbf{x}_{N+1})$ is Gaussian. Using the block form of the inverse of a matrix, we can express the inverse of $\mathbf{C}_{N+1}$ in terms of $\mathbf{C}_N^{-1}$ and then, by integrating over $\mathbf{a}_N$, find the mean a_l^{MP} and variance s_l^2 of $\mathcal{P}_Q(a(\mathbf{x}_{N+1})|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\}, \Theta)$,

$$a_l^{MP} = \mathbf{k}_{N+1}^T \mathbf{H}_N^{-1} \mathbf{d} \tag{4.17}$$

$$s_l^2 = \kappa + 2\mathbf{k}_{N+1}^T \mathbf{H}_N^{-1} \Lambda_N \mathbf{k}_{N+1} \tag{4.18}$$

where

$$\mathbf{H}_N = \mathbf{I} + 2\Lambda_N \mathbf{C}_N \tag{4.19}$$

and

$$\mathbf{k}_{N+1} = (\mathcal{C}_f(\mathbf{x}_1, \mathbf{x}_{N+1}; \Theta) \cdots, \mathcal{C}_f(\mathbf{x}_N, \mathbf{x}_{N+1}; \Theta)) \tag{4.20}$$

$$\kappa = \mathcal{C}_f(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}; \Theta). \tag{4.21}$$

We now substitute the Gaussian lower bound approximation $\mathcal{P}_Q(a(\mathbf{x}_{N+1})|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\}, \Theta)$ into equation 4.3 and use an approximation to the integral of the product of a Gaussian and a sigmoid (MacKay 1992d),

$$\int \mathrm{d}x \; g(x) \mathrm{Gaussian}(a_l^{MP}, s_l^2) \simeq g(\tau(s_l) a_l^{MP}) \tag{4.22}$$

where $\tau(s) = 1/\sqrt{1 + \pi s^2/8}$. This provides us with an approximation to the predictive distribution:

$$P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D}) \simeq g(\tau(s_l) a_l^{MP}) \tag{4.23}$$

We should note that, although we have been using the lower bound on $P(t_n = 1|a(\mathbf{x}_n))$, we have not generated a lower bound on the probability $P(t_{N+1} = 1|\mathcal{D})$ but an approximation to it. Looking back at equation 4.14 we see that $\mathcal{P}_Q(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\}, \Theta)$ is an approximation to $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ rather than a lower bound. The normalising constant Z' (a lower bound to $P(\mathcal{D}|\Theta)$) is in the denominator of equation 4.14 and the factors $Q(t_n|a(\mathbf{x}_n), \nu_n)$ (lower bounds to $P(t_n|a(\mathbf{x}_n))$) are in the numerator. Hence the conflicting approximations introduced by these two terms mean any solution derived from $\mathcal{P}_Q(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\nu_n\})$ is an approximation not a lower bound.

4.3.2 Predictions based on the upper bound

Now let us consider the approximation to $P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D})$ found using the upper bound. We can write

$$P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}) \simeq \mathcal{P}_R(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\mu_n\}, \Theta) = \frac{1}{Z''} P(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \{\mathbf{x}_n\}, \Theta) \prod_{n=1}^{N} R(t_n|a(\mathbf{x}_n), \mu_n) \tag{4.24}$$

Using the previous definition of $R(t = 1|a, \mu)$,

$$\prod_{n=1}^{N} R(t_n|a(\mathbf{x}_n), \mu_n) \propto \exp\left[\mathbf{b}^T \mathbf{a}_N\right] \tag{4.25}$$

where $b_i = \mu_i(-1)^{t_i+1}$, reflecting which class the training data is in. As in the lower bound case, we have ignored terms that are independent of $\mathbf{a}_N$ and $a(\mathbf{x}_{N+1})$ as they simply contribute to a normalising constant that we shall not need to evaluate when making predictions.

Introducing the Gaussian process prior we write

$$\mathcal{P}_R(\mathbf{a}_{N+1}|\mathbf{x}_{N+1}, \mathcal{D}, \{\mu_n\}, \Theta) \propto \exp\left[-\frac{1}{2}\mathbf{a}_{N+1}^T\mathbf{C}_{N+1}^{-1}\mathbf{a}_{N+1} + \mathbf{b}^T\mathbf{a}\right]. \tag{4.26}$$

As with the lower bound, we can use the block form of the inverse to calculate the mean a_u^{MP} and the variance s_u^2 of the approximate marginal distribution $\mathcal{P}_R(a(\mathbf{x}_{N+1})|\mathcal{D}, \{\mu_n\}, \Theta)$.

$$a_u^{MP} = -s_u^2\mathbf{k}_{N+1}^T\mathbf{C}_N^{-1}\mathbf{b} \tag{4.27}$$

$$s_u^2 = \left(\kappa - \mathbf{k}_{N+1}^T\mathbf{C}_N^{-1}\mathbf{k}_{N+1}\right)^{-1} \tag{4.28}$$

We use equation 4.22 again to calculate the approximation

$$P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D}) \simeq g(\tau(s_u)a_u^{MP}). \tag{4.29}$$

4.4 Determining the Parameters

We can now derive two approximations to $P(t_{N+1} = 1|\mathbf{x}_{N+1}, \mathcal{D})$ using our upper and lower bounds for the sigmoid function. In order to make use of these approximations we need to find appropriate values for the variational parameters and the hyperparameters of the covariance function.

Consider the upper and lower bounds on $P(\mathcal{D}|\Theta)$,

$$Z' \leq P(\mathcal{D}|\Theta) \leq Z'', \tag{4.30}$$

where

$$Z' = \int d^N\mathbf{a}_N P(\mathbf{a}_N|\Theta) \prod_{n=1}^{N} Q(t_n|a(\mathbf{x}_n), \nu_n) \tag{4.31}$$

$$Z'' = \int d^N\mathbf{a}_N P(\mathbf{a}_N|\Theta) \prod_{n=1}^{N} R(t_n|a(\mathbf{x}_n), \mu_n) \tag{4.32}$$

Note that Z' and Z'' are the normalising constants from equations 4.14 and 4.24 respectively.

We wish to set the variational parameters $\{\nu_n\}$ and $\{\mu_n\}$ so that Z' and Z'' are as tight bounds on $P(\mathcal{D}|\Theta)$ as possible. We can do this by maximizing Z' with respect to $\{\nu_n\}$ and minimizing Z'' with respect to $\{\mu_n\}$. We would also like to set the hyperparameters of the covariance function to their most probable values given the data (a Monte Carlo approach could also be used (Barber and Williams 1996)). This is not possible as we do not have an analytic expression for $P(\mathcal{D}|\Theta)$. However we can maximize Z' and Z'' with respect to Θ to obtain approximations to the most probable Θ given the data.

We now calculate the derivatives of the lower and upper bound on $P(\mathcal{D}|\Theta)$ with respect to the variational parameters and the hyperparameters of the covariance function. Given these derivatives we can then use a gradient based optimization algorithm such as conjugate gradients to optimize the bounds.

4.4.1 The Lower Bound

We can write Z' in the following manner

$$Z' = \prod_n g(\nu_n) \exp\left[-\left(\nu_n/2 - \lambda(\nu_n)\nu_n^2\right)\right] \int d^N\mathbf{a}_N \frac{1}{Z_{gp}} \exp\left[-\frac{1}{2}\mathbf{a}_N^T(\mathbf{C}_N^{-1} + 2\Lambda_N)\mathbf{a}_N + \mathbf{d}^T\mathbf{a}_N\right] \tag{4.33}$$

The integral in Z' is tractable as the bound on the sigmoid function is a Gaussian. Hence

$$\log(Z') = \sum_n \left(\log(g(\nu_n)) - \nu_n/2 + \lambda(\nu_n)\nu_n^2\right) + \frac{1}{2}\mathbf{d}^T\left(\mathbf{C}_N^{-1} + 2\Lambda_N\right)\mathbf{d} - \frac{1}{2}\log\det(\mathbf{I} + 2\Lambda_N\mathbf{C}_N). \tag{4.34}$$

We can show that for symmetric $\mathbf{C}_N$

$$\begin{aligned}\frac{\partial \log Z'}{\partial \nu_k} &= -\frac{1}{4}\left[1 + 2g_k\left((1-g_k)\nu_k - 1\right)\right] - \mathbf{d}^T\mathbf{C}_N\mathbf{H}_N^{-1}\frac{\partial \Lambda_N}{\partial \nu_k}\mathbf{C}_N\mathbf{H}_N^{-1}\mathbf{d} \\ &\quad -\frac{1}{2}\mathrm{tr}\left(\mathbf{H}_N^{-1}\frac{\partial \mathbf{H}_N}{\partial \nu_k}\right) \qquad (4.35)\\ \frac{\partial \log Z'}{\partial \theta} &= \frac{1}{2}\left(\mathbf{H}_N^{-1}\mathbf{d}\right)^T\frac{\partial \mathbf{C}_N}{\partial \theta}\mathbf{H}_N^{-1}\mathbf{d} - \frac{1}{2}\mathrm{tr}\left(\mathbf{H}_N^{-1}\frac{\partial \mathbf{H}_N}{\partial \theta}\right) \qquad (4.36)\end{aligned}$$

where $g_k = g(\nu_k)$ and $\theta \in \Theta$ is some generic hyperparameter of the covariance function. We should note that in order to calculate either of these derivatives we need only perform one inversion, i.e., find $\mathbf{H}_N^{-1}$ for any given $\{\nu_k\}$ and Θ.

4.4.2 The Upper Bound

Let us now consider the upper bound Z''.

$$Z'' = \exp\left[-\sum_n \mathcal{H}_2(\mu_n)\right] \int d^N\mathbf{a}_N \frac{1}{Z_{gp}} \exp\left(-\frac{1}{2}\mathbf{a}_N^T\mathbf{C}_N^{-1}\mathbf{a}_N + \mathbf{b}^T\mathbf{a}_N\right) \tag{4.37}$$

Again the integral in Z'' is tractable.

$$\log Z'' = -\sum_n \mathcal{H}_2(\mu_n) + \frac{1}{2}\mathbf{b}^T\mathbf{C}_N\mathbf{b} \tag{4.38}$$

We can calculate the derivatives of Z'',

$$\begin{aligned}\frac{\partial \log Z''}{\partial \mu_k} &= \log\left(\frac{\mu_k}{1-\mu_k}\right) + (-1)^{t_k+1}\sum_n (\mathbf{C}_N)_{kn}\, b_n \qquad (4.39)\\ \frac{\partial \log Z''}{\partial \theta} &= \frac{1}{2}\mathbf{b}^T\frac{\partial \mathbf{C}_N}{\partial \theta}\mathbf{b} \qquad (4.40)\end{aligned}$$

The evaluation of these derivatives is trivial as no inversion is required.

4.4.3 Optimization Procedure

We have calculated the derivatives of the upper and lower bounds on $P(\mathcal{D}|\Theta)$ with respect to their variational parameters and with respect to the hyperparameters of the covariance function. However we have not yet addressed the question of how we should go about the optimization of these hyperparameters.

The lower bound case is simple as we can simultaneously optimize Z' with respect to the variational parameters $\{\nu_n\}$ and the hyperparameters of the covariance function Θ using a gradient based optimization algorithm. The upper bound presents us with a slightly more difficult problem. Minimization of Z'' with respect to the variational parameters $\{\mu_n\}$ is straightforward. However maximizing Z'' with respect to the hyperparameters of the Gaussian process is problematic. For example Z'' increases to infinity as θ_1 increases and hence no finite maximum exists.

An alternative approach to the optimization of Z'' is to fix the hyperparameters Θ at the values determined by the optimization of Z' and optimize Z'' with respect to the variational parameters $\{\mu_n\}$ alone. The justification for this comes from the likely relative quality of the upper and lower bounds. Let us look back at figure 4.1. We can see that the lower bound provides a good fit for a larger range of values of a than the upper bound. This is especially well demonstrated by the log plot.

Thus we use an optimization procedure in which Z' is maximized with respect to the variational parameters $\{\nu_n\}$ and the hyperparameters Θ and then Z'' is minimized with respect to the variational parameters $\{\mu_n\}$ using the hyperparameters Θ generated by the maximization of Z'.

4.5 Examples

4.5.1 1D Toy Example

In order to illustrate various features of the Variational Gaussian process Classifier (VGC), we generated a one dimensional toy problem (see Figure 4.3(a)). We ran the optimization procedure described in Section 4.4.3 using conjugate gradients from 20 different sets of initial conditions to guard against the presence of multiple minima in hyperparameter space. It is the authors' experience that, given sensible priors on the hyperparameters (a Gamma distribution $Ga(r_l|10.0, 3.0)$ was used in this case to limit the length scales $\{r_l\}$ to moderate values), such multiple minima rarely occur and did not occur in this case. The average of the predictions made by each of the 20 runs (which were almost identical) can be seen in Figure 4.3(b).

The results are much as expected. The VGC identifies the regions which belong to class 1 and class 0 and models the transitions between these regions smoothly with no over-fitting. The VGC also behaves well where there is no training data. On either side of the training data, the classifier's predictions for $P(t_{N+1} = 1|\mathcal{D})$ tend to 0.5. It should be noted that the lower bound gives more confident predictions than the upper bound. This is expected as the lower bound is generally tighter than the upper bound.

4.5.2 The CRABS and PIMA examples

We next tried our method on two well known classification problems: the *Leptograpsus* crabs and Pima Indian diabetes datasets[1]. The results for both tasks, together with comparisons with several other methods (from Barber and Williams (1996), Ripley (1994) and Ripley (1996)) are given in Table 4.1.

[1] Available from `http://markov.stats.ox.ac.uk/pub/PRNN`.

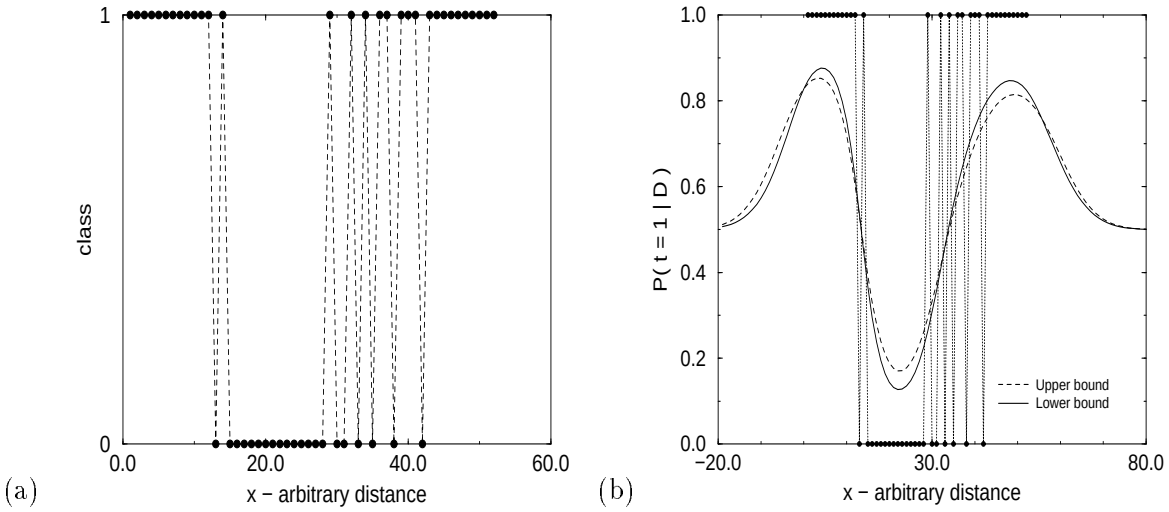


Figure 4.3: 1D Toy Example : (a) shows the training data used in this binary toy example. The data were chosen specifically so that there would be a sharp transition from class 1 to class 0 ($x \simeq 13$) and a more gentle transition from class 0 to class 1 ($29 \leq x \leq 43$). (b) shows the upper bound and lower bound approximations of $P(t_{N+1} = 1|\mathcal{D})$. There are some interesting features to note. Firstly the lower bound makes more confident predictions than the upper bound. This is expected as the lower bound is generally tighter than the upper bound. Secondly the classifier makes sensible predictions in regions where it has little data, i.e., $P(t_{N+1} = 1|\mathcal{D})$ tends to 0.5. Thirdly $P(t_{N+1} = 1|\mathcal{D})$ does not swing rapidly from 0 to 1 on each data point on the boundary between classes but instead gives a smooth transition, i.e. there is no over-fitting. Finally, even in the regions where there are a significant number of 0's or 1's, the classifier does not over-fit the data and make wildly over-confident predictions.

Method	Crab		Pima	
	Error	% Error	Error	% Error
Neural Network (1)	3 ± 1.7	2.5 ± 1.4	-	-
Neural Network (2)	5 ± 2.1	4.2 ± 1.8	-	-
Neural Network (3)	-	-	75	22.6
Linear Discriminant	8 ± 2.7	6.7 ± 2.3	67 ± 7.3	20.2 ± 2.2
MARS (degree = 1)	8 ± 2.7	6.7 ± 2.3	75 ± 7.6	22.6 ± 2.3
2 Gaussian Mixture	-	-	64 ± 7.2	19.3 ± 2.2
HMC Gaussian process	3 ± 1.7	2.5 ± 1.4	68 ± 7.4	20.5 ± 2.2
VGC	4 ± 2	3.3 ± 1.6	70 ± 7.4	21.1 ± 2.2

Table 4.1: Pima and Crabs Results : The table shows the performance of a range of different classification models on the Pima and Crabs problems (Ripley (1994) and Ripley (1996)). The number of classification errors and the percentage of errors both refer to the test set. The error bars given are calculated using binomial statistics. The results quoted for the VGC are those obtained using the approximations from the lower bound. The HMC Gaussian process is the classifier described in Barber and Williams (1996).

In the *Leptograpsus* crabs problem we attempted to classify the sex of crabs based upon six characteristics. 200 labelled examples are split into a training set of 80 and a test set of 120. The performance of the VGC is not significantly different from the best of the other methods. The Pima Indian diabetes problem involved the prediction of the occurrence of diabetes in women of Pima Indian heritage based on seven characteristics. 532 examples were available and these were split into 200 training examples and 332 test examples. 33% of the population were reported to have diabetes so an error rate of 33% can be achieved by declaring all examples to be non-diabetic. The VGC achieved an error rate of 21% - again comparable with the best of the other methods.

4.5.3 Weld Strength Example

Hot cracking can occur in welds as they cool. The occurrence of such cracks depends on the chemical composition of the weld metal, the cooling rate and the weld geometry. We wish to predict whether a given weld will crack by examining the dependence of cracking on 13 specific characteristics of a weld. In a previous treatment of this problem using Bayesian neural networks (Ichikawa *et al.* 1996) the relationship between cracking and carbon content was highlighted and compared with experimental data. We performed a similar analysis using VGCs.

An initial test was performed using a training set of 77 examples and a test set of 77 examples. The test error rates and test log likelihoods for the VGC and the Bayesian neural network approach (Ichikawa *et al.* 1996) can be seen in Table 4.2 where the test log likelihood is defined as

$$\text{test log likelihood} = \sum_{n=1}^{N_{test}} t_n \log(\hat{t}_n) + (1 - t_n) \log(1 - \hat{t}_n) \tag{4.41}$$

where t_n is the true test set classification (either 0 or 1) and $\hat{t}_n$ is the prediction $P(t_n = 1|\mathcal{D})$.

Method	Test Error	Log Likelihood
Bayesian Neural Network	8	-23.6
Variational GP Classifier	10/10	-25.73/-31.57

Table 4.2: Weld Strength Classification Problem : This table shows the test error and log likelihood scores of the VGC and the Bayesian neural network of Ichikawa *et al.* (1996). The two results given for the VGC correspond to the approximations using the lower and upper bound respectively.

The performance of the VGC is slightly inferior to that of the Bayesian neural network. However the neural network result was obtained using a committee of four networks, a large amount of experimentation with different architectures and parameter settings was performed and the four networks found *with the best test error* were used in the committee. The VGC results required no such experimentation. 20 runs of the VGC with differing initial conditions were performed to guard against multiple minima. The results quoted in Table 4.2 are from the first run but all of the runs produced almost identical results.

We then trained the VGC using all 154 examples in order to model the carbon dependence of the weld strength. A plot of the carbon dependence was obtained by fixing all the inputs to their mean values other than carbon which was allowed to vary. The result shown in Figure 4.4 was much as expected. It shows the reduction in the probability of cracking at intermediate carbon concentrations (as found experimentally) and also shows a tendency for increased strength at low carbon concentrations. The corresponding results of Ichikawa *et al.* (1996) are also shown in Figure 4.4.

4.6 Conclusions

We have shown that Gaussian processes can be used to produce effective binary classifiers. The results using the VGC are comparable to the best of current classification models. Using Gaussian processes we obtain a parameterization of our model that is easily interpretable. Another point to note is that VGCs are moderately simple to implement and use. The small amount of the model that needs to be determined by hand (generally only the priors on the hyperparameters and the choice of covariance function) makes VGCs a useful tool for automated tasks where fine tuning for each problem is not possible. However we do not sacrifice any performance for this simplicity.

As with the Gaussian process approach to regression, one problem with this method is the computational cost associated with inverting an $N \times N$ matrix exactly. To avoid this problem we can use the approximate inversion techniques described in Section 3.2. Another problem with the variational approach is the profileration of variational parameters when dealing with very large amounts of the data. Unfortunately, it is not clear how the number of variational parameters might be reduced if we wish to take the approach of finding upper and lower bounds for the sigmoid function. One possible improvement would be to find variational bounds for a product of sigmoid functions (see equation 4.5) rather than for the individual sigmoid functions themselves. However this approach has not been pursued.

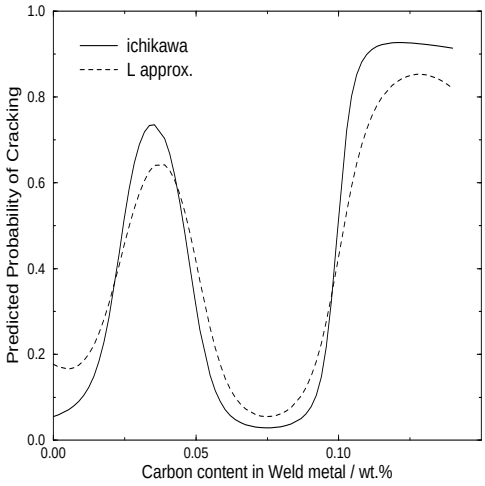


Figure 4.4: Carbon Dependence of Weld Strength : The two plots shown on this graph are the results obtained in Ichikawa *et al.* (1996) and those found using the lower bound approximation of a VGC. Both have the same large scale features but the VGC makes less confident predictions where the training data density tails off near zero carbon content.

Chapter 5

Multi-class classification

Having considered the binary case, let us now investigate the problem of multi-class classification using Gaussian processes. In the binary case we used the sigmoid function to define $P(t=1|a(\mathbf{x}))$ where $a(\mathbf{x})$ was a real valued function of $\mathbf{x}$. We then found exponential family parameterized upper and lower bounds to the sigmoid function which allowed us to integrate out $a(\mathbf{x})$ and hence obtain approximations to the posterior probability $P(t_{N+1}=1|\mathcal{D},\mathbf{x}_{N+1})$. In the multi-class case, we take a similar variational approach. For a problem with I classes, we define the probability of the point $\mathbf{x}$ being in class j in terms of a *softmax* function (Bridle 1989):

$$P(t=j|a^{(1)}(\mathbf{x}),a^{(2)}(\mathbf{x}),\cdots,a^{(I)}(\mathbf{x}))=\frac{\exp(a^{(j)}(\mathbf{x}))}{\sum_{i=1}^{I}\exp(a^{(i)}(\mathbf{x}))} \tag{5.1}$$

Here we have introduced I functions $a^{(1)},a^{(2)},\cdots,a^{(I)}$ to model the probability of being in any one class across the input space. Please note that throughout this chapter, superscripts will refer to different classes whereas subscripts will refer to different data points. We shall also denote the vector $(a_1^{(i)},a_2^{(i)},\cdots,a_N^{(i)})$ as $\mathbf{a}^{(i)}$ and the vector $(a_n^{(1)},a_n^{(2)},\cdots,a_n^{(I)})$ as $\mathbf{a}_n$ where $a_n^{(i)}\equiv a^{(i)}(\mathbf{x}_n)$.

Now consider some data $\mathcal{D}=\{\mathbf{x}_n,t_n\}_{n=1}^N$ where the t_n can take integer values 1 to I. Given a new point $\mathbf{x}_{N+1}$, we wish to calculate the probability of being in any one class j. Using the softmax definition above, we write

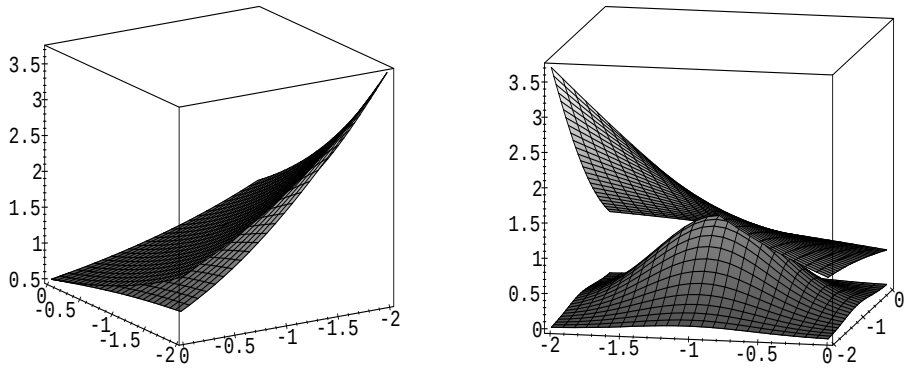
$$P(t_{N+1}=j|\mathcal{D},\mathbf{x}_{N+1})=\int P(t_{N+1}=j|\mathcal{D},\mathbf{a}_{N+1})P(\mathbf{a}_{N+1}|\mathcal{D},\mathbf{x}_{N+1})\,\mathrm{d}\mathbf{a}_{N+1}. \tag{5.2}$$

$P(t_{N+1}=j|\mathcal{D},\mathbf{a}_{N+1})$ is a softmax function. We can obtain $P(\mathbf{a}_{N+1}|\mathcal{D},\mathbf{x}_{N+1})$ by marginalizing over the distribution

$$P(\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_{N+1}|\mathcal{D},\mathbf{x}_{N+1})=\frac{1}{Z}\prod_{i=1}^{I}P(\mathbf{a}^{(i)},a^{(i)}(\mathbf{x}_{N+1})|\{\mathbf{x}_n\},\mathbf{x}_{N+1})\prod_{n=1}^{N}P(t_n|\mathbf{a}_n,\mathbf{x}_n). \tag{5.3}$$

We assume a Gaussian process prior on $\mathbf{a}^{(i)}$ with hyperparameters $\Theta^{(i)}$. Thus the distribution $P(\mathbf{a}^{(i)},a^{(i)}(\mathbf{x}_{N+1})|\{\mathbf{x}_n\},\mathbf{x}_{N+1})$ is an $N+1$ dimensional Gaussian with covariance matrix $\mathbf{C}_{N+1}^{(i)}$.

Unfortunately, even given a Gaussian process prior, such a marginalization is difficult. In the binary case, it involves the integral of a product of sigmoid functions. For the multi-class case, we are faced with an integral of a product of softmax functions. This is analytically intractable and so we shall approximate the integral using variational bounds on the softmax function which belong to the exponential family.



(a) (b)

Figure 5.1: Lower and Upper Bounds on $Z^{-1}(\mathbf{a})$: (a) shows $Z^{-1}(\mathbf{a})$ and an example of the exponential upper bound given by equation 5.5 ($\alpha_1 = -1.0, \alpha_2 = -1.0$). (b) also shows $Z^{-1}(\mathbf{a})$ and a similar example of the Gaussian lower bound given by equation 5.8 ($\mu_1 = -1.0, \mu_2 = -1.0, \gamma = 0.95$).

5.1 The Upper Bound

In the derivation of both the upper and lower bounds on the softmax function, we shall consider bounding

$$\frac{1}{Z(\mathbf{a})} = \frac{1}{\sum_{i=1}^{I} e^{a^{(i)}}}. \tag{5.4}$$

An upper bound is straightforward to calculate as $\log Z^{-1}$ is a concave function of the $a^{(i)}$. Given any arbitrary point $\mathbf{a} = \alpha$, we can calculate the gradient of $\log Z^{-1}$ at that point with respect to $\mathbf{a}$. Because it is concave, $\log Z^{-1}$ is bounded above by any plane which touches $\log Z^{-1}$ at α and has gradient $\nabla(\log Z^{-1}(\alpha))$. Hence

$$\frac{1}{Z(\mathbf{a})} \leq \frac{1}{Z(\alpha)} \exp\left(\sum_{i=1}^{I} g^{(i)}(a^{(i)} - \alpha^{(i)})\right) \tag{5.5}$$

where

$$g_i = -\frac{\partial \log Z(\mathbf{a})}{\partial a^{(i)}}\bigg|_{\mathbf{a}=\alpha} = \frac{-e^{\alpha^{(i)}}}{Z(\alpha)} \tag{5.6}$$

The exponent of the upper bound is linear in $\mathbf{a}$ and so, when combined with the Gaussian process prior, gives a bound on $P(\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_{N+1}|\mathcal{D}, \mathbf{x}_{N+1})$ which is simple to integrate. Figure 5.1(a) shows an example of the upper bound.

5.2 The Lower Bound

Finding a lower bound is slightly more complicated than finding an upper bound. It is difficult to generalize the Gaussian lower bound to the sigmoid given in equation 4.8 to the multi-dimensional case. However we shall retain the idea of using a Gaussian as a lower bound for Z^{-1}.

Figure 5.2(a) shows a contour plot of $\log Z^{-1}(\mathbf{a})$. Here we can see that it has two rather useful properties. Firstly, the gradient of $\log Z^{-1}$ in the $a^{(1)}$ direction for $a^{(1)} < a^{(2)}$ is close to zero, shown by the almost horizontal contours. Secondly, the gradient in the $a^{(2)}$ direction in the same region is close to but always less negative than -1.0. We can use these two pieces of information to try to bound $\log Z^{-1}$ below using a quadratic. To determine this quadratic, we pick a point $\mu = (\mu^{(1)}, \cdots, \mu^{(I)})$ and a height h where

$$h = \log \left[\frac{\gamma}{\sum_{i=1}^{I} e^{\mu^{(i)}}} \right] \tag{5.7}$$

and where γ is in the interval $[0, 1)$. Now consider a slice passing through μ along the $a^{(k)}$ axis (see Figure 5.2(b)). We can find the intercept of the line $y = h$ with $y = \log(1/(D + e^{a^{(k)}}))$ where $D = \sum_{i \neq k} \exp(\mu^{(i)})$. We can then construct a line $?^{(k)}$ with gradient -1.0 starting at the intercept which we know will not cross $\log Z^{-1}$. We can then calculate the curvature of our bounding quadratic in the $a^{(k)}$ direction by constructing a quadratic which has value h at μ and which just touches $?^{(k)}$. Performing this procedure in every $a^{(k)}$ direction leads to a separable quadratic form which, when exponentiated, gives the Gaussian

$$\frac{\gamma}{\sum_{i=1}^{I} e^{\mu^{(i)}}} \exp \left(- \sum_{i=1}^{I} \frac{(a^{(i)} - \mu^{(i)})^2}{2\sigma_i^2} \right) \tag{5.8}$$

where $\sigma_i^2 = 2(r^{(i)} - \mu^{(i)})$ and

$$r^{(i)} = \log \left[\frac{\sum_{j=1}^{I} e^{\mu^{(j)}}}{\gamma} - \sum_{j \neq i} e^{\mu^{(j)}} \right] \tag{5.9}$$

Equation 5.8 is Gaussian and so, assuming it is a true lower bound on $Z(\mathbf{a})$ (see the next section), we can obtain a bound on $P(\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_{N+1} | \mathcal{D}, \mathbf{x}_{N+1})$ which is simple to integrate. An example of the lower bound can be seen in Figure 5.1(b).

5.3 Tests of the Lower Bound

The question of whether equation 5.8 is a true lower bound for $Z(\mathbf{a})$ is difficult to answer. For the I class case, let us take a horizontal slice through the log lower bound and $\log Z^{-1}$. Fixing all the $a^{(i)}$ other than $a^{(j)}$ and $a^{(k)}$, the contours of $\log Z^{-1}$ and the log lower bound at a height $\log h$, where $h < \gamma / Z(\mu)$, are given by

$$a_Z^{(k)} = \log \left(\frac{1}{h} - e^{a^{(j)}} - C_t \right) \tag{5.10}$$

$$a_l^{(k)} = \sqrt{2}\sigma_k \left[\log \left(\frac{Z(\mu) h}{\gamma} \right) - \frac{(a^{(j)} - \mu^{(j)})^2}{2\sigma_j^2} - C_l \right]^{\frac{1}{2}} + \mu^{(k)} \tag{5.11}$$

respectively where

$$C_t = \sum_{i \neq j,k} e^{a^{(i)}} \tag{5.12}$$

$$C_l = \sum_{i \neq j,k} \frac{(a^{(i)} - \mu^{(i)})^2}{2\sigma_i^2}. \tag{5.13}$$

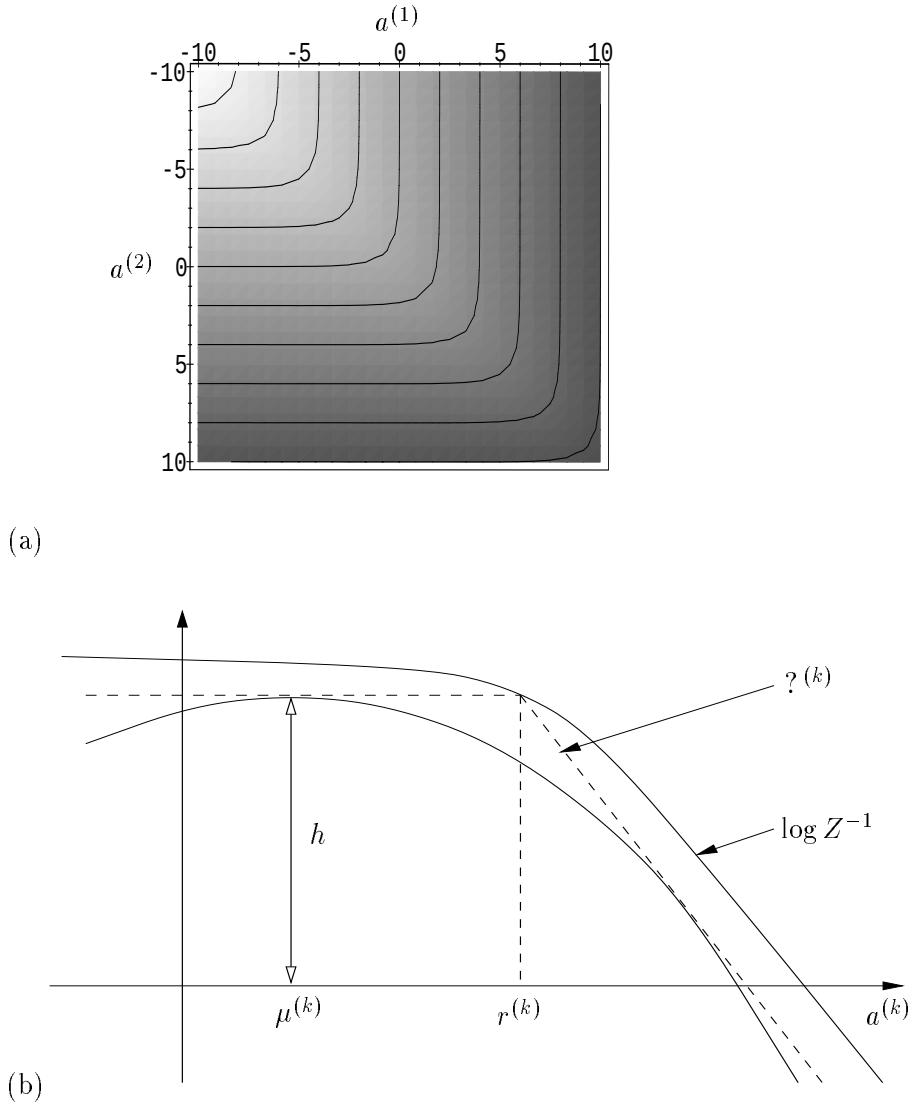


Figure 5.2: Lower Bound for $\log Z^{-1}$ **:** (a) shows the contours of $\log Z^{-1}(\mathbf{a})$. We can see that, in the region $a^{(1)} < a^{(2)}$, the contours are almost horizontal and equally spaced. The gradient in the $a^{(2)}$ direction in this region is very close to but always less negative than -1.0. For a given point $\mathbf{a} = \mu$, (b) shows a slice through $\log Z^{-1}(\mathbf{a})$ in the $a^{(k)}$ direction passing through μ. By considering the intercept of the line $y = h$, we can construct $\Gamma^{(k)}$ with gradient -1.0. We can then define a bounding quadratic which has value h at $\mu^{(k)}$ and which just touches $\Gamma^{(k)}$.

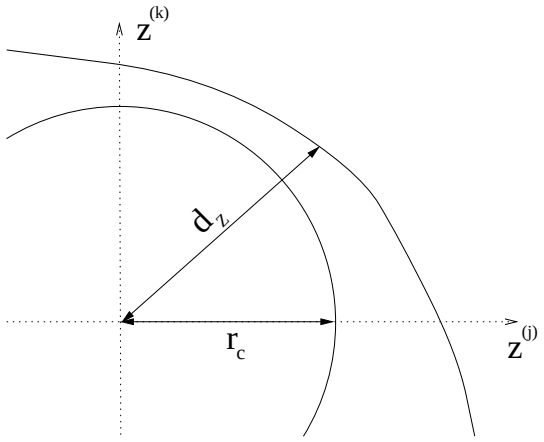


Figure 5.3: Contours of $\log Z^{-1}$ and the log lower bound : The figure shows the contour of $\log Z^{-1}$ and the contour of the log lower bound in the $(z^{(j)}, z^{(k)})$ plane at height $Z^{-1} = h < \gamma/Z(\mu)$. The contour of the log lower bound is a circle of radius r_c centred at the origin.

Using the transformation

$$a^{(i)} = \sqrt{2}\sigma_i z^{(i)} + \mu^{(i)}, \tag{5.14}$$

the contours for the log lower bound become circles centred at the origin (see Figure 5.3):

$$z_l^{(k)} = \sqrt{\log \frac{Z(\mu)h}{\gamma} - C_l - (z^{(j)})^2} \tag{5.15}$$

To establish that the lower bound is valid, we must prove that the distance of closest approach of the contour of $\log Z^{-1}$ from the origin is smaller than the radius of the circular contour of the log lower bound:

$$(z_Z^{(k)})^2 + 2\sigma_k^2 \left\{ \log \left(\frac{1}{h} - C_t - \exp\left(\sqrt{2}\sigma_j z^{(j)} + \mu^{(j)}\right)\right) - \mu^{(k)} \right\} \geq r_c^2 \tag{5.16}$$

where

$$r_c^2 = \log \frac{Z(\mu)h}{\gamma} - C_l \tag{5.17}$$

If we could prove equation 5.16 is true for all values of $\{a^{(i)}\}$, $\{\mu^{(i)}\}$ and h then this would establish the validity of the lower bound (the value of γ can be fixed without loss of generality). Unfortunately establishing the inequality analytically is difficult. Instead a series of numerical tests were performed on lower bounds for one to fifteen classes inclusive. For each number of classes, 100,000 random samples were taken from $\{a^{(i)}\}$, $\{\mu^{(i)}\}$ and h. γ was fixed at 0.9. From these initial points, the vertical distance between Z^{-1} and the lower bound was minimized with respect to $\{a^{(i)}\}$, $\{\mu^{(i)}\}$ and $h \leq Z^{-1}(\mu)$ using a conjugate gradient algorithm. In all of these minimizations, no case was found where the inequality in equation 5.16 was violated. Based upon this evidence, it is my strong belief that the lower bound is valid and it is used without question throughout the rest of this chapter.

5.4 Making Predictions

For a given point $\mathbf{x}_{N+1}$ we are trying to find the posterior probability of this point belonging to class j: $P(t_{N+1} = j|\mathbf{x}_{N+1}, \mathcal{D})$. Using the bounds on Z^{-1} derived above, we can write

$$P(t_n|\mathbf{a}_n) \geq Q(t_n|\mathbf{a}_n, \mu_n, \gamma_n) = \frac{\gamma}{Z(\mu_n)} \exp\left(-\frac{1}{2}\sum_{i=1}^{I} \frac{(a^{(i)}(\mathbf{x}_n) - \mu_n^{(i)})^2}{2\sigma_i^2} + a^{(t_n)}(\mathbf{x}_n)\right) \quad (5.18)$$

$$P(t_n|\mathbf{a}_n) \leq R(t_n|\mathbf{a}_n, \alpha_n) = \frac{1}{Z(\alpha_n)} \exp\left(\sum_{i=1}^{I} g_n^{(i)}(a^{(i)}(\mathbf{x}_n) - \alpha_n^{(i)}) + a^{(t_n)}(\mathbf{x}_n)\right) \quad (5.19)$$

$\{\alpha_n^{(i)}\}$ and $\{\mu_n^{(i)}, \gamma_n\}$ are the variational parameters for the upper and lower bounds respectively. We can now substitute these bounds into equation 5.3 in order to approximate the posterior. Let us deal with each bound separately.

5.4.1 Predictions based on the Upper Bound

Substituting the upper bound in equation 5.19 into equation 5.3, we obtain

$$P(\mathbf{a}_{N+1}|\mathcal{D}, \mathbf{x}_{N+1}) \simeq \frac{1}{Z_R}\prod_{i=1}^{I}\int \exp\left[-\frac{1}{2}(\mathbf{v}^{(i)})^T(\mathbf{C}_{N+1}^{(i)})^{-1}\mathbf{v}^{(i)} + (\mathbf{u}^{(i)})^T\mathbf{a}^{(i)}\right] d\mathbf{a}^{(i)} \quad (5.20)$$

where $\mathbf{v} = (\mathbf{a}^{(i)}, a^{(i)}(\mathbf{x}_{N+1}))$, $\mathbf{u}^{(i)}$ is a vector with elements $u_n^{(i)} = g_n^{(i)} + \delta_{t_n,i}$ and $\mathbf{C}_{N+1}^{(i)}$ is the $(N+1)\times(N+1)$ covariance matrix associated with class i. Z_R is an appropriate normalizing constant. Performing the integral, we obtain a separable Gaussian distribution for the elements of $\mathbf{a}_{N+1}$ with mean $\overline{a}_u^{(i)}$ and variance $v_u^{(i)}$ for $a_{N+1}^{(i)}$ given by

$$\overline{a}_u^{(i)} = (\mathbf{k}_{N+1}^{(i)})^T\mathbf{u}^{(i)} \quad (5.21)$$

$$v_u^{(i)} = \kappa \quad (5.22)$$

where $\mathbf{k}_{N+1}^{(i)} = (C(\mathbf{x}_{N+1}, \mathbf{x}_1; \Theta^{(i)}), \cdots, C(\mathbf{x}_{N+1}, \mathbf{x}_N; \Theta^{(i)}))$ and $\kappa = C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}; \Theta^{(i)})$. We will discuss how to make use of this distribution after examining the lower bound case.

5.4.2 Predictions based on the Lower Bound

Substituting the lower bound in equation 5.18 into equation 5.3, we obtain

$$P(\mathbf{a}_{N+1}|\mathcal{D}, \mathbf{x}_{N+1}) \simeq \frac{1}{Z_Q}\prod_{i=1}^{I}\int \exp\left[-\frac{1}{2}(\mathbf{v}^{(i)})^T\left((\mathbf{C}_{N+1}^{(i)})^{-1} + \mathbf{G}_{N+1}^{(i)}\right)\mathbf{v}^{(i)} + (\mathbf{w}^{(i)})^T(\mathbf{G}^{(i)})^{-1}\mathbf{a}^{(i)}\right] d\mathbf{a}^{(i)} \quad (5.23)$$

$\mathbf{G}^{(i)}$ is an $N\times N$ diagonal matrix with elements $\mathbf{G}_{nn}^{(i)} = 1/(\sigma_n^{(i)})^2$ where $\sigma_n^{(i)}$ is the standard deviation of the lower bound at data point n for class i. $\mathbf{G}_{N+1}^{(i)}$ is the $(N+1)\times(N+1)$ extension of $\mathbf{G}^{(i)}$ where $(\mathbf{G}_{N+1}^{(i)})_{N+1,N+1} = 0$. Z_Q is an appropriate normalizing constant. The vector $\mathbf{w}^{(i)}$ has elements $w_n^{(i)} = \mu_n^{(i)} + \delta_{t_n,i}(\sigma_n^{(i)})^2$ and $\mathbf{C}_{N+1}^{(i)}$ and $\mathbf{v}^{(i)}$ are defined in the same way as for the upper bound.

Performing the integral, we obtain a separable Gaussian distribution for the elements of $\mathbf{a}_{N+1}$ with mean $\overline{a}_l^{(i)}$ and variance $v_l^{(i)}$ for $a_{N+1}^{(i)}$ given by

$$\overline{a}_l^{(i)} = (\mathbf{k}_{N+1}^{(i)})^T (\mathbf{H}^{(i)})^{-1} \mathbf{G}^{(i)} \mathbf{w}^{(i)} \tag{5.24}$$

$$v_l^{(i)} = \kappa - (\mathbf{k}_{N+1}^{(i)})^T (\mathbf{H}^{(i)})^{-1} \mathbf{G}^{(i)} \mathbf{k}_{N+1}^{(i)} \tag{5.25}$$

$\mathbf{H}^{(i)}$ is an $N \times N$ matrix defined by

$$\mathbf{H}^{(i)} = \mathbf{I} + \mathbf{G}^{(i)} \mathbf{C}_N^{(i)} \tag{5.26}$$

where $\mathbf{C}_N$ is the $N \times N$ covariance matrix. $\mathbf{k}^{(i)}$ and κ are defined in the same way as for the upper bound.

5.4.3 Using the Gaussian approximations

We now have two Gaussian approximations for the posterior distribution of $\mathbf{a}_{N+1}$, one calculated using the upper bound and one calculated using the lower. In order to calculate the predictive posterior distribution at $\mathbf{x}_{N+1}$ we need to find the integral of the product of the Gaussian approximation and a softmax function:

$$P(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1}) \simeq \int \frac{\exp(a_{N+1}^{(j)})}{\sum_i \exp(a_{N+1}^{(i)})} \left(\prod_i 2\pi v^{(i)} \right)^{-\frac{1}{2}} \exp\left[-\frac{1}{2} \sum_i \frac{(a_{N+1}^{(i)} - \overline{a}^{(i)})^2}{v^{(i)}} \right] \mathrm{d}\mathbf{a}_{N+1}. \tag{5.27}$$

where $(\overline{a}^{(i)}, v^{(i)}) = (\overline{a}_l^{(i)}, v_l^{(i)})$ or $(\overline{a}_u^{(i)}, v_u^{(i)})$ depending on whether we are using the lower or the upper bound on $P(t_n | \mathbf{a}_n, \mathbf{x}_n)$. Let us consider three approaches to approximating the integral in the above equation.

Joint optimization approach : We can use the upper and lower bounds on $Z^{-1}(\mathbf{a}_{N+1})$ to approximate the softmax term. This requires us not only to perform an optimization of a new set of variational parameters $\{\mu_{N+1}, \gamma_{N+1}, \alpha_{N+1}\}$ for each new point $\mathbf{x}_{N+1}$ but also for each class at each new point. The log lower and upper bounds on the lower bound approximation to $P(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1})$ can be written

$$\begin{aligned} \log \mathcal{P}_Q(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1}, \{\mu_n, \gamma_n\}) \geq\ & \log \gamma_{N+1} - \log Z(\mu_{N+1}) - \frac{1}{2} \sum_i \frac{\left(w_{N+1}^{(i)} - \overline{a}_l^{(i)} \right)^2}{(\sigma_{N+1}^{(i)})^2 + v_l^{(i)}} \\ & - \frac{1}{2} \log \left(1 + \frac{v_l^{(i)}}{(\sigma_{N+1}^{(i)})^2} \right) + \frac{1}{2} \left((\sigma_{N+1}^{(j)})^2 + 2\mu_{N+1}^{(j)} \right) \end{aligned} \tag{5.28}$$

$$\begin{aligned} \log \mathcal{P}_Q(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1}, \{\mu_n, \gamma_n\}) \leq\ & - \log Z(\alpha_{N+1}) + \frac{1}{2} \sum_i (u_{N+1}^{(i)})^2 v_l^{(i)} \\ & + u_{N+1}^{(i)} \overline{a}_l^{(i)} - \alpha_{N+1}^{(i)} g_{N+1}^{(i)} \end{aligned} \tag{5.29}$$

$w_{N+1}^{(i)}$ and $u_{N+1}^{(i)}$ are defined in the previous two sections. By maximizing equation 5.28 with respect to μ_{N+1} and γ_{N+1} and by minimizing equation 5.29 with respect to α_{N+1}, we can obtain upper and lower bounds on $\mathcal{P}_Q(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1}, \{\mu_n, \gamma_n\})$. Thus to obtain predictions at $\mathbf{x}_{N+1}$ using the lower bound, two optimizations have to be performed for each different j, a total of $2I$ optimizations in all. A similar approach can be taken to bounding $\mathcal{P}_R(t_{N+1} = j | \mathcal{D}, \mathbf{x}_{N+1}, \{\alpha_n\})$.

Partial optimization approach : Having to perform $2I$ optimizations to obtain predictions at every new point is far from ideal. An alternative is to ignore the $\exp(a_{N+1}^{(j)})$ term in the numerator of the softmax function and calculate upper and lower bounds on

$$F = \int \frac{1}{\sum_i \exp(a_{N+1}^{(i)})} \left(\prod_i 2\pi v^{(i)} \right)^{-\frac{1}{2}} \exp\left[-\frac{1}{2} \sum_i \frac{(a_{N+1}^{(i)} - \overline{a}^{(i)})^2}{v^{(i)}} \right] \mathrm{d}\mathbf{a}_{N+1}. \tag{5.30}$$

where $(\overline{a}^{(i)}, v^{(i)})$ is equal to $(\overline{a}_l^{(i)}, v_l^{(i)})$ or $(\overline{a}_u^{(i)}, v_u^{(i)})$. The bounds on $\log F$ can be written

$$\begin{aligned} \log F &\geq \log \gamma_{N+1} - \log Z(\mu_{N+1}) - \frac{1}{2} \sum_i \frac{\left(\mu_{N+1}^{(i)} - \overline{a}^{(i)}\right)^2}{(\sigma_{N+1}^{(i)})^2 + v^{(i)}} \\ &\quad -\frac{1}{2} \log \left(1 + \frac{v_l^{(i)}}{(\sigma_{N+1}^{(i)})^2} \right) \qquad (5.31) \\ \log F &\leq -\log Z(\alpha_{N+1}) + \frac{1}{2} \sum_i (g_{N+1}^{(i)})^2 v^{(i)} \\ &\quad + g_{N+1}^{(i)} \left(\overline{a}^{(i)} - \alpha_{N+1}^{(i)} \right) \qquad (5.32) \end{aligned}$$

where $g_{N+1}^{(i)}$ is defined in equation 5.6. We can use the values of the variational parameters obtained by maximizing the lower bound and minimizing the upper bound on F to calculate lower and upper bounds on $\mathcal{P}_Q(t_{N+1} = j|\mathcal{D}, \mathbf{x}_{N+1}, \{\mu_n, \gamma_n\})$ $((\overline{a}^{(i)}, v^{(i)}) = (\overline{a}_l^{(i)}, v_l^{(i)}))$ or $\mathcal{P}_R(t_{N+1} = j|\mathcal{D}, \mathbf{x}_{N+1}, \{\alpha_n\})$ $((\overline{a}^{(i)}, v^{(i)}) = (\overline{a}_u^{(i)}, v_u^{(i)}))$. As F has no dependence on j (the label of the class we want a predictive probability for), we only have to perform two optimizations for each new point $\mathbf{x}_{N+1}$.

Extended MacKay approach : The final approach is based on a generalization of MacKay (1992c)'s approximation to the integral of a Gaussian and a sigmoid:

$$\int \frac{\exp(a^{(j)})}{\sum_i \exp(a^{(i)})} \frac{1}{Z} \exp\left[-\frac{1}{2} \sum_i \frac{(a^{(i)} - \overline{a}^{(i)})^2}{v^{(i)}} \right] \mathrm{d}\mathbf{a} \simeq \frac{\exp(\tau(v^{(j)})\overline{a}^{(j)})}{\sum_i \exp(\tau(v^{(i)})\overline{a}^{(i)})} \tag{5.33}$$

where $\tau(v) = 1/\sqrt{1 + \pi v/8}$. This requires no optimization to make predictions but the approximation is based on the assumption that $\tau(v^{(i)})$ is close to unity for all i. This is reasonable if the values of $a_n^{(i)}$ are believed to lie between ± 1, i.e., if the θ_1 hyperparameter of the Gaussian process is smaller than 1.0. For data that supports extreme predictions, this is unlikely to be true.

A comparison of the three approximations listed above is given for the first toy problem tackled in Section 5.6.

5.5 Determining the Parameters

We have obtained two approximations to the posterior probability $P(t_{N+1}|\mathcal{D}, \mathbf{x}_{N+1})$ using our upper and lower bounds on the softmax function. We now need to find a way to determine the value of the variational parameters and the hyperparameters of the Gaussian process priors.

As with the binary case, we shall consider the upper and lower bounds on $P(\mathcal{D}|\{\Theta^{(i)}\})$:

$$Z' \leq P(\mathcal{D}|\{\Theta^{(i)}\}) \leq Z'' \tag{5.34}$$

where

$$\log Z' = \sum_{i=1}^{I} -\frac{1}{2}\log(\det \mathbf{H}^{(i)}) - \frac{1}{2}(\mathbf{w}^{(i)})^T(\mathbf{H}^{(i)})^{-1}\mathbf{G}^{(i)}\mathbf{w}^{(i)} + \sum_{n=1}^{N} \log \gamma_n - \log Z(\mu_n) + \frac{1}{2}((\sigma_n^{(t_n)})^2 + 2\mu_n^{(t_n)}) \quad (5.35)$$

$$\log Z'' = \sum_{i=1}^{I} \frac{1}{2}(\mathbf{u}^{(i)})^T \mathbf{C}_N^{(i)} \mathbf{u}^{(i)} + \sum_{n=1}^{N} \left[-\log Z(\alpha_n) - \sum_{i=1}^{I} g_n^{(i)} \alpha_n^{(i)} \right] \quad (5.36)$$

The above expressions are obtained in a similar way to those found for the binary case.

Using equations 5.35 and 5.36, we shall set the variational parameters so that the bounds on $P(\mathcal{D}|\{\Theta^{(i)}\})$ are as tight as possible by maximizing $\log Z'$ with respect to $\{\mu_n, \gamma_n\}$ and minimizing $\log Z''$ with respect to $\{\alpha_n\}$. We would also like to maximize Z' and Z'' with respect to the hyperparameters $\{\Theta^{(i)}\}$ of the Gaussian process priors to obtain approximations to the most probable $\{\Theta^{(i)}\}$ given the data.

Maximization of $\log Z'$ with respect to $\{\mu_n, \gamma_n\}$ and the $\{\Theta^{(i)}\}$ is straightforward. This can be done simultaneously using a gradient based optimization routine. However maximization of $\log Z''$ with respect to the hyperparameters $\{\Theta^{(i)}\}$ encounters the same associated problems as the binary case (see Section 4.4.3). So the binary optimization scheme is also used in the multi-class case, i.e., $\log Z'$ is maximized with respect to its variational parameters and the hyperparameters of the Gaussian process priors simultaneously. Then, with the hyperparameters fixed, $\log Z''$ is minimized with respect to its variational parameters.

5.6 Examples

5.6.1 Toy Problem no.1

The training data for the first toy problem was generated by choosing 100 points randomly from the unit square. A class was assigned to each point using the distributions shown in Figure 5.4(a)-(c), yielding the training data shown in Figure 5.4(d). A multi-class variational Gaussian process classifier (MGC) with the covariance function given in equation 4.7 was then trained on the 100 points using the optimization scheme described above. A conjugate gradient optimizer was used for the hyperparameters and variational parameters and the optimization was terminated when the modulus of the gradient of the objective function ($\log Z'$ or $\log Z''$) with respect to the parameters had fallen below a threshold value. No priors were place on the variational parameters but weak Gamma priors and Inverse Gamma priors were placed on the length scales $\{r_l^{(i)}\}$ and $\theta_1^{(i)}$ respectively for each class $(Ga(r_l|0.3, 2.0), Ig(\theta_1|3.0, 3.0))$.

The resulting predictive distributions can be seen in Figure 5.5(a)-(f). It is interesting to note that the predictions of the upper and lower bound are very similar suggesting, unlike the binary case, that the quality of the lower bound is no greater than that of the upper bound.

The predictive distributions shown in Figure 5.5 were calculated using the extended MacKay formula. It is interesting to see which approach for approximating the integral in equation 5.27 is most accurate for this particular example. Table 5.1 shows the RMS error between the true value of the integral based on a lower bound on $P(t_n|\mathbf{a}_n, \mathbf{x}_n)$ and the approximation generated by each approach for 400 points randomly drawn from the unit square. The true value of the integral for each class at each point was found using 10,000 Monte Carlo samples.

The extended MacKay formula predictions and the predictions using the upper bound for both the joint and partial optimization approaches have identical average RMS fractional errors. The

Approach	RMS Fractional Error				Time
	Class 1	Class 2	Class 3	Average	
Extended MacKay	0.0040	0.0048	0.0028	0.0039	0.17
Joint Optimization (average)	0.0142	0.0143	0.0143	0.0143	70.09
Joint Optimization (lower)	0.0320	0.0322	0.0320	0.0321	68.58
Joint Optimization (upper)	0.0039	0.0039	0.0038	0.0039	1.46
Partial Optimization (average)	0.0143	0.0145	0.0143	0.0144	24.77
Partial Optimization (lower)	0.0322	0.0324	0.0321	0.0323	23.75
Partial Optimization (upper)	0.0039	0.0040	0.0039	0.0039	0.96

Table 5.1: Comparison of Prediction Approximation Approaches : The table shows the RMS fractional error between the true predictions of the lower bound of a MGC and the predictions found using the approximations to equation 5.27. The error is average over the three classes and 400 randomly selected points. The true values were found using 10,000 Monte Carlo samples for each class at each point. (average), (lower) and (upper) correspond to predictions made using the average of the upper and lower bound, the lower bound and the upper bound respectively in the context of the approximation of equation 5.27. The time given is the average number of milliseconds of CPU time required to make predictions for all three classes at one point on a SUN Sparc 10.

accuracy of the extended MacKay formula can partially be explained by the fact that none of the predictive probabilities calculated using Monte Carlo sampling are outside the region [0.2,0.8]. The large difference between the error for the predictions using the upper and lower bounds suggests that the upper bound is generally of greater quality than the lower bound. Certainly by just looking at Figure 5.1 this seems to be the case. The CPU time required to make predictions using the extended MacKay formula is, unsurprisingly, low in comparison to the time needed for the other methods. However the predictions made using the upper bound for the joint and partial optimization approaches only require one order of magnitude more time. Given that the extended MacKay formula relies on the dubious assumption that $\tau(v^{(i)}) \simeq 1$, it seems preferable to use the partial optimization approach for prediction.

5.6.2 Toy Problem no.2

The second toy problem uses the data set defined by Neal (1997). Neal drew 1000 random samples, $\mathbf{x}_n = (x_n^{(1)}, x_n^{(2)}, x_n^{(3)}, x_n^{(4)})$, from the four dimensional unit hypercube and then assigned each a class according to the following rules: if $(x_n^{(1)} - 0.4)^2 + (x_n^{(2)} - 0.5)^2 < 0.35$ then $t_n = 0$; else if $0.8x_n^{(1)} + 1.8x_n^{(2)} < 0.6$ then $t_n = 1$; else $t_n = 2$. Note that $x_n^{(3)}$ and $x_n^{(4)}$ have no effect on the class assignment.

400 of the samples were used as training data and the remaining 600 were used as a test set. An MGC was trained using the same covariance function and optimization procedure as before. The predictions were made using the partial optimization approach. After approximately 6 hours of training, the MGC achieved a classification error rate on the 600 test cases of 14% (using both the lower bound and upper bound). This is similar to the 13% reported by Neal for his Monte Carlo Gaussian process classifier but the training time is longer (Neal's Monte Carlo approach took 3.5 hours on a computer of comparable speed). Using only 100 of the training examples, Neal reported a classification error of 17% on the 600 test cases, the MGC managing 19%. The most probable hyperparameters found using the MGC and 400 training samples are shown in Table 5.2. Notice

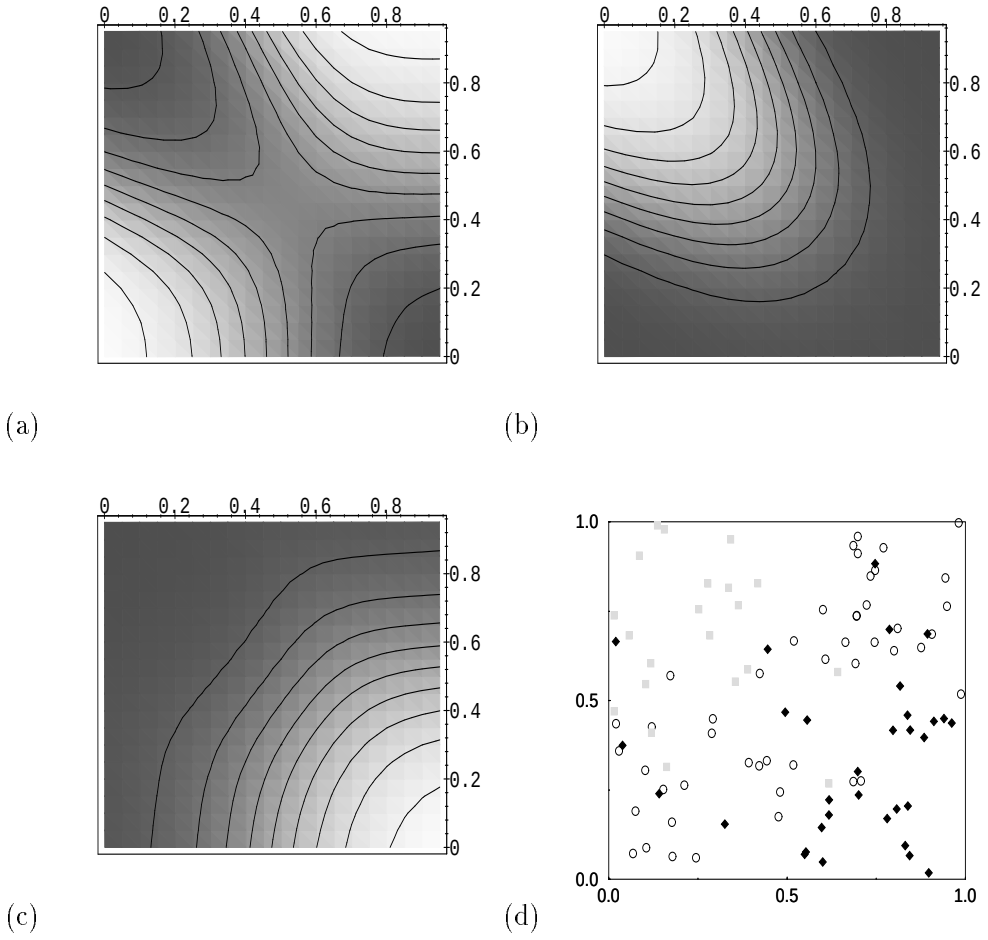


Figure 5.4: Training data for multi-class toy problem no.1 : This figure shows the probability distributions ((a)-(c) for classes 1-3 respectively) which were used to generate the training data. Light regions of the plots denote areas of high probability of belonging to a given class. The 100 training examples are shown in (d). Class one is shown as open circles, class two as grey squares and class three as black diamonds.

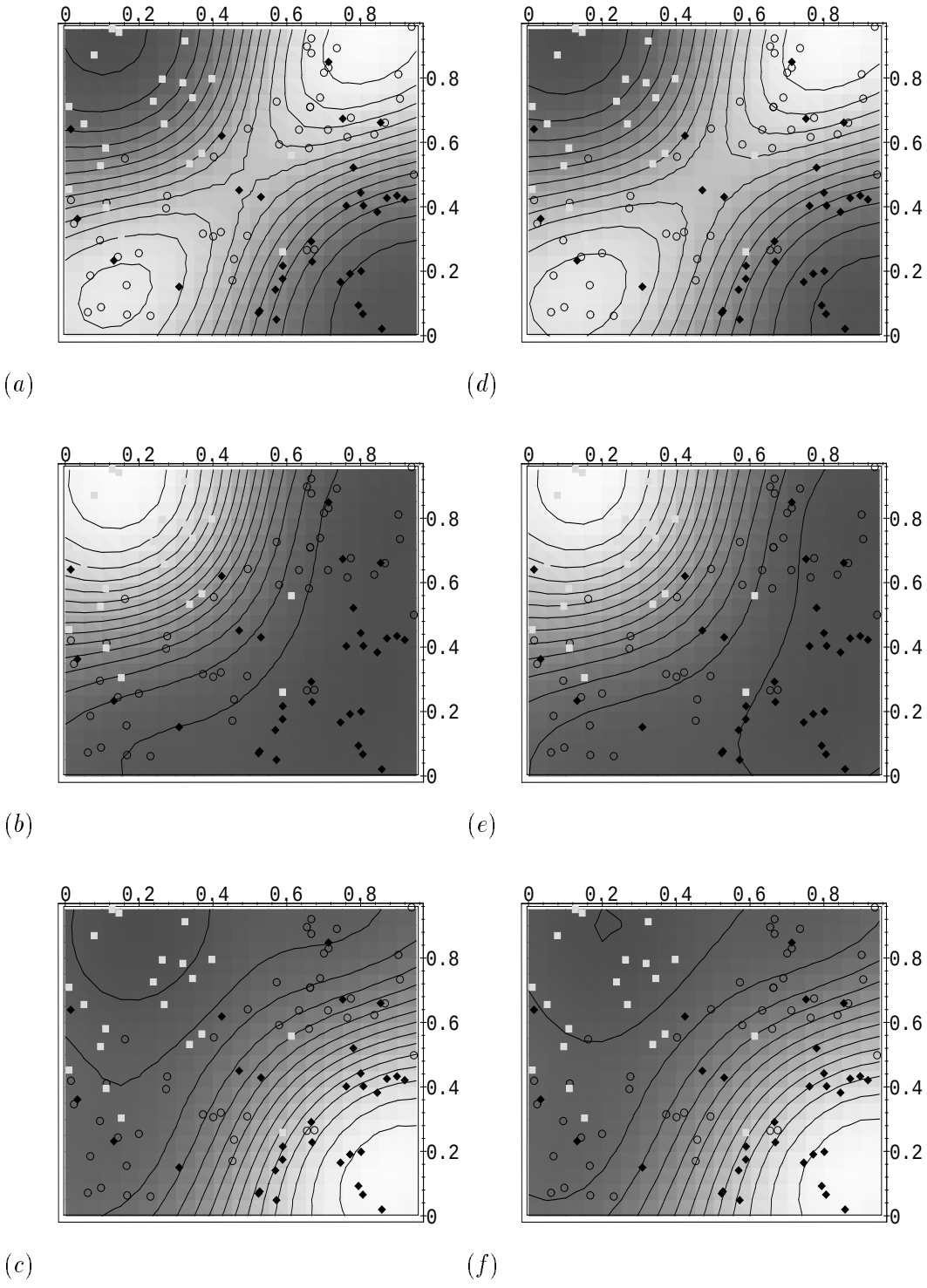


Figure 5.5: Results for multi-class toy problem no.1 : (a)-(c) show the predictive probability distributions for classes 1-3 respectively obtained using the lower bound of the MGC with the training data superimposed. Class one is shown as open circles, class two as grey squares and class three as black diamonds. The corresponding plots for the upper bound are shown in (d)-(f). The predictive probabilities were generated using the extended MacKay formula.

Input	Class 1	Class 2	Class 3
$x^{(1)}$	0.460	0.747	0.366
$x^{(2)}$	0.290	0.445	0.711
$x^{(3)}$	1.198	1.147	1.596
$x^{(4)}$	1.331	1.129	1.732

Table 5.2: Length scales for multi-class toy problem no.2 : The figure shows the most probable length scales found training the MGC on 400 samples from toy problem no.2. Notice how the inputs which have no bearing on the class ($x^{(3)}$ and $x^{(4)}$) have larger length scales than the other inputs.

how, as we would expect, the inputs which had no bearing on the class ($x^{(3)}$ and $x^{(4)}$) have larger length scales than the other inputs.

5.7 Conclusions

Multi-class classification using variational methods is a more complex problem than binary classification. This is due to the difficulty involved in finding good variational bounds for multi-dimensional functions. However even using the relatively simple bounds presented in this chapter, the results are promising. Performance similar to that of Neal (1997)'s Monte Carlo Gaussian process classifier was obtained on the three class problem discussed above although the training times for the MGC were longer. A comparison on more data sets (perhaps using DELVE (Rasmussen 1996)) would clarify the position of the MGC approach in the rankings of classification methods.

Despite the initially encouraging progress, there are several problems that still need to be tackled. The validity of the lower bound is an important issue and an analytic proof is required. The parameterization of the lower bound might also be called into question. It may be possible to express the properties of the bound in a more intuitive manner which will aid with optimization and hence decrease run times. Another possibility for improving the lower bound is to introduce some correlations between the $\{a^{(i)}\}$. This might allow us to capture the characteristics of the data more accurately although it is hard to see how such correlations could be introduced without substantially increasing the computational cost of the method.

Chapter 6

Theoretical Densities of States

The use of *ab initio* quantum mechanics techniques to solve problems in physics and chemistry is now becoming commonplace. Advances in techniques as well as a steady increase in the power of computers has meant that problems previously thought to be intractable or, at least, infeasibly computationally expensive have been tackled successfully.

Within solid state physics, there has been much use of such *ab initio* techniques to investigate the properties of crystals (Payne *et al.* 1992). One area that has provoked interest has been that of calculating core-edge Electron Energy Loss Spectra (EELS). EELS are experimentally generated using Scanning Transmission Electron Microscopes fitted with spectrometers. The ability to focus the electron probe of the STEM allows us to obtain EELS with atomic resolution in a strongly localized area. Thus EELS are ideal for studying the local micro-structure and electronic structure of crystals, particularly at grain boundaries or edges.

Theoretical calculations of EELS are desirable for several reasons. Firstly, it is challenging to generate high quality experimental EELS. Most experimental EELS are dogged by high noise levels, especially as energies increase. Theoretical EELS avoid such noise and give greater resolution (although other sources of error are introduced into the calculations). Secondly, a theoretical framework allows us the freedom to "play" with the system we are investigating, giving us the ability to build up an intuition about the underlying physics of the problem. Thirdly, experimental EELS require a significant amount of specialist equipment and a large amount of time and effort. Theoretical calculations only require a certain (often quite large) amount of computing resources. These computer resources are not confined to EELS calculations but can be used for a wide variety of *ab initio* problems, introducing an economy of scale for the theoretical approach.

In order to calculate an EELS, we need to find the density of states of the crystal structure of interest. In this section, we shall present a variety of methods used to generate theoretical densities of states. Firstly we shall discuss the underlying physics behind EEL spectra. Then we shall consider the use of *ab initio* quantum mechanical techniques to transform the calculation of density of states required to obtain an EEL spectrum into a combinatorial allocation and interpolation problem. We shall then present three possible methods for solving the problem. The first is based on a mixture model approach using free-energy minimization techniques. The second also uses free-energy minimization but is based on Gaussian processes and the third is based on the theory of random matrices. We shall compare the results of these methods with those of Pickard's quadratic expansion method (Pickard 1997).

6.1 The Physics of Electron Energy Loss Spectra

In a Scanning Transmission Electron Microscope, electrons are focused in a very narrow beam onto a specimen. The electrons pass through the specimen and are collected by a spectrometer which

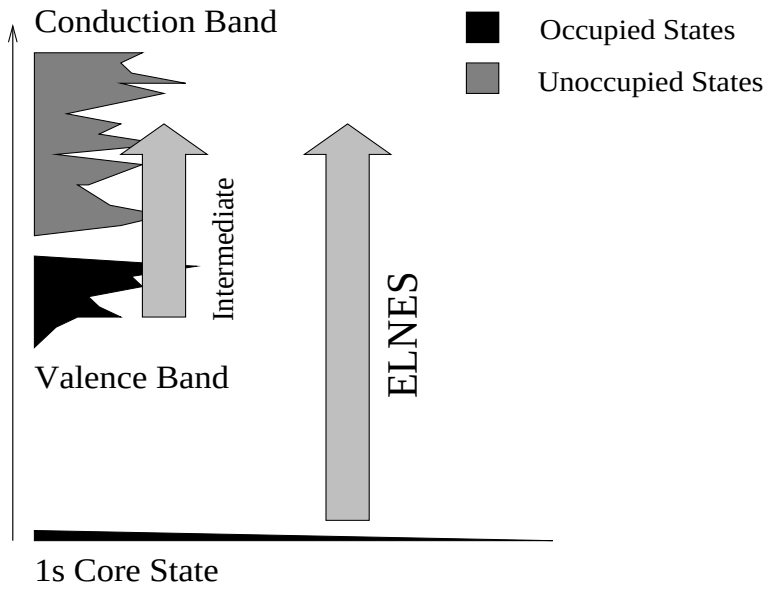


Figure 6.1: EELS scattering : This diagram shows two of the three main mechanisms for inelastic scattering in STEMs. Firstly, the intermediate energy losses are caused by complicated excitations of electrons in the valence band and by excitation of plasmons. The resulting spectrum is a weighted convolution of the valence and conduction density of states. Secondly, the high energy losses result from excitation of core electrons which, as the core states have very well defined energies, give us a detailed picture of the density of states of the conduction band.

measures their energies.

The electrons can lose energy, i.e., scatter off the specimen in three main ways (see Figure 6.1). Some electrons excite phonons within the specimen resulting in energy losses of the order of 10-100meV. Such losses are below the resolution of most spectrometers and hence contribute to the zero-loss peak in which the electrons appear to be scattering elastically. Electrons with slightly higher energies primarily lose energy (typically 1 to 50eV) due to a complicated mixture of single electron transitions from the valence band of the specimen to the conduction band. The rate of such transitions is proportional to the density of electrons in the valence band and the density of empty states in the conduction band. Thus the section of the EEL spectrum corresponding to these excitations is a convolution of the valence and conduction density of states (slightly modified by certain quantum mechanical matrix elements). Finally, for high energies, we obtain the largest losses (>100eV) caused by the excitation of the core states of the specimen giving us a detailed picture of the conduction band (again weighted with matrix elements).

It is the losses caused by the excitation of the core states or "Energy Loss Near core Edge Structure" (ELNES) that we are primarily interested in. We can write down the equation for the differential cross-section of the scattering process;

$$\frac{d^2\sigma}{d\Omega dE} = \frac{1}{(\pi e a_0)^2}\frac{1}{q^2}\mathrm{Im}\left\{\frac{-1}{\epsilon_M(\mathbf{q},\omega)}\right\} \tag{6.1}$$

where $q = |\mathbf{q}|$ is the momentum transfer, $\hbar\omega$ is the energy lost in the scattering and ϵ_M is the macroscopic dielectric function. At high energies, the real part of ϵ_M tends to unity and the imaginary part tends to zero. Hence

$$\mathrm{Im}\left\{\frac{-1}{\epsilon_M(\mathbf{q},\omega)}\right\} \simeq \mathrm{Im}\{\epsilon_M\} \tag{6.2}$$

$$\simeq \frac{4\pi e^2}{q^2 V}\sum_f \left|\langle f|e^{-i\mathbf{q}.\mathbf{r}}|i\rangle\right|^2 \delta\left(\hbar\omega - E_f + E_i\right). \tag{6.3}$$

$|i\rangle$ is the initial state of the electron involved in the excitation (a core state for the region of the EELS that we are concerned with) and $|f\rangle$ is the final state. The summation is the density of states (DOS) of the specimen weighted by the matrix elements $\langle f|e^{-i\mathbf{q}.\mathbf{r}}|i\rangle$. Thus, in order to calculate the differential cross-section and hence the EELS, we need to find the specimen's electronic states, $|i\rangle$, and energy levels, E_i. We can then find the density of states and hence calculate the EELS. It is the calculation of the density of states that is the problem that we shall tackle.

6.2 Density of States and *ab initio* quantum mechanics

Quantum mechanics, while not fully describing all physical phenomena, has been hugely successful in describing many of the systems that modern scientists investigate, from low energy physics to chemistry and biology. In the field of solid state physics, we use the quantum mechanical description of the behaviour of electrons and ions to calculate many properties of crystals. We shall focus on calculations of the total energy of a crystal, E, defined by

$$H\Psi = E\Psi \tag{6.4}$$

where H is the Hamiltonian of the crystal and Ψ is the many-body wavefunction. Given the total energy of the crystal, we can, by taking derivatives, calculate many of its properties including the lattice constant, the density and the bonding structure.

Unfortunately, finding the total energy is not easy. Let us consider a typical form of the Hamiltonian:

$$H = \sum_{n=1}^{N}\left(\frac{-\hbar^2}{2m}\nabla_n^2 - Ze^2\sum_{\mathbf{R}}\frac{1}{|\mathbf{r}_n - \mathbf{R}|}\right) + \frac{1}{2}\sum_{n\neq m}\frac{e^2}{|\mathbf{r}_n - \mathbf{r}_m|} \tag{6.5}$$

The first term is the kinetic energy of the electrons. We treat the heavier ions classically and so consider them to be fix when performing the quantum mechanical calculations for the faster moving electrons. The second term is the electron-ion interaction and the third is the electron-electron interaction. It is the third term that creates the difficulties. Its physical origin is well known but it is very difficult to account for.

We can overcome the problems associated with the electron-electron interaction by using the fact that the total energy is a unique function of the electron density. This transforms the problem of calculating the total energy into one of solving a modified version of equation 6.4. The solutions to this modified Schrödinger equation are called the Kohn-Sham eigenstates and eigenvalues (Kohn and Sham 1965). Each eigenstate corresponds to a single electron wavefunction of the system with a specific $\mathbf{k}$-vector, the eigenvalues being the corresponding energies. Solutions of the modified Schrödinger equation can be found in a variety of ways but we choose to use pseudo-potential approximations and a plane wave basis set (implemented in the program CASTEP). Details of the methods and approximations mentioned above are beyond the scope of this thesis. See Payne *et al.* (1992) for a comprehensive review.

Ordinarily we would use the Kohn-Sham eigenstates to write down an approximation to the total energy and use its derivatives to calculate physical properties of the system. However in

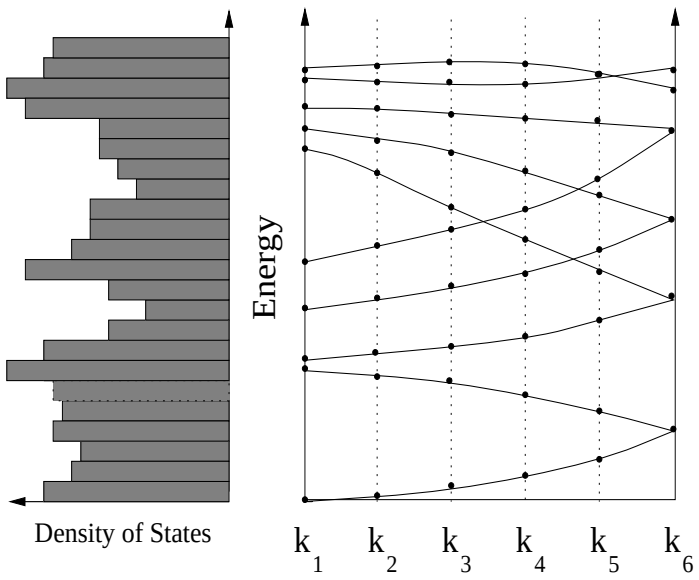


Figure 6.2: Band Structure and the Density of States : This diagram shows a one dimensional slice through **k**-space. We can see the smoothly varying bands corresponding to the Kohn-Sham eigenvalues for different **k** vectors. The vertical dotted lines show points in **k**-space at which the Kohn-Sham eigenvalues have been calculated. Many eigenstates exist with a specific **k**-vector and hence we obtain a stack of energies at each **k**-point. A crude density of states can be generated using the energies at each of the **k**-points. The resolution of the histogram is deliberately low so that errors introduced by band crossing are not pronounced.

order to generate the density of states that we need to calculate the EELS, we shall consider only the single electron wavefunctions and their energies. We could, in theory, calculate the energies of wavefunctions for a huge number of different **k**-vectors (or **k**-points). The energies form into apparently smooth bands with respect to variations in **k** and so, given enough **k**-points, we could create a histogram approximation of the density of states by projecting the energies onto the energy axis (see Figure 6.2). However the data is very expensive to generate as the plane wave basis set used by CASTEP is very large. Thus the number of **k**-points used needs to be kept to a minimum.

One solution would therefore be to have a few **k**-points and interpolate between them in order to approximate the band structure. The tetrahedron method of Blochl *et al.* (1994) is commonly used. For smooth bands that do not cross, this approach works well. However in most systems, the bands have sharp cusps where they appear to cross (see Figure 6.3). In this case simple interpolation with a small number of **k**-points can go disastrously wrong, creating band gaps in the density of states where no band gaps actually exist.

Rather than perform a simple interpolation, we can take a slightly more sophisticated approach. At first glance, the bands are smooth and appear to cross each other. While this is not physically correct, let us assume it is true. We can therefore attempt to find a set of smooth functions which pass through the data generated by our *ab initio* code, one and only one function passing through each data point. We can now re-express the problem in terms of the following generative model: We have a set of K functions $\{y_k(\mathbf{k})\}(k = 1 \cdots K)$ and a set of N **k**-points $\{\mathbf{k}_n\}(n = 1 \cdots N)$. At each **k**-point, noisy versions of each function are generated.

$$t_n^k = y_k(x_n) + \nu \tag{6.6}$$

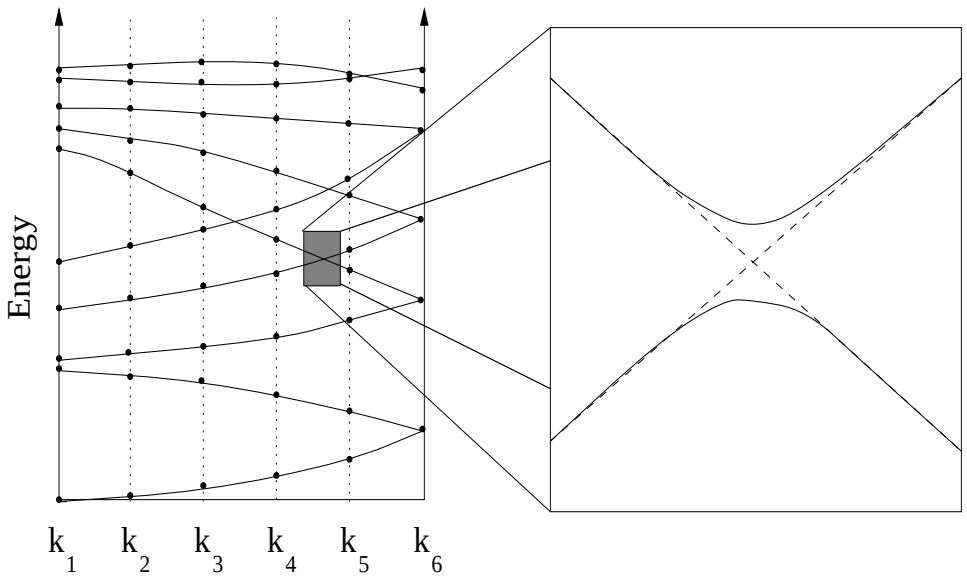


Figure 6.3: Band Crossings : At first glance, it appears that the bands shown in the figure do indeed cross. At higher resolution we can see that the bands "repel" each other as they come close together and have sharp cusps.

where ν is Gaussian noise. The data obtained from the *ab initio* code is almost noise-free and hence ν is expected to be very small. The data we receive for each **k**-point is the set $\{t_n^k\}$ with the labels k removed. Let us call this set $\{\tilde{t}_n^i\}$ where the label n $(n = 1 \cdots N)$ refers to the n^{th} **k**-point and the label i $(i = 1 \cdots K)$ refers to the i^{th} data point in the stack at **k**-point $\mathbf{k}_n$.

The problem is to infer the functions $\{y_k(\mathbf{k})\}$ given some data $\mathcal{D} = \{\mathbf{k}_n, \{\tilde{t}_n^i\}\}(n = 1 \cdots N, i = 1 \cdots K)$ and some prior information on the form of the functions. This is a standard problem except for one complication. We know that at every **k**-point, each function only produces one data point. Hence we have a permutation matrix $\mathbf{R}_n$ at each point which defines the assignment of data points $\{\tilde{t}_n^i\}$ to unknown functions $\{y_k(\mathbf{k})\}$.

6.3 The Mixture Model Approach

Let us now consider a possible approach to the allocation and interpolation problem based on mixture models (McLachlan and Basford 1988). Temporarily, we shall ignore the constraints placed on the problem by the permutation matrices $\{\mathbf{R}_n\}$. Instead we shall assume that we have K smooth functions which have generated the $\{\tilde{t}_n^i\}$ at each **k**-point and that any one function might have produced none, one or more than one data point at any given **k**-point.

We define K neural networks, $o_k(\mathbf{k}; \mathbf{w}_k)$, to model the K functions $\{y_k(\mathbf{k})\}$. Each neural network is a fully connected MLP with one hidden layer and a weight vector $\mathbf{w}_k$. Following a Bayesian approach, we wish to find the posterior distribution $P(\mathbf{W}|\mathcal{D})$ where $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_K)$. We can then use the posterior distribution to infer the weights of the networks given the data. Using Bayes' rule,

$$P(\mathbf{W}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathbf{W})P(\mathbf{W})}{P(\mathcal{D})}. \tag{6.7}$$

$P(\mathbf{W})$ is the prior on the network weights which we will later use to express our belief that the bands are smooth (see Section 6.3.2). We assume that the likelihood, $P(\mathcal{D}|\mathbf{W})$, is a separable distribution:

$$P(\mathcal{D}|\mathbf{W}) = \prod_{i=1}^{K} \prod_{n=1}^{N} P(\tilde{t}_n^i|\mathbf{w}_i) \tag{6.8}$$

We now introduce the energy function E_n^{ik} which defines the probability of a data point, $\tilde{t}_n^i$, given that it was generated by the function $y_k(\mathbf{k})$ at $\mathbf{k}_n$:

$$\begin{aligned} E_n^{ik} &= -\log P(\tilde{t}_n^i|\tilde{t}_n^i \in k, \mathbf{w}_k) + \text{const.} & (6.9)\\ &= \frac{(\tilde{t}_n^i - o_k(\mathbf{k}_n;\mathbf{w}_k))^2}{2\sigma_E^2} & (6.10)\end{aligned}$$

where $o_k(\mathbf{k}_n;\mathbf{w}_k)$ is the output of the k^{th} network at $\mathbf{k}_n$ and σ_E^2 is the noise variance.

Using a uniform prior on which data points belong to which function,

$$P(\tilde{t}_n^i|\mathbf{W}) = \sum_{k=1}^{K} \frac{1}{Z} \exp\left(-E_n^{ik}\right) \tag{6.11}$$

where Z is an appropriate normalizing constant. Differentiating the log probability of $\tilde{t}_n^i$ given the weights, we obtain

$$-\frac{\partial \log P(\tilde{t}_n^i|\mathbf{W})}{\partial \mathbf{w}_j} = \underbrace{\frac{1}{\sum_k \exp\left(-E_n^{ik}\right)} \exp\left(-E_n^{ij}\right)}_{P(\tilde{t}_n^i \epsilon j|\tilde{t}_n^i,\mathbf{W})} \frac{\partial E_n^{ij}}{\partial \mathbf{w}_j} \tag{6.12}$$

where $P(\tilde{t}_n^i \epsilon j|\tilde{t}_n^i,\mathbf{W})$ is the probability that the data point $\tilde{t}_n^i$ was generated by the function modelled by the j^{th} neural network. Equation 6.12 is telling us that the gradient of the log probability of the data point $\tilde{t}_n^i$ with respect to a given network's weights is proportional to the gradient of the energy function E_n^{ik} for the k^{th} neural network multiplied by a weighting factor that tell us how likely it is that the data point "belongs" to that network.

6.3.1 Free Energy Method

The above derivation is for an unconstrained mixture model. Let us consider how we might incorporate into the mixture model framework the constraints placed on us by the nature of the problem. We see that we can no longer write

$$P(\tilde{t}_n^i \in j|\tilde{t}_n^i,\mathbf{W}) = \frac{1}{\sum_k \exp\left(-E_n^{ik}\right)} \exp\left(-E_n^{ij}\right) \tag{6.13}$$

as we know that every function $y_n(\mathbf{k})$ produces a data point at every $\mathbf{k}$-point, i.e., for the permutation matrix $\mathbf{R}_n$,

$$P(\tilde{t}_n^i \in j|\tilde{t}_n^i,\mathbf{W}) = \begin{cases} 1 & \text{if } (\mathbf{R}_n)_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

If we wish to rigidly enforce these constraints we can write the probability of the data $\tilde{\mathbf{t}}_n = (\tilde{t}_n^{(1)}, \tilde{t}_n^{(2)}, \cdots, \tilde{t}_n^{(K)})$ at $\mathbf{k}_n$ as

$$P(\tilde{\mathbf{t}}_n|\mathbf{W}) = \sum_{\mathbf{R}_n} P(\tilde{\mathbf{t}}_n|\mathbf{R}_n, \mathbf{W}) P(\mathbf{R}_n) \tag{6.15}$$

and hence derive an expression for the derivative of the log probability of all the data with respect to the weights. However, the summation in the above expression is over all possible permutation matrices. The number of possible permutation matrices scales as the factorial of the number of functions, K, making this summation infeasible if the number of functions is large. Hence such an approach is not viable. However, if we could find some method to calculate or approximate $P(\tilde{t}_n^i \in j|\tilde{t}_n^i, \mathbf{W})$ given the constraints that did not involve a summation over all possible permutation matrices then we could solve the problem using the framework of the standard mixture model.

With this goal in mind, let us consider a variational free energy minimization method. Hinton and Zemel (1994) introduced the concept that, when faced with a badly behaved or awkward posterior distribution $P(\mathbf{w}|\mathcal{D})$ over some vector of parameters $\mathbf{w}$, we should work in terms of an approximating *ensemble*, $Q(\mathbf{w};\Xi)$, which is easier to deal with. We then optimize the parameters, Ξ, of the ensemble so that $Q(\mathbf{w};\Xi)$ approximates the distribution $P(\mathbf{w}|\mathcal{D})$ as closely as possible. We use the variational free energy, a well-established tool in statistical physics, as the measure of quality of the approximation.

$$F(\Xi) = -\int d\mathbf{w}\, Q(\mathbf{w};\Xi) \log \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{Q(\mathbf{w};\Xi)} \tag{6.16}$$

The free energy $F(\Xi)$ is the sum of the Kullback-Leibler divergence between $Q(\mathbf{w};\Xi)$ and $P(\mathbf{w}|\mathcal{D})$ and $-\log P(\mathcal{D})$. It is bounded below by $-\log P(\mathcal{D})$ and only attains this value for $Q(\mathbf{w};\Xi) = P(\mathbf{w}|\mathcal{D})$.

For the constrained mixture model, we wish to find approximations to $P(\tilde{t}_n^i \in j|\tilde{t}_n^i, \mathbf{W})$ given the constraints so that we can approximate the gradient expression given in equation 6.12. Let us define a separable approximating distribution for $P(\mathbf{A}_n|\tilde{\mathbf{t}}_n, \mathbf{W})$:

$$Q(\mathbf{A}_n;\Theta) = \prod_{k=1}^{K} q_n^{a_n^k k} \simeq P(\mathbf{A}_n|\tilde{\mathbf{t}}_n, \mathbf{W}) \tag{6.17}$$

where a_n^k is the label of the data point at $\mathbf{k}_n$ which is assigned to neural network $o_k(\mathbf{k};\mathbf{w}_k)$ by the matrix $\mathbf{A}_n$. The elements of $\mathbf{A}_n$ are given by $A_{ij} = \delta_{ia_n^i}$. Note that $\mathbf{A}_n$ is not necessarily a permutation matrix. We have deliberately relaxed the constraints imposed by the permutation matrices in order to make the problem tractable.

The individual factors of $Q(\mathbf{A}_n;\Xi)$, q_n^{jk}, are approximations to $P(\tilde{t}_n^j \in k|\tilde{t}_n^j, \mathbf{W})$ and hence must satisfy the constraints:

$$\sum_{j=1}^{K} q_n^{jk} = 1 \;\forall\, k \tag{6.18}$$

$$\sum_{k=1}^{K} q_n^{jk} = 1 \;\forall\, j. \tag{6.19}$$

Assuming that we have found a form for the q's that obeys equation 6.18 and 6.19, we can write down an expression for the free energy at each $\mathbf{k}$-point,

$$F_n(\Theta) = -\sum_{\mathbf{A}_n} Q(\mathbf{A}_n;\Xi) \log \frac{P(\tilde{\mathbf{t}}_n|\mathbf{A}_n, \mathbf{W})P(\mathbf{A}_n)}{Q(\mathbf{A}_n;\Xi)} \tag{6.20}$$

We require that $Q(\mathbf{A}_n;\Xi) = 0$ if $\mathbf{A}_n$ is not a permutation matrix but $Q(\mathbf{A}_n;\Xi)$ is not necessarily zero in this case. However the normalization conditions imposed on the q's combined with the nature of the problem that we are trying to solve mean that, as $Q(\mathbf{A}_n;\Xi) \to P(\mathbf{A}_n|\tilde{\mathbf{t}}_n, \mathbf{W})$, non-permutation matrices become increasingly improbable. For example, if two of the networks have a high probability of having generated one data point then the normalization conditions will favour the network with the highest probability, correspondingly reducing the probability of the other network. As the weights of the networks are adapted and their outputs draw closer to the data points, the probability of non-permutation matrices falls away to zero. Thus the relaxation of the permutation matrix constraint which allows us to perform a summation over all possible "allocation" matrices, $\mathbf{A}_n$, is balanced by enforcing the normalization conditions in equations 6.18 and 6.19.

We assume that the prior on the permutation matrices is uniform. The distribution $P(\tilde{\mathbf{t}}_n|\mathbf{A}_n, \mathbf{W})$ is defined by equation 6.9,

$$P(\tilde{\mathbf{t}}_n|\mathbf{A}_n, \mathbf{W}) \propto \prod_{k=1}^{K} \exp\left(-E_n^{a_n^k k}\right), \tag{6.21}$$

and so

$$\begin{aligned} F_n(\Xi) &= \sum_{\mathbf{A}_n} \left\{ Q(\mathbf{A}_n;\Xi) \sum_{k=1}^{K} E_n^{a_n^k k} + Q(\mathbf{A}_n;\Xi) \log Q(\mathbf{A}_n;\Xi) \right\} & (6.22) \\ &= \sum_{\mathbf{A}_n} \left\{ \prod_k q_n^{a_n^k k} \sum_{k=1}^{K} E_n^{a_n^k k} + \prod_k q_n^{a_n^k k} \sum_k \log q_n^{a_n^k k} \right\} & (6.23) \end{aligned}$$

We know that $F_n(\Xi)$ achieves its minimum when $Q(\mathbf{A}_n;\Xi) = P(\mathbf{A}_n|\tilde{\mathbf{t}}_n, \mathbf{W})$. Hence, using equation 6.23, we can write down an expression for q_n^{jk} at the minimum of $F(\Xi)$:

$$q_n^{jk} = \exp\left(-E_n^{jk} + \xi_n^{jk}\right) \tag{6.24}$$

where $\Xi = \{\xi_n^{jk}\}$ are such that the q's obey the normalization constraints in equations 6.18 and 6.19. The $\{\xi_n^{jk}\}$ can be set by a process of repeated normalization, i.e., if we alternately normalize the rows and columns of the matrix $\mathbf{Q}^{(n)}$ where $Q_{jk}^{(n)} = q_n^{jk}$ then $\mathbf{Q}^{(n)}$ will almost always converge to a matrix with simultaneously normalized rows and columns. We shall refer to this as the `NORM` algorithm (see Figure 6.4).

There are a few cases in which simultaneous normalization will not occur (or will take many iterations to occur). This is symptomatic either of a particular data point being very unlikely to belong to any of the functions, $o_n(\mathbf{k};\mathbf{w}_k)$, or of a particular function, $o_n(\mathbf{k};\mathbf{w}_k)$, being very unlikely to pass through any of the data points at a given $\mathbf{k}$-point. Only zeros in the $Q^{(n)}$ matrix will cause a complete failure. Slow convergence can be avoided by a careful choice of the parameter of the energy function E_n^{jk}, σ_E (see Section 6.3.2), and by performing the normalization calculations with respect to ξ_n^{jk} rather than q_n^{jk}.

We have now found approximations, q_n^{jk}, to the probabilities $P(\tilde{t}_n^j \in k|\tilde{t}_n^j, \mathbf{W})$ which, having been subjected to the `NORM` algorithm, obey a relaxed version of the permutation matrix constraints. These can now be substituted into the mixture model framework.

$$-\frac{\partial \log P(\tilde{t}_n^j|\mathbf{W})}{\partial \mathbf{w}_k} \simeq q_n^{jk} \frac{\partial E_n^{jk}}{\partial \mathbf{w}_k} \tag{6.25}$$

Define tolerance $\epsilon(\ll 1)$

do {

for$(i = 1$ to $K)\{$
$Z_i = \sum_k q_n^{ik}$
for$(k = 1$ to $K)\{$
$\xi_n^{ik} = \xi_n^{ik} - \log Z_i$
$\}$
$\}$

for$(k = 1$ to $K)\{$
$Z_k = \sum_i q_n^{ik}$
for$(i = 1$ to $K)\{$
$\xi_n^{ik} = \xi_n^{ik} - \log Z_k$
$\}$
$\}$

} while { $\left|\sum_i q_n^{ik} - 1.0\right| > \epsilon$ for any k or $\left|\sum_k q_n^{ik} - 1.0\right| > \epsilon$ for any i }

Figure 6.4: `NORM` algorithm : This algorithm performs alternate normalization of the rows and columns of the matrix $\mathbf{Q}^{(n)}$ where $Q_{ik}^{(n)} = q_n^{ik}$. Once all the row and column sums are within ϵ of unity, the algorithm will terminate.

6.3.2 Priors and Hyperparameters

The success of the mixture model approach relies on the distinguishability of the bands based on a prior belief that the bands are smooth. The smoothness of the functions produced by the neural networks is dependent on the priors placed on the weights of the networks. In the Bayesian approach that was used, the weights were separated into four classes: input to hidden neuron weights (class 1), bias to hidden neuron (class 2), hidden neuron to output (class 3) and bias to output (class 4). Each class had a separable Gaussian prior,

$$P(\mathbf{w}^{(c)}) = \frac{1}{Z_c} \exp\left(-\frac{1}{2}\sum_i \left(\frac{w_i^{(c)}}{\sigma_w^{(c)}}\right)^2\right) \tag{6.26}$$

where c denotes a different class of weight. $\sigma_w^{(3)}$ and $\sigma_w^{(4)}$ affect the vertical offset and scaling of the functions and $\sigma_w^{(2)}$ controls the total number of fluctuations in the functions across the input space. It is $\sigma_w^{(1)}$ which determines the smoothness of the functions ($1/\sqrt{\sigma_w^{(1)}}$ can be treated as the horizontal length scale of the band). As we wish to allow only smooth bands, we set $\sigma_w^{(1)}$ to a small value so that the corresponding length scale is equal to approximately half the average width of the Brillouin zone ($\mathbf{k}$-space).

The other hyperparameter that significantly affects the results obtained using the mixture model approach is σ_E (see equation 6.10). σ_E is a noise term describing how far the output of a neural network, $o_k(\mathbf{k};\mathbf{w}_k)$, can deviate from a data point at $\mathbf{k}_n$ that we believe to have been generated by the function the network is modelling. It could also be viewed as a "distance" parameter: each function can only "see" a data point that is a distance $\mathcal{O}(2\sigma_E)$ from its present position along the

energy axis. The influence of that data point on the function is then dependent on the distance of the point from the function relative to σ_E and also the probability of the data point belonging to any of the other functions.

The setting of $\sigma_w^{(1)}$ and σ_E is crucial to obtaining a plausible solution in a manageable time. For example, imagine the situation where we are initializing the weights of the neural networks. If we choose the initial weights in a quasi-random fashion then it is possible that all of the initial neural networks $\{o_k(\mathbf{k};\mathbf{w}_k^{ini})\}$ will not come within a distance δ of one of the data points, $\tilde{t}_n^i$. If σ_E is set to small value ($\sigma_E \ll \delta$) then the mixture model will almost completely ignore the contribution of this point. The `NORM` algorithm will encounter convergence problems and the mixture model will take a long time to find a plausible solution. It might therefore be tempting to set σ_E to a large value. However we would then obtain a solution in which the outputs of the neural networks would not pass through the training data points. We know that the training data is virtually noise-free and so such a solution would be unacceptable. The solution we adopt to the problem of setting σ_E is to set it to an initially high value and then decrease it during training. This adds an element of simulated annealing to the free energy minimization process.

6.3.3 Matrix Elements and Crystal Symmetry

Initial experiments performed using the mixture model approach showed that the smooth prior on the bands was not enough to uniquely determine which data point should belong to which band. Fortunately there are two other sources of information that we can use to help us.

As well as being able to calculate a stack of energies $\tilde{\mathbf{t}}_n = (\tilde{t}_n^{(1)}, \cdots, \tilde{t}_n^K)$ at each $\mathbf{k}$-point, we can also calculate certain matrix elements, $S_c(\mathbf{k})$, associated with each of the energies:

$$S_c(\mathbf{k}) = \langle \psi_m | \hat{\mathbf{p}} | \psi_n \rangle \tag{6.27}$$

where ψ_m and ψ_n are single particle wavefunctions and $\hat{\mathbf{p}}$ is the momentum operator. The matrix elements are also smooth functions of position in $\mathbf{k}$-space along a given band and we can use these elements to help us with our interpolation. When faced with an ambiguous crossing in terms of energy, we can examine the matrix elements on each side of the crossing to determine the correct allocation. The elements can be incorporated into the mixture model approach by expanding the number of outputs of the neural network provided that the matrix elements are scaled such that the noise parameter σ_E is appropriate for both the energy and the matrix elements.

The smoothness of the bands is often not the only prior information we have about them. Frequently we will also know the crystallographic point group of the crystal in question (hexagonal, cubic, orthorhombic etc.) and hence the symmetries that the bands (both energy and matrix elements) must possess. These symmetries can be described in terms of 3×3 reflection or rotation matrices, $\mathbf{M}_s$, under which the bands are invariant. It is possible to build symmetries into neural networks using a process called weight-tying. We expand the number of inputs from three to $3S$ where S is the number of symmetry matrices. Each group of three inputs has a different matrix $\mathbf{M}_s$ applied to it but the weights joining each group to the hidden neurons are the same. The value of a hidden neuron at $\mathbf{k}_n$ becomes

$$h_i(\mathbf{k}_n) = \phi\left[\sum_{s=0}^{S}\sum_{j=1}^{J} w_{ij}^{(1)} \left(\sum_{k=1}^{3} \mathbf{M}_s \mathbf{k}_n\right)_j\right] \tag{6.28}$$

where $\phi()$ is the activation function, $w_{ij}^{(1)}$ are the input-to-hidden weights and $\mathbf{M}_0$ is always the identity. This guarantees that any output of the neural network has the symmetries described by $\{\mathbf{M}_s\}$.

6.3.4 Performance of the Mixture Model Approach

The first crystal studied using the mixture model approach was Silicon with four atoms per unit cell. Data was generated at 48 **k**-points for the bottom 20 bands. $\sigma_w^{(1)}$ was set so that the corresponding length scale was 0.3 (the width of the Brillouin zone being 0.5). 10 matrix elements and 48 symmetry matrices (the total number available for cubic Silicon) were used. Neural networks with one hidden layer containing 10 neurons were used to model each band. A conjugate gradient algorithm was used to optimize the weights. σ_E was initially set to 1.0 for the first 10 weight updates of the network and then decreased by a factor of 1.2 after every subsequent update until σ_E reached 0.1. A total of 100 weight updates were performed. The density of states was calculated by drawing 20000 random samples for the Brillouin zone and constructing a histogram of the most probable predicted energies of each band using 200 bins. A similar study was performed on hexagonal Graphite. 48 **k**-points, 9 matrix elements, 24 symmetry matrices and a length scale of 0.3 were used in conjunction with neural networks with one hidden layer containing 10 neurons. The same annealing schedule was used for σ_E. 100 weight updates using conjugate gradients were performed. The density of states was calculated using 200 bins and 20000 randomly generated samples from the Brillouin zone.

The results for both Silicon and Graphite obtained using the mixture model were disappointing (see Figure 6.5). The low energy densities of states are broadly similar to the results obtained by Pickard (1997) but the higher energy densities and the small scale features are substantially in error. Pickard's results are taken to be the best presently available and the method for obtaining them is described in Section 6.6.

The errors in the mixture model results could be due to the use of random sampling to calculate the density as the convergence is very slow, especially for sharp peaks. However a density of states calculation using 100,000 samples and 1000 bins produced very similar results. The most likely explanation for the errors is that not enough weight updates were performed. Examination of densities of states using weight vectors after 20, 40, 60 and 80 updates showed the convergence of the DOS to be slow. This was especially true for the higher bands where the bands are much closer together and contain many more "crossing" than for lower energies. The series of sharp spikes in the upper Graphite density of states correspond to these poorly determined higher bands. The total training time for 100 updates was about 60 hours on a SUN Sparc 10 for both Silicon and Graphite. This is unacceptably slow and rules out the possibility of more updates to increase accuracy.

Neural networks are not a convenient model for the bands. While the training times can be reduced by using a smaller number of hidden neurons, the quality of the density of states also declines. While the matrix elements improve the ability of the mixture model to decide on an allocation of data points to bands, each element only makes it possible to differentiate between a couple of bands. Hence $\mathcal{O}(K)$ matrix elements are needed to gain a noticeable improvement in performance. This dramatically increases the number of weights and results in the long training times described. So let us look for an alternative model for the bands which is not crippled by time consuming weight optimization.

6.4 The Gaussian Process Approach

As the band structure problem has a training data set that is limited in number of **k**-points, Gaussian processes present themselves as a promising tool to use. Gaussian processes also do not have any weights to optimize, can deal with almost noise-free data, do not require any optimization of architecture and can easily be constrained to produce smooth functions. Hence let us pursue an approach to the allocation problem based on Gaussian processes.

To begin with, we can define a set of data $\mathbf{u} = (\tilde{t}_1, \cdots, \tilde{t}_N)$ as coming from a Gaussian process

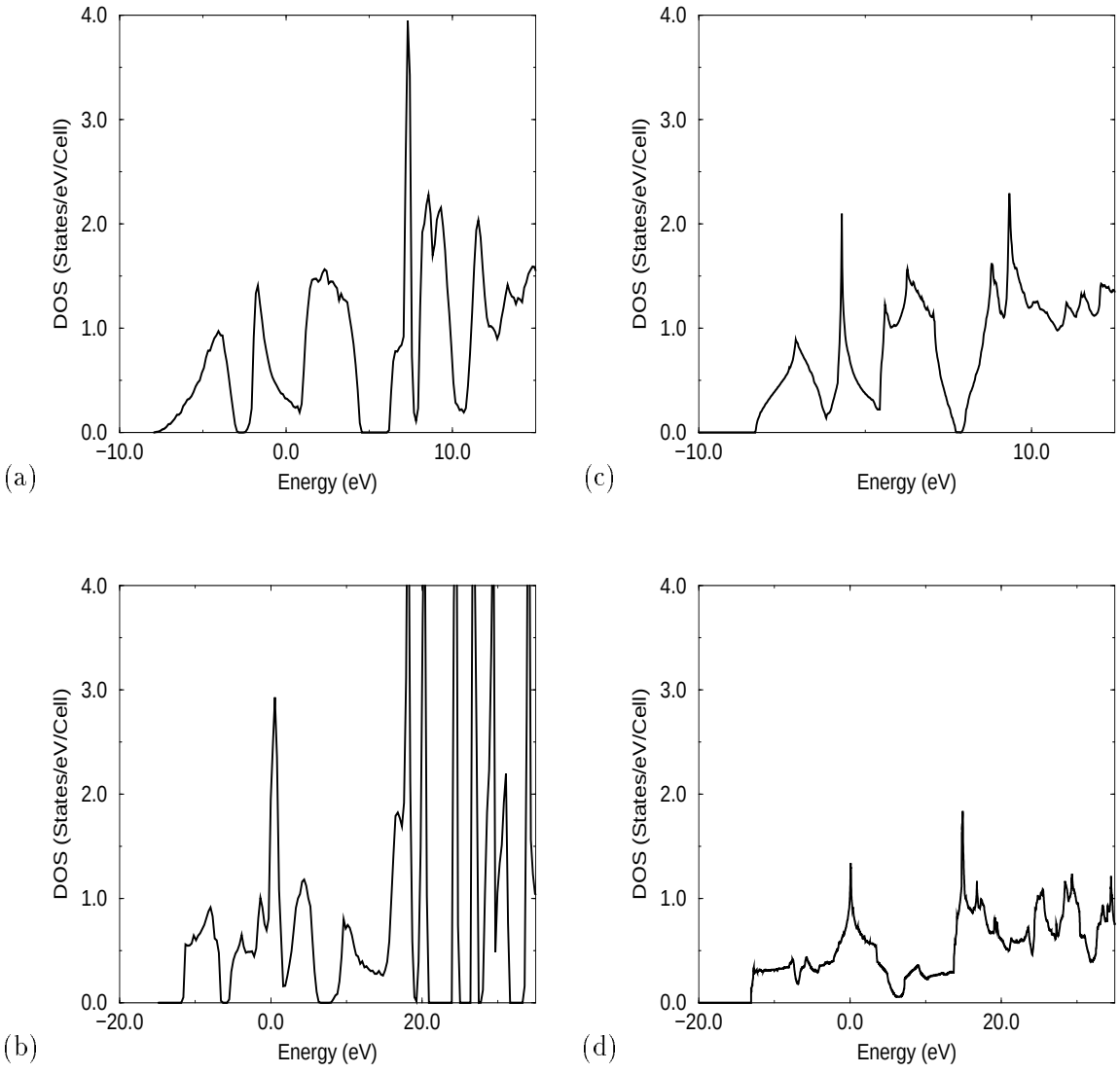


Figure 6.5: Mixture Model Approach Results : (a) and (b) show the densities of states obtained using the mixture model approach on 48 **k**-points of data for Silicon with four atoms per unit cell and hexagonal Graphite respectively. (c) and (d) show the corresponding results obtained using the method of Pickard (1997).

by writing

$$P(\mathbf{u}|\{\mathbf{k}_n\},\Theta) = \frac{1}{Z}\exp\left(-\frac{1}{2}\mathbf{u}^T\mathbf{C}_N^{-1}\mathbf{u}\right). \tag{6.29}$$

Given that the data comes from an almost noise-free smooth function, we can set the hyperparameters of the Gaussian process, Θ, accordingly. We could model every band using the same set of hyperparameters. However we expect that the mean of the Gaussian process modelling a high energy band is not the same as that modelling a low energy band. As, initially, we have no idea what the mean of each of the bands should be, we would like a covariance matrix which is insensitive to arbitrary vertical offsets of the data points associated with it, i.e.,

$$\mathbf{C}_N^{-1}(\mathbf{u}+\alpha\mathbf{n}) \equiv \mathbf{C}_N^{-1}\mathbf{u} \tag{6.30}$$

where $\mathbf{n}^T = (1,1,\cdots,1)$. Recalling the terminology of Section 2.2, this implies the vertical uncertainty, θ_2, should be infinite. Fortunately for $\theta_2 = \infty$, we can find a tractable expression for the inverse of the covariance matrix. Consider the identity

$$\left[\mathbf{C}_N + \theta_2\mathbf{n}\mathbf{n}^T\right]^{-1} = \mathbf{C}_N^{-1} - \frac{\theta_2\mathbf{C}_N^{-1}\mathbf{n}\mathbf{n}^T\mathbf{C}_N^{-1}}{1+\theta_2\mathbf{n}^T\mathbf{C}_N^{-1}\mathbf{n}} \tag{6.31}$$

As $\theta_2 \to \infty$,

$$\left[\mathbf{C}_N + \infty\mathbf{n}\mathbf{n}^T\right]^{-1} = \mathbf{C}_\infty^{-1} = \mathbf{C}^{-1} - \frac{\mathbf{C}^{-1}\mathbf{n}\mathbf{n}^T\mathbf{C}^{-1}}{\mathbf{n}^T\mathbf{C}^{-1}\mathbf{n}} \tag{6.32}$$

where

$$\mathbf{C}_\infty^{-1}\mathbf{n} = \mathbf{C}^{-1}\mathbf{n} - \frac{\mathbf{C}^{-1}\mathbf{n}\left(\mathbf{n}^T\mathbf{C}^{-1}\mathbf{n}\right)}{\mathbf{n}^T\mathbf{C}^{-1}\mathbf{n}} = 0 \tag{6.33}$$

Hence we can use $\mathbf{C}_\infty^{-1}$ as the inverse covariance matrix for all the bands.

Let us return to the free energy framework. We can write down the free energy:

$$F(\Xi) = -\sum_{k=1}^{K}\sum_{\mathbf{a}^{(k)}}\left\{Q(\mathbf{a}^{(k)};\Xi)\log P(\mathbf{a}^{(k)}|\mathcal{D}) - Q(\mathbf{a}^{(k)};\Xi)\log Q(\mathbf{a}^{(k)};\Xi)\right\} \tag{6.34}$$

The summation over all bands and the summation over $\mathbf{a}^{(k)} = (a_1^k, a_2^k.\cdots, a_N^k)$ (the vector of which data points have been assigned to band k at every $\mathbf{k}$-point) are equivalent to a summation over all possible allocation matrices $\{\mathbf{A}_n\}_{n=1}^N$. Again, we are relaxing the constraints of the permutation matrix to make the problem tractable.

We shall use a separable approximating distribution,

$$Q(\mathbf{a}^{(k)};\Theta) = \prod_{n=1}^{N} q_n^{a_n^k k}, \tag{6.35}$$

where

$$q_n^{jk} = \exp\left(\xi_n^{jk}\right) \tag{6.36}$$

Let us define every band as a Gaussian process with covariance function

$$C(\mathbf{k}_n,\mathbf{k}_m;\Theta) = \theta_1\exp\left(-\frac{1}{2r^2}\sum_{s=0}^{S}|\mathbf{k}_n - \mathbf{M}_s\mathbf{k}_m|^2\right) + \theta_2 + \delta_{nm}\theta_3 \tag{6.37}$$

The length scale r is the same for each direction in $\mathbf{k}$-space and is set to approximately half the average Brillouin zone width so that the Gaussian process generates smooth most probable interpolants. θ_1 is set to the square of the greatest variation in energy expected across a single band. θ_2 is set to infinity so that the inverse covariance matrix, $\mathbf{C}_\infty^{-1}$, has the same form given in equation 6.32. θ_3 is set to a small number ($\simeq 10^{-5}\theta_1$) to reflect the low noise level while preserving numerical stability. The matrices $\{\mathbf{M}_s\}$ describe the crystal symmetry where $\mathbf{M}_0$ is always the identity.

The variation of the matrix elements, $S_c(\mathbf{k})$, over $\mathbf{k}$-space has similar characteristic length scales to the variation of the energy. Hence we shall also model the variation of all the matrix elements using the covariance matrix $\mathbf{C}_\infty^{-1}$. The matrix elements must be scaled so that the θ_1 parameter is appropriate.

We now have data $\mathcal{D}$ which consists of the set of N $\mathbf{k}$-points $\{\mathbf{k}_n\}$, the stack of K energies $\{\tilde{t}_n^k\}$ at each of the $\mathbf{k}$-points and the C matrix elements $\{S_n^{kc}\}$ associated with each of the energies. The free energy is now

$$F(\Xi) = -\frac{1}{2}\sum_{k=1}^{K}\sum_{n=1}^{N}\sum_{j=1}^{K}\sum_{c=0}^{C}\left\{q_n^{jk}\left((\mathbf{C}_\infty^{-1})_{nn}(S_n^{jc})^2 + \sum_{m\neq n}\sum_{i=1}^{K} q_m^{ik}(\mathbf{C}_\infty^{-1})_{nm}S_n^{jc}S_m^{ic}\right) + q_n^{jk}\log q_n^{jk}\right\} \tag{6.38}$$

assuming a uniform prior on allocations. For the sake of brevity, we have defined $S_n^{j0} = \tilde{t}_n^j$ so that the summation over c is a summation over the energy and all the matrix elements.

Equation 6.38 initially looks daunting. However, we are attempting to find the minimum of $F(\Xi)$ which occurs when $Q(\mathbf{a}^{(k)};\Xi) = P(\mathbf{a}^{(k)})$. Hence by inspection of the equation, we can write down a re-estimation formula for ξ_n^{jk}:

$$\xi_n^{jk} = -\frac{1}{2}\sum_{c=0}^{C}(\mathbf{C}_\infty^{-1})_{nn}(S_n^{jc})^2 - \frac{1}{2}\sum_{m\neq n}\sum_{i=1}^{K} q_m^{ik}\sum_{c=0}^{C}(\mathbf{C}_\infty^{-1})_{nm}S_n^{jc}S_m^{ic} \tag{6.39}$$

$$= -\Upsilon_{nn}^{jj} - \sum_{m\neq n}\sum_{i=1}^{K} q_m^{ik}\Upsilon_{nm}^{ji} \tag{6.40}$$

where the "energy", Υ_{nm}^{ji}, is defined as

$$\Upsilon_{nm}^{ji} = \frac{1}{2}\sum_{c=0}^{C}(\mathbf{C}_\infty^{-1})_{nm}S_n^{jc}S_m^{ic} \tag{6.41}$$

Given this re-estimation formula for ξ_n^{jk}, how do we use it to allocate the data points to bands? Initially we define the labels of the bands by picking one $\mathbf{k}$-point, $\mathbf{k}_{ini}$, and fixing the q_{ini}^{jk}'s such that $q_{ini}^{jk} = \delta_{jk}$. We then pick another $\mathbf{k}$-point and apply the re-estimation formula, updating all the ξ_n^{jk}'s and hence the q's. We then normalize the q's using the `NORM` algorithm and the definition of q_n^{jk} given in equation 6.36. This tightens the permutation matrix constraints to partially compensate for the previous relaxation. We then move to another $\mathbf{k}$-point (not $\mathbf{k}_{ini}$) and repeat the procedure until the q's have reached equilibrium. The convergence of the algorithm can be gauged by considering the variation in the free energy of the approximating ensemble. However calculating the free energy is computationally costly. An alternative *ad hoc* guide to convergence is the variation in the number of iterations of the `NORM` algorithm required at each data point to normalize the q's. Once the number of iterations at every data point has settled down to a constant or is oscillating between two similar numbers then the algorithm has probably found a solution.

Care needs to be taken when choosing $\mathbf{k}_{ini}$ and deciding which order to pick $\mathbf{k}$-points for re-estimation. It is a good idea to pick $\mathbf{k}_{ini}$ to be in the centre of the Brillouin zone, away from any degeneracy on the boundaries. When choosing re-estimation points, I found that re-estimating the points in order of proximity to $\mathbf{k}_{ini}$, in terms of distance in $\mathbf{k}$-space, expanding out from $\mathbf{k}_{ini}$ and then contracting back in produced more stable solutions in less time than a random choice or a choice that ignored the position of $\mathbf{k}_{ini}$.

Having obtained a set of normalized q's at each $\mathbf{k}$-point, we need to allocate the data points to particular bands. To do this we draw $\mathcal{O}(K)$ samples from $Q(\mathbf{R}_n|\{q_n^{jk}\})$ at every $\mathbf{k}$-point, where $\mathbf{R}_n$ is a true permutation matrix, and then pick the most probable $\mathbf{R}_n$ from these samples. We then use a Gaussian process with the covariance function given in equation 6.37 to model each band based on the allocations made by the permutation matrices. The posterior distribution of the k^{th} band can therefore be written as

$$P(\mathbf{u}_k|\{\mathbf{k}_n\},\Theta) = \frac{1}{Z}\exp\left(-\frac{1}{2}\mathbf{u}_k^T\mathbf{C}_N^{-1}\mathbf{u}_k\right) \tag{6.42}$$

where $\mathbf{u}_k = (\tilde{t}_1^{a_1^k}, \tilde{t}_2^{a_2^k}, \cdots, \tilde{t}_N^{a_N^k})$ and a_n^k is the label of the data point allocated to the k^{th} band at $\mathbf{k}_n$.

6.4.1 Incomplete Bands

A problem which the Gaussian process approach can encounter, due to its separation of allocation and interpolation, is the presence of incomplete bands in the data. When the data is generated by the *ab initio* code, a specific number of energies (and matrix elements) are generated at each $\mathbf{k}$-point. However it is possible that the highest band contained in the data and the band immediately above it cross and hence we are left with two halves of different bands at the top of the data. The allocation algorithm, requiring that all the data be assigned to a band, might make mis-allocations near the top of the data which would then filter down to lower bands.

To avoid this problem, we introduce a buffer zone at the top of the data. Let us take this buffer to contain the upper K_2 ($< K$) data points at each $\mathbf{k}$-point. Rather than insisting that

$$\sum_{j=1}^{K} q_n^{jk} = 1 \;\forall\; k \tag{6.43}$$

$$\sum_{k=1}^{K} q_n^{jk} = 1 \;\forall\; j \tag{6.44}$$

we will now only attempt to fit $K - K_2 + \kappa$ bands to this data where κ is an integer between 1 and K_2. Thus we place the following new constraints on the q's:

$$\sum_{j=1}^{K} q_n^{jk} = 1 \text{ for } k = 1 \cdots K - K_2 + \kappa \tag{6.45}$$

$$\sum_{k=1}^{K-K_2+\kappa} q_n^{jk} = 1 \text{ for } j = 1 \cdots K - K_2 \tag{6.46}$$

and

$$\sum_{j=K-K_2+1}^{K} \sum_{k=1}^{K-K_2+\kappa} q_n^{jk} = \kappa \tag{6.47}$$

Here we are insisting that the bottom $K - K_2$ data points all belong to one of the $K - K_2 + \kappa$ bands. The upper K_2 data points might belong to one of these bands but it is possible for

$$\sum_{k=1}^{K-K_2+\kappa} q_n^{jk} = 0 \quad \text{for} \quad K - K_2 + 1 \leq j \leq K, \tag{6.48}$$

i.e., that one of the data points does not belong to any of the bands.

6.4.2 Performance of the Gaussian process Approach

The performance of the Gaussian process approach is greatly superior to that of the mixture model approach. The use of Gaussian processes to model the bands rather than neural networks increases the speed of the algorithm significantly. This allows us to run the algorithm until convergence which greatly increases the quality of the results.

Silicon

The first example shown in Figure 6.6 is that of Silicon with four atoms per unit cell. Data was generated at 48 **k**-points for the bottom 20 bands with a buffer zone of 2 data points to guard against incomplete bands. 10 matrix elements and 12 of the possible 48 symmetry matrices were used. Using all 48 symmetry matrices caused severe numerical problems for the algorithm unless the noise level was raised to an unrealistically high level. The length scales of the Gaussian process were set to 0.3 to force the bands to be smooth. A noise variance of 10^{-2} was included to improve the conditioning of the problem. The density of states was calculated by drawing 20000 random samples for the Brillouin zone and constructing a histogram of the most probable predicted energies for each band using 200 bins.

The density of states of Silicon calculated using the Gaussian process approach is close to the results obtained by Pickard (1997). However several of the peaks (most noticeably those at -2.0eV and 9.0eV) are not tall enough. The most probable source of this inaccuracy is the gradient of the bands. All the bands are supposed to be flat at the Gamma point $\mathbf{k} = \mathbf{0}$. Any band that is not flat will significantly affect the density of states, especially if the band is contributing to a noticeable van Hove singularity (which is the case at -2.0eV and 9.0eV).

As well as producing a most probable prediction for each band at each point in **k**-space, the Gaussian process model also provides error bars. In order to demonstrate the magnitude of the corresponding error bars on the density of states, eight samples of 2000 points were drawn from the posterior distribution of each band, $P(\mathbf{u}_k|\mathbf{C}_N, \Theta)$. This is in contrast to the approach of using the most probable predictions taken above. Only 2000 points were used to reduce computational expense. The samples were used to generate the densities of states shown in Figure 6.7. We can see that the densities vary considerably, despite the moderate magnitude of the error bars on the bands ($\simeq$ 1eV for most bands). This highlights the sensitivity of the density of states to small variations in the gradient and curvature of the bands. It is therefore unsurprising that the Gaussian process results fail to capture the small scale detail of the density of states accurately.

Another important test is the quality of the results as the amount of data is decreased. Figure 6.8 shows the density of states of Silicon calculated for 12, 24, 36 and 48 **k**-points. Using 12 **k**-points, we obtain the broad features of the Silicon density of states although there are many spurious spikes present. These are due to poorly defined bands which are flat across large areas of the Brillouin zone. As the number of **k**-points is increased, the number of spurious bands decreases and we begin to see the more detailed features of the DOS emerge. However it is not until we used all 48 **k**-points that an even close to acceptable result is obtained.

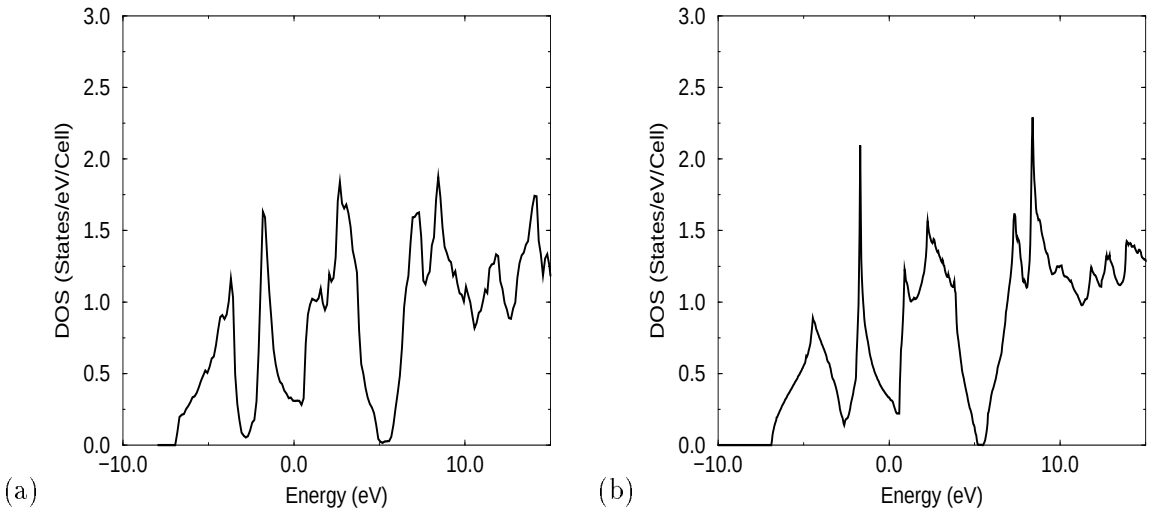


Figure 6.6: Density of States of Silicon : The figure shows a comparison between density of states of Silicon calculated using the Gaussian processes approach (a) and results obtained using Pickard's method (see Section 6.6) on the same data (b).

Hexagonal Graphite

The second crystal that was studied was hexagonal Graphite. Data was generated at 48 **k**-points for the bottom 16 bands with a buffer zone of 2 data points to guard against incomplete bands. 9 matrix elements and 12 of the 24 possible symmetry matrices were used. The length scales of the Gaussian process were set to 0.3 and a noise variance of 10^{-2} was included to improve the conditioning of the problem. The density of states was calculated by drawing 20000 random samples for the Brillouin zone and constructing a histogram of the most probable predicted energies for each band using 200 bins.

The differences between Pickard's results and those obtained using the Gaussian process approach for Graphite can be seen in Figure 6.9. While the band structure is broadly similar, there are several regions in which bands that are supposed to be touching have pulled apart (the fourth and fifth bands between H and K for example). This has introduced some incorrect features into the density of states, most seriously the band gap between 6eV and 7eV. The error bars on the band structure generated by the Gaussian process approach for Graphite were approximately the same as those found for Silicon ($\simeq$ 1eV). Figure 6.10 shows the large variational possible in the density of states despite the moderate magnitude of the error bars on the bands.

6.5 Random Matrix Approach

The two previous approaches to the calculation of the density of states have assumed that the approximation of treating the bands as smooth is valid. Certainly, looking at a one dimensional slice through the band structure, replacing the cusps by the crossing of smooth bands and then allocating the data to smooth bands seems reasonable. However, we must consider how these "crossing" behave in four dimensions.

We have, so far, assumed the bands are well defined surfaces in four dimensions. Investigation by Pickard using curvature information seems to suggest that this is the case. However, the energies are

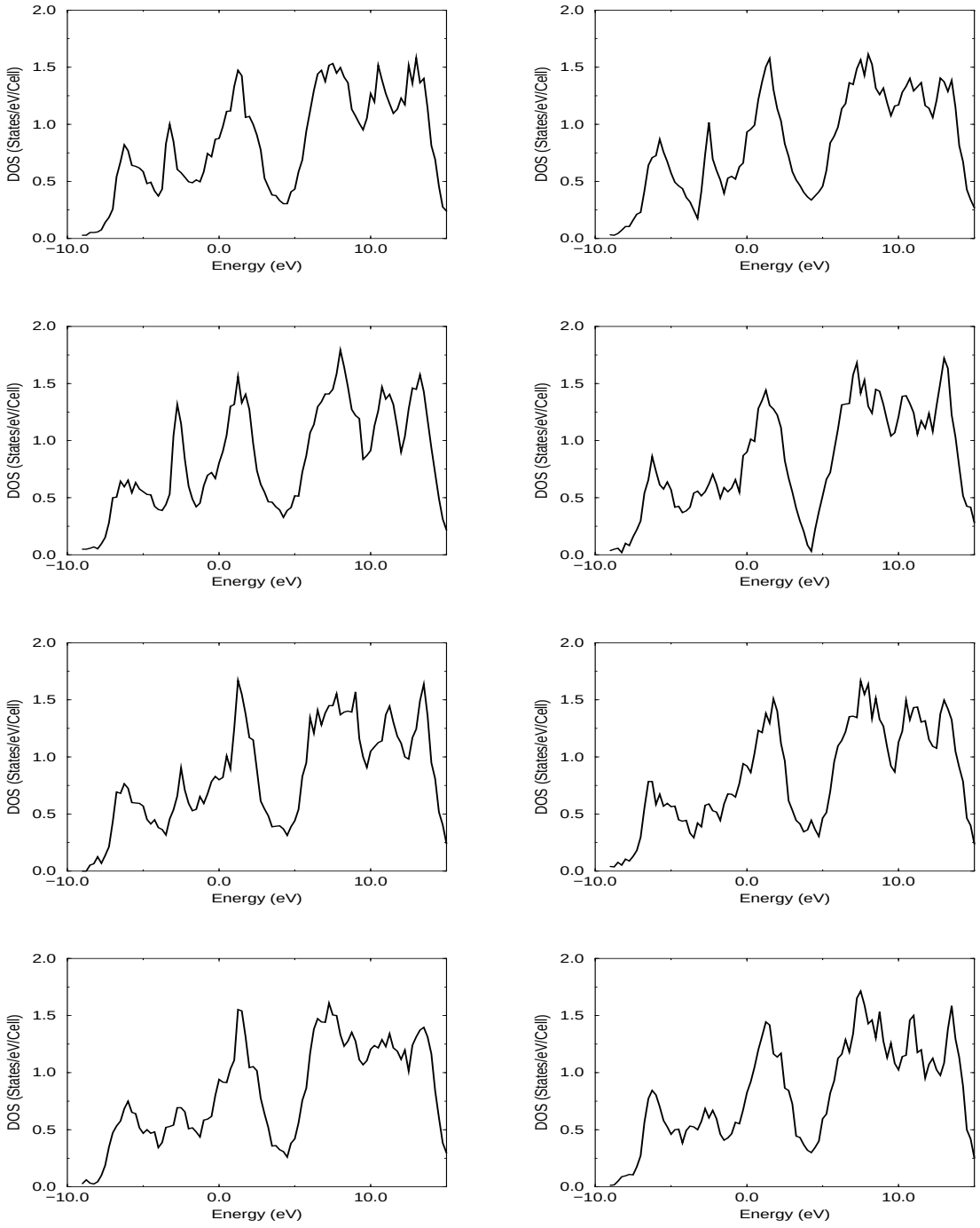


Figure 6.7: Samples from the Posterior for Silicon : Eight samples of 2000 points were drawn from the posterior distribution of each band found using the Gaussian process approach. These were then used to generate the densities of states shown in the Figure. Despite the moderate error bars ($\simeq$ 1eV for most bands) on the most probable predictions of the Gaussian processes, the variation in the densities of states is considerable.

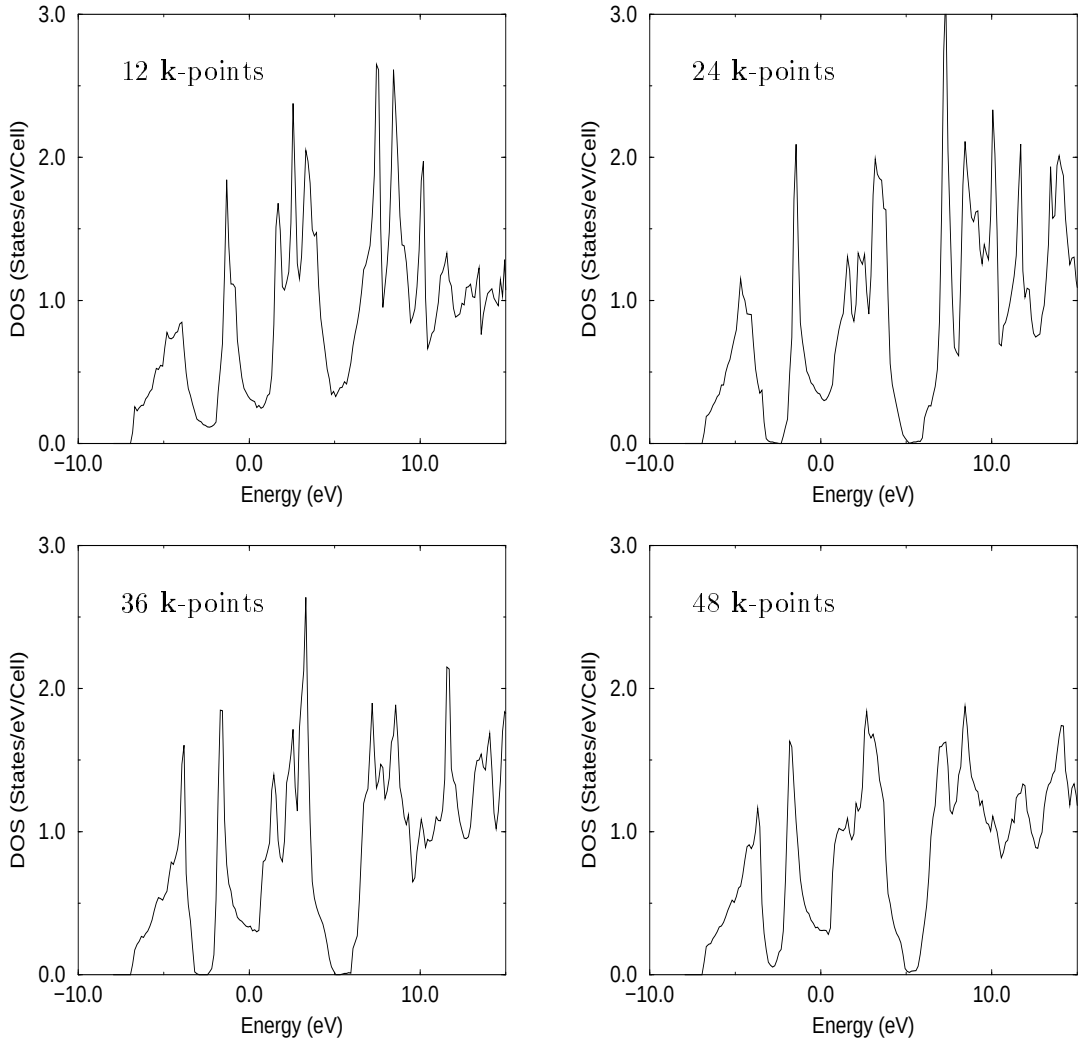


Figure 6.8: Sensitivity of DOS to number of k-points : The figure shows the density of states of Silicon calculated using the Gaussian process method with a progressively increasing number of **k**-points. While the broad features are present for fewer **k**-points, details, especially in the upper bands, do not appear until the number of **k**-points has reached 48.

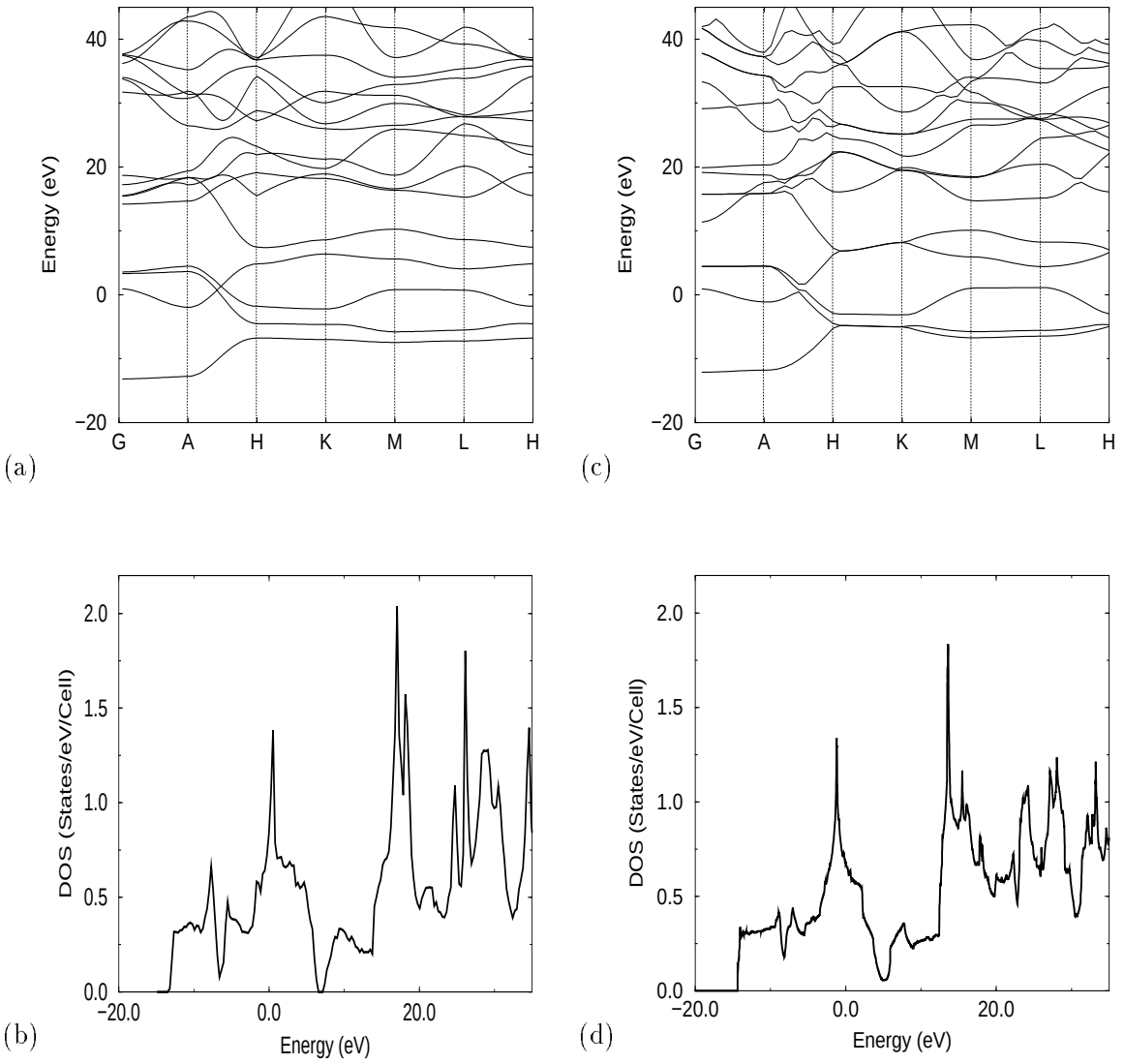


Figure 6.9: Band Structure and Density of States of Graphite : The figure shows a comparison between the band structure and density of states of Graphite calculated using the Gaussian processes approach, (a) and (b), and results obtained using Pickard's method on the same data, (c) and (d) (see Section 6.6). The letters at the base of the band structure plots denote specific points in the Brillouin zone between which the slices were taken.

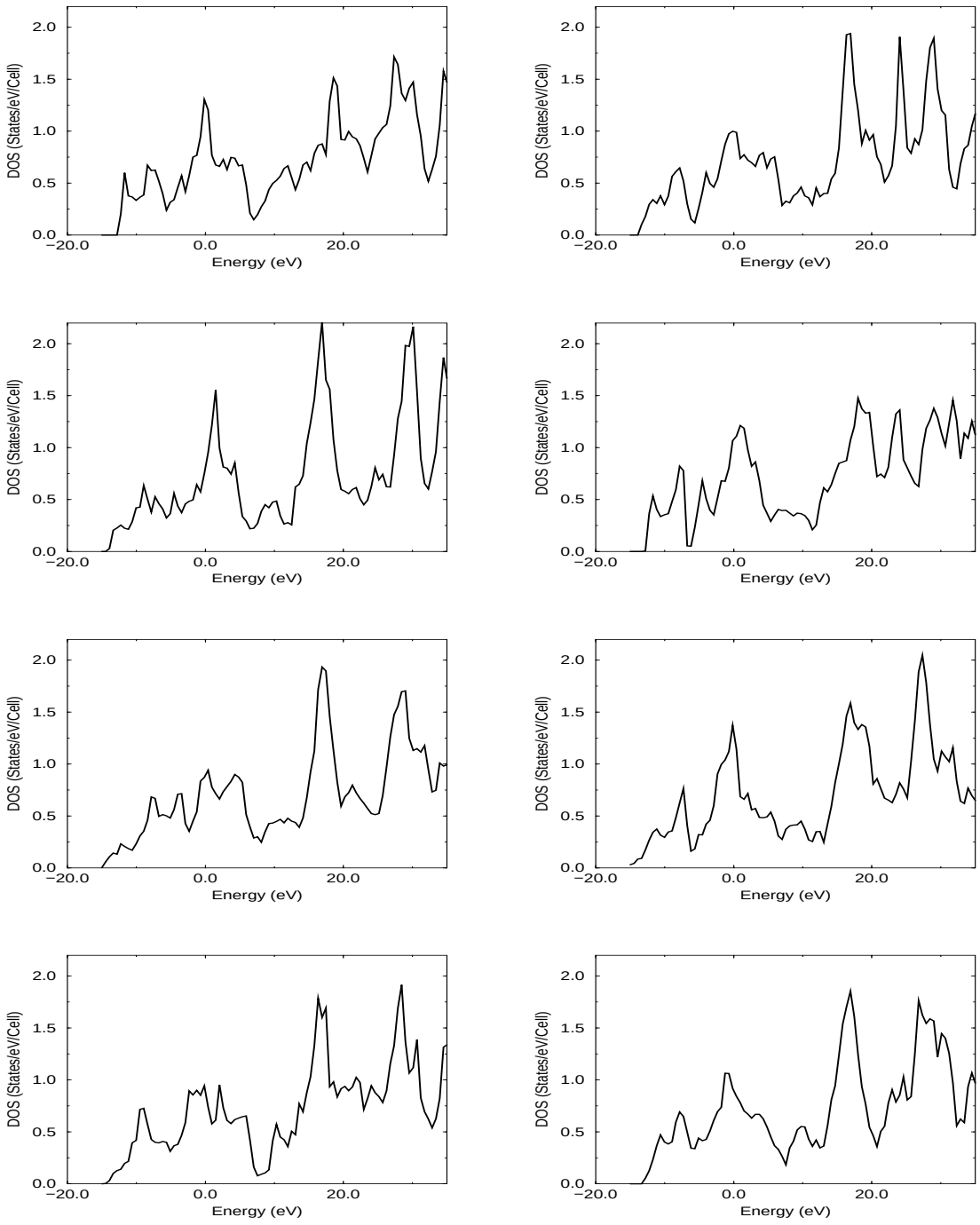
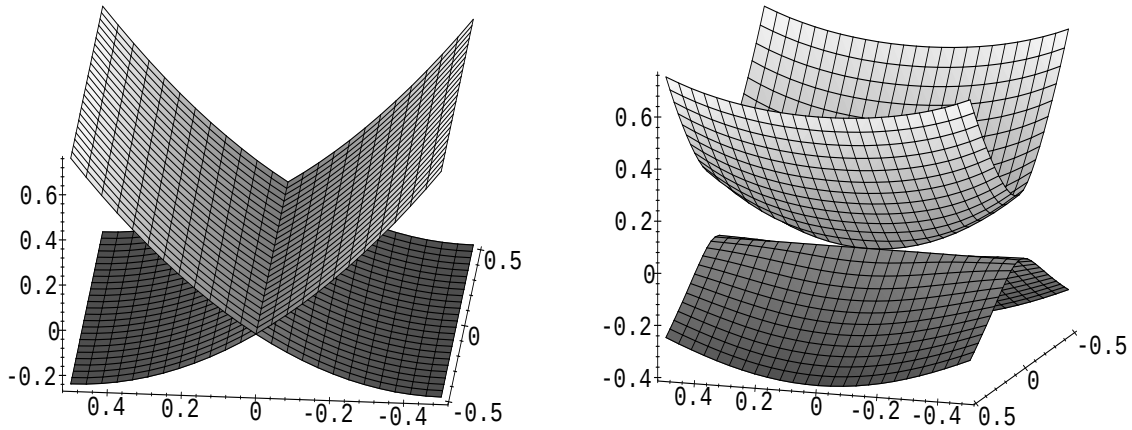


Figure 6.10: Samples from the Posterior for Graphite : Eight samples of 2000 points were drawn from the posterior distribution of each band found using the Gaussian process approach. These were then used to generate the densities of states shown in the Figure. Despite the moderate error bars ($\simeq$ 1eV for most bands) on the most probable predictions of the Gaussian processes, the variation in the densities of states is considerable.



(a) (b)

Figure 6.11: Eigenvalues of $\mathbf{H}(\mathbf{k})$: This figure shows the variation of the eigenvalues of a symmetric 2×2 matrix $\mathbf{H}(\mathbf{k})$ whose elements $H_{mn}(\mathbf{k})$ are continuous functions of position in 2D space. (a) shows, for one choice of the functions, a standard "crossing" where the eigenvalues appear to behave like smooth functions of $\mathbf{k}$, touching along a line. (b) shows, for another choice of the functions, a "kissing" where the eigenvalues touch at a point.

the eigenvalues of a matrix whose elements vary with respect to $\mathbf{k}$. If we construct a matrix whose elements have an arbitrary but continuous dependence on $\mathbf{k}$, we find that, as well as the surface-like "crossings" hypothesised for the band structure, we also obtain band "kissings" (Figure 6.11). Along one section it might appear that the bands cross but along another section they might repel each other. Unique labelling of the data points into smooth bands would therefore be impossible. Such kissings cannot be accommodated by the mixture model or Gaussian process methods.

It would therefore seem appropriate to model the energies as though they were eigenvalues from a matrix. Not only would this allow us to accommodate any expected geometry of the bands but it would also remove the need for allocation and also remove any problems caused by incomplete bands. The difficulty lies in constructing a prior on the energies which embodies the properties of a set of eigenvalues. In order to do this, we turn to the field of random matrix theory.

The statistics of Gaussian ensembles and random matrix theory in general, while pioneered in the 1960's by Wigner (1957), Porter and Rosenzweig (1960) and Dyson (1962) for the study of nuclear energy levels, has recently been used to explain and predict the behaviour of complex systems (for example Odlyzko (1987), Weaver (1989), Mucciolo *et al.* (1994)). There is not space in this thesis to give a complete introduction to the subject nor to investigate anything other than a very specific problem due to the mathematical complexity involved. A superb review of the fundamentals of the field can be found in Mehta (1991).

6.5.1 Gaussian Ensembles

Quantum systems can be described in terms of their Hamiltonians which can, in turn, be described by Hermitian matrices. Such systems may have certain symmetry constraints (time-reversal symmetry, rotational symmetry etc.). We shall assume that, aside from these constraints, the system is so complex that the elements of the matrices representing the Hamiltonians are essentially random. This is known as the property of maximum statistical independence of the elements under

the symmetry requirements and is crucial to our later definitions of Gaussian ensembles.

The matrix representing the Hamiltonian can fall into three distinct categories depending on whether the Hamiltonian is invariant under time reversal, given maximum statistical independence:

Invariant - Even Spin : If the system is invariant under time reversal and has even spin then the resulting matrix is real symmetric and belongs to the Gaussian Orthogonal Ensemble (GOE).

Invariant - Odd Spin : If the system is invariant under time reversal and has odd spin then the resulting matrix is self-dual Hermitian[1] and belongs to the Gaussian Symplectic Ensemble (GSE).

Not Invariant : If the system is not invariant under time reversal then the resulting matrix is Hermitian and belongs to the Gaussian Unitary Ensemble (GUE).

Let us now formally define each of these three ensembles. We shall then go on to use their properties to derive the joint probability density function of their eigenvalues.

Gaussian Orthogonal Ensemble : The Gaussian orthogonal ensemble is defined in the space of real symmetric matrices, $\mathbf{H}$, by the two requirements:

1. The probability $P(\mathbf{H})\mathrm{d}\mathbf{H}$ is invariant under real orthogonal transformations:

$$P(\mathbf{H})\mathrm{d}\mathbf{H} = P(\mathbf{H}')\mathrm{d}\mathbf{H}' \tag{6.51}$$

where

$$\mathbf{H}' = \mathbf{W}^T\mathbf{H}\mathbf{W} \tag{6.52}$$

and

$$\mathbf{W}^T\mathbf{W} = \mathbf{W}\mathbf{W}^T = 1. \tag{6.53}$$

2. The probability density function $P(\mathbf{H})$ is a product of functions, each of which depends on at most one variable:

$$P(\mathbf{H}) = \prod_{i \le j} f_{ij}(H_{ij}) \tag{6.54}$$

Gaussian Unitary Ensemble : The Gaussian unitary ensemble is defined in the space of Hermitian matrices, $\mathbf{H}$, by the two requirements:

[1] A self-dual Hermitian matrix is best described in terms of quaterions. If the elements of $\mathbf{H}$ are themselves 2×2 quaterions, i.e.,

$$H_{ij} = \begin{bmatrix} a_{ij} & b_{ij} \\ c_{ij} & d_{ij} \end{bmatrix} \tag{6.49}$$

then for $\mathbf{H}$ to be self-dual Hermitian,

$$H_{ij}^{\dagger} = \overline{H}_{ij} = H_{ji} \tag{6.50}$$

where $H_{ij}^{\dagger}$ denotes the Hermitian conjugate of H_{ij} and $\overline{H}_{ij}$ denotes the quaterion conjugate. For a full description see Mehta (1991) pages 42-45

1. The probability $P(\mathbf{H})\mathrm{d}\mathbf{H}$ is invariant under unitary transformations:

$$P(\mathbf{H})\mathrm{d}\mathbf{H} = P(\mathbf{H}')\mathrm{d}\mathbf{H}' \tag{6.55}$$

where

$$\mathbf{H}' = \mathbf{U}^{-1}\mathbf{H}\mathbf{U} \tag{6.56}$$

where $\mathbf{U}$ is any unitary matrix.

2. The probability density function $P(\mathbf{H})$ is a product of functions, each of which depends on at most one variable:

$$P(\mathbf{H}) = \prod_{i\leq j} f_{ij}^{(0)}(H_{ij}) \prod_{i<j} f_{ij}^{(1)}(H_{ij}) \tag{6.57}$$

where $f_{ij}^{(0)}$ and $f_{ij}^{(1)}$ refer to functions of the real and imaginary parts of H_{ij} respectively.

Gaussian Symplectic Ensemble : The Gaussian symplectic ensemble is defined in the space of self-dual Hermitian matrices, $\mathbf{H}$, by the two requirements:

1. The probability $P(\mathbf{H})\mathrm{d}\mathbf{H}$ is invariant under symplectic transformations:

$$P(\mathbf{H})\mathrm{d}\mathbf{H} = P(\mathbf{H}')\mathrm{d}\mathbf{H}' \tag{6.58}$$

where

$$\mathbf{H}' = \mathbf{W}^{R}\mathbf{H}\mathbf{W} \tag{6.59}$$

Here $\mathbf{W}^{R}$ denotes the 'dual' of $\mathbf{W}$;

$$\mathbf{W}^{R}\mathbf{W} = \mathbf{W}\mathbf{Z}\mathbf{W}^{T} = 1 \tag{6.60}$$

where $\mathbf{Z}$ is the block diagonal matrix:

$$Z = \begin{bmatrix} 0 & +1 & 0 & 0 & \cdots \\ -1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & +1 & \cdots \\ 0 & 0 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix} \tag{6.61}$$

2. The probability density function $P(\mathbf{H})$ is a product of functions, each of which depends on at most one variable:

$$P(\mathbf{H}) = \prod_{i\leq j} f_{ij}^{(0)}(H_{ij}) \prod_{\lambda=1}^{3} \prod_{i<j} f_{ij}^{(\lambda)}(H_{ij}) \tag{6.62}$$

where the product over λ corresponds to a summation over the 2×2 quaterion matrices that are generally used to describe symplectic systems.

6.5.2 Joint Probability Density Function of the Eigenvalues

The joint probability density function of the elements of $\mathbf{H}$, a matrix which belongs to one of the Gaussian Ensembles described above, can be written

$$P(\mathbf{H}) = \exp\left(-a\ \mathrm{tr}\mathbf{H}^2 + b\ \mathrm{tr}\mathbf{H} + c\right) \tag{6.63}$$

where a is real and positive and b and c are real. In terms of the eigenvalues of $\mathbf{H}$, $\{\lambda_k\}$,

$$P(\mathbf{H}) = \exp\left(-a\sum_{k=1}^{K}\lambda_k^2 + b\sum_{k=1}^{K}\lambda_k + c\right) \tag{6.64}$$

A derivation of the above formulae can be found in Appendix C.

Given $P(\mathbf{H})$, we wish to find an expression for $P(\{\lambda_n\})$. Again we shall concentrate on the derivation for the GOE. The derivations for the GUE and GSE are similar. For the GOE, $\mathbf{H}$ has $\frac{1}{2}K(K+1)$ real independent parameters whereas $\{\lambda_k\}$ has only K. In order to find $P(\{\lambda_n\})$, we shall introduce a further $\frac{1}{2}K(K-1)$ parameters $\{p_l\}$ for $l = 1\cdots L$ $(L = \frac{1}{2}K(K-1))$, and hence write

$$P(\lambda_1,\cdots,\lambda_K,p_1,\cdots,p_L) = \exp\left(-a\sum_{k=1}^{K}\lambda_k^2 + b\sum_{k=1}^{K}\lambda_k + c\right)J(\{\lambda_k\},\{p_l\}) \tag{6.65}$$

where $J(\{\lambda_k\},\{p_l\})$ is the Jacobian,

$$J(\{\lambda_k\},\{p_l\}) = \left|\frac{\partial\left(H_{11}^{(0)},H_{12}^{(0)},\cdots,H_{KK}^{(0)}\right)}{\partial\left(\lambda_1,\cdots,\lambda_K,p_1,\cdots,p_L\right)}\right|. \tag{6.66}$$

We would then, ideally, like to choose the $\{p_l\}$ such that the Jacobian was a separable of function of the eigenvalues and the $\{p_l\}$. We could then integrate out the new parameters in order to obtain the j.p.d.f. over the eigenvalues alone. In order to decide what these new parameters might be, let us consider possible definitions of $\mathbf{H}$. In the GOE case, $\mathbf{H}$ can be defined in terms of a diagonal matrix of its eigenvalues and a real orthogonal matrix:

$$\mathbf{H} = \mathbf{U}\Lambda\mathbf{U}^T \tag{6.67}$$

where $\Lambda = \mathrm{diag}(\lambda_1,\cdots,\lambda_K)$ and

$$\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}. \tag{6.68}$$

The matrix $\mathbf{U}$ has $\frac{1}{2}K(K-1)$ independent parameters which we can choose to be U_{ij} for $i > j$. We now let $p_l = U_{ij}$ for $i > j$ and for $l = i + K(j-1)$. By taking the derivatives of equations 6.67 and 6.68 with respect to p_l and λ_k (see Mehta (1991) Chapter 3), we obtain

$$J(\{\lambda_k\},\{p_l\}) = \prod_{i<j}|\lambda_j - \lambda_i| f(p) \tag{6.69}$$

where $f(p)$ is independent of the eigenvalues, depending only on the $\{p_l\}$.

By using the expression for the Jacobian and integrating out the parameters $\{p_l\}$, we obtain the j.p.d.f. for the eigenvalues of the matrices of an orthogonal ensemble,

$$P(\{\lambda_k\}) = \prod_{i<j}|\lambda_j - \lambda_i|^{\beta}\exp\left(-a\sum_{k=1}^{K}\lambda_k^2 + b\sum_{k=1}^{K}\lambda_k + c'\right) \tag{6.70}$$

where c' is a new constant and $\beta = 1$. The solutions for the GUE and GSE cases are very similar to this result but have $\beta = 2$ and $\beta = 4$ respectively. Using the co-ordinate transform

$$x_k = \sqrt{\frac{2a}{\beta}}\lambda_k - \frac{b}{\sqrt{2a\beta}} \tag{6.71}$$

we can write the j.p.d.f. of the $\{x_k\}$ as

$$P(\{x_k\}) = C_{K\beta} \prod_{i<j} |x_j - x_i|^\beta \exp\left(-\frac{1}{2}\sum_{k=1}^{K} x_k^2\right). \tag{6.72}$$

The normalizing constant can be evaluated using Selberg's Integral (Selberg 1944);

$$\begin{aligned} C_{K\beta}^{-1} &= \int \cdots \int \mathrm{d}x_1 \cdots \mathrm{d}x_K \prod_{i<j} |x_j - x_i|^\beta \exp\left(-\frac{1}{2}\sum_{k=1}^{K} x_k^2\right) \\ &= (2\pi)^{K/2} \beta^{-K/2-\beta K(K-1)/4} \left(?\,(1+\beta/2)\right)^{-K} \prod_{k=1}^{K} ?\,(1+\beta k/2), \end{aligned} \tag{6.73}$$

and hence we can evaluate the constant c' from equation 6.70:

$$e^{c'} = C_{K\beta}\left(\frac{2a}{\beta}\right)^{K/2+\beta K(K-1)/4} \exp\left\{-\frac{Kb^2}{4a}\right\} \tag{6.74}$$

6.5.3 Implementation of the Random Matrix Approach

We can use the above theory to model our band structure data by treating the energies $\tilde{t}_n^k$ as though they are the eigenvalues of a matrix $\mathbf{H}(\mathbf{k})$. We will discuss the correctness of this approach in the next section. We assume that $\mathbf{H}(\mathbf{k})$ belongs to one of the Gaussian ensembles and its elements are functions of position in $\mathbf{k}$-space defined by a Gaussian process with covariance matrix $\mathbf{C}$:

$$P(\{\mathbf{H}(\mathbf{k}_n)\}_{n=1}^N) = \frac{1}{Z}\exp\left\{-\frac{1}{2}\sum_{ij}\sum_{mn} H_{ij}(\mathbf{k}_m) C_{mn}^{-1} H_{ij}(\mathbf{k}_n)\right\} \tag{6.75}$$

We can sample from the prior over matrix elements in order to see what sort of behaviour to expect from the eigenvalues. Figure 6.12 shows such a sample (for a one dimensional input space) and we can see its close resemblance to a slice through the band structure. The marginal distribution of the eigenvalues of $\mathbf{H}(\mathbf{k}_n)$ will correspond to equation 6.70. However, in order to perform inferences given the data we need the joint distribution of the eigenvalues at all of the $\mathbf{k}$-points, $P(\{\lambda_k^{(n)}\})$ where $n = 1 \cdots N$ and $k = 1 \cdots K$. Unfortunately, obtaining this distribution is mathematically challenging and no tractable expression exists at present.

Before further investigation into the model defined above, let us first investigate whether the data exhibits the correct behaviour to be accurately modelled using one of the Gaussian ensembles.

6.5.4 Is Random Matrix Theory applicable?

The theory of random matrices cannot be applied to every system. Even systems of eigenvalues are not necessarily covered by the theory as they must also obey the principle of maximum statistical independence. How can we then determine whether or not our band structure data can be modelled using Gaussian ensembles? There are several statistics of the data that we can calculate

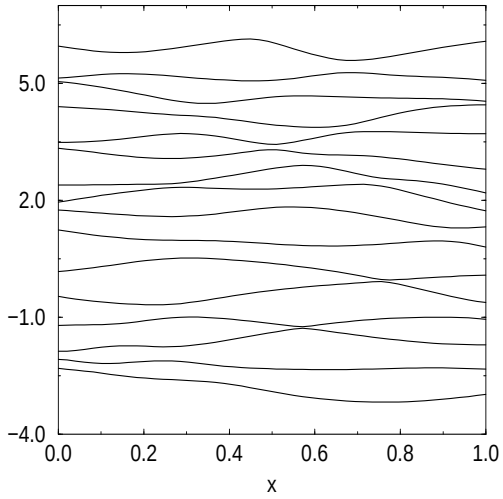


Figure 6.12: Samples from the random matrix/Gaussian process prior : The figure shows the eigenvalues of $\mathbf{H}$ corresponding to samples taken from a Gaussian process prior over the elements of $\mathbf{H}$. Each matrix element has the same covariance matrix $\mathbf{C}_N$, generated using the covariance function in equation 2.20 ($\theta_1 = 1.0$, $r = 2.0$, $\theta_2 = \theta_3 = 0.0$). We can see that the eigenvalues behave in a very similar manner to electronic bands. While generally smooth functions of x, some of the eigenvalues contain cusps at what appear initially to be crossings (for example the third and four lowest eigenvalues at $x \simeq 0.6$).

and compare with the predictions of random matrix theory. The most accessible of these is the distribution of the spacings between the eigenvalues at a given $\mathbf{k}$-point.

We can write down approximations to the distribution of the normalized energy spacings, s, for two of the three different Gaussian ensembles:

$$P(s) = \frac{1}{Z_\beta} s^\beta e^{-c_\beta s^2} \tag{6.76}$$

where $c_1 = \pi/4$ (GOE) and $c_2 = 4/\pi$ (GUE). The normalized spacing s is the actual energy spacing divided by the mean energy spacing, D. Figure 6.13(a) shows the estimated distribution of energy spacings for Silicon using the lowest 100 bands averaged over 48 $\mathbf{k}$-points. The $\mathbf{k}$-points were chosen so that they did not lie on points of high symmetry. A comparison with the predictions for GOE and GUE systems shows that the spacings of the lowest 100 bands do not obey the statistics of either of the Gaussian ensembles or of a Poisson model representing a random arrangement of energy levels.

Another useful measure of conformation to random matrix behaviour is the Δ statistic,

$$\Delta(L) = \min_{a,b} \left(\frac{1}{2L} \int_{-L}^{L} \left[N(E) - aE - b \right]^2 \mathrm{d}E \right), \tag{6.77}$$

where $N(E)$ is the number of levels below the energy E. Δ is a measure of the fluctuations away from the average level spacing) and again shows significant deviation from the predicted behaviour of GOE, GUE and Poisson models (see Figure 6.13(b)).

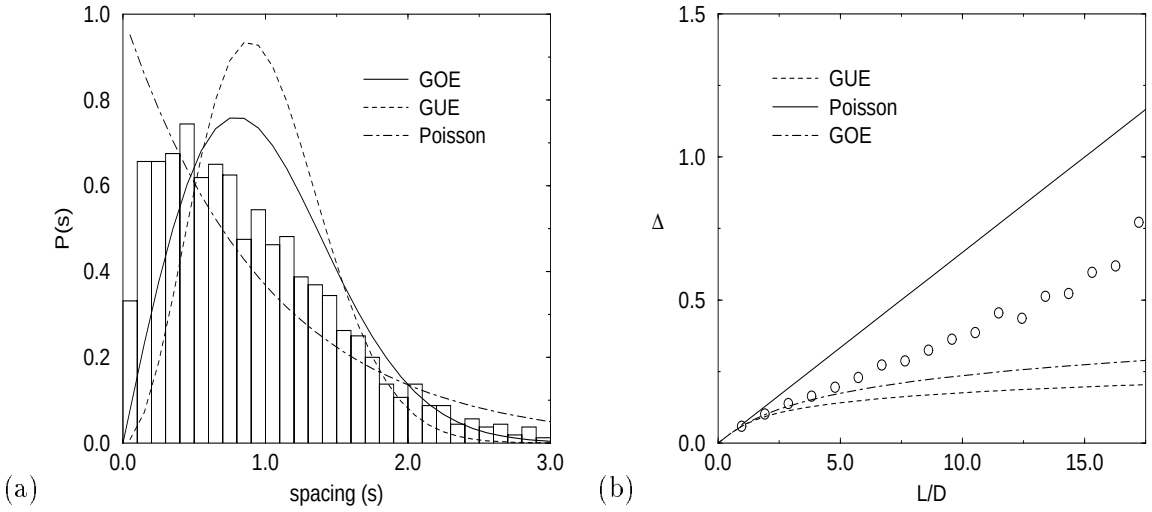


Figure 6.13: The distribution of level spacings and the Δ statistic for Silicon : (a) shows a histogram which is an estimate of the distribution of energy spacings for Silicon using the lowest 100 bands averaged over 48 **k**-points. The predictions obtained using the Wigner Surmise for the GOE and GUE and also using a Poisson model are shown for comparison. The spacings, s, are given in units of the mean energy spacing. The Δ statistic calculated for the same data is shown in (b) by open circles. Predictions for the GOE, GUE and Poisson models are also given. D is the average energy spacing. For both cases, the Silicon does not conform to any of the three models. This is due to problems introduced by using non-primitive unit cells and by considering the lower bands.

These result are somewhat odd as Mucciolo *et al.* (1994) found that the higher bands of Silicon did, indeed, show strong signs of GOE behaviour. However there are two possible explanations for this discrepancy. Firstly, we are considering the lowest bands. The quantum chaos that is exhibited strongly by Silicon (and other crystals) at high energies is less pronounced as we approach the ground state. Secondly unlike Mucciolo *et al.* (1994), we are considering a system that does not have a primitive unit cell. As the size of the unit cell increases, the Brillouin zone, being defined in terms of reciprocal lattice vectors, shrinks. Consequently, the bands are folded over on each other for non-primitive cells. This introduces new regions in which apparently separate bands come close together and so the mode of the estimate of $P(s)$ is displaced towards smaller values of s.

We therefore must conclude that the GOE and GUE (and also GSE) are not appropriate for modelling the lower bands of a crystal where the data is drawn from a non-primitive unit cell. However, this exploration of random matrix theory points toward some interesting models for future investigation. Of particular interest are models resulting for values of β smaller than 1. These will hopefully retain the useful correlations which exist between eigenvalues but will not object so strongly to eigenvalues having similar values, i.e., to band "kissing".

6.6 Pickard's Approach

Subsequent to the development of these three approaches, Pickard (1997) discovered that the matrix elements that had been used to aid with the allocation could be put to better use. Using the matrix elements, the gradient vectors and Hessian matrices of the bands at every data point $\tilde{t}_n^k$ could be calculated. Methfessel *et al.* (1983) had previously used the matrix elements to calculate

the gradient vectors but the ability to also obtain the Hessian matrices meant that a quadratic expansion approach could be taken to the problem of finding the density of states.

The results of Pickard's method are significantly better than those of the Gaussian processes approach. As well as avoiding the problems associated with allocation, Pickard's method allows us to use differential geometry to calculate the density of states from the band structure analytically (Methfessel *et al.* 1983). The removal of the time-consuming sampling approach is a significant benefit. The use of the second derivative information means that the gradients of the bands are more clearly defined and so the densities of states are more accurate for the same number of **k**-points and do not exhibit the occasional serious errors found in those produced using the Gaussian process approach (for example the erroneous band gap in the Graphite DOS). The band structures and densities of states generated using this method shown in Figure 6.6 and Figure 6.9 agree very well with experiment. Full details of Pickard's approach can be found in Pickard (1997).

6.7 Conclusions

None of the three approaches presented in this chapter can be recommended for calculating densities of states from limited *ab initio* data. The mixture model approach is very slow and produces poor results, the Gaussian processes approach produces reasonable results but is prone to occasional catastrophic errors and the random matrix approach is not applicable to the region of the band structure of interest. All three approaches do not achieve the quality of results obtained using Pickard's approach.

However, while not providing state-of-the-art methods for EELS calculation, the three approaches do present three interesting models which may find application elsewhere. In the case of the mixture model approach, we have shown how one can, using free energy techniques, place constraints on a mixture model that cannot easily be embodied in a prior on the weights. The Gaussian process approach once again illustrates the flexibility and ease of use associated with Gaussian processes when applied to regression. Finally, the theory of random matrices, while not providing an applicable model for the lower non-primitive bands, opens up a variety of new models for regression of correlated functions. The mathematics underlying the theory promises to provide many useful tools to the field of Bayesian inference.

Chapter 7

Conclusions

In the introduction to this thesis, I explored the need to construct models that are mathematically straightforward and yet have the flexibility to model complex phenomena without requiring large computational resources. I also suggested that the parameterization of our models should allow us to perform a continuous search over a large area of model space and that the parameters should be interpretable, both to facilitate the incorporation of prior beliefs into our models and to allow a deeper understanding about the system generating the data. Gaussian processes were then put forward as candidates which would fulfill these demands. Have Gaussian processes lived up to our expectations?

In chapter 2, I introduced Gaussian processes from the Bayesian perspective and showed how they can be used as regression models. Their underlying linearity simplifies the Bayesian analysis and allows us to obtain analytical expressions for the predictive distributions. The hyperparameters of commonly used covariance functions are readily interpretable and, by varying them, we can explore large regions of model space. While several possible forms for the covariance function are discussed in chapter 2, the development of new covariance functions is a vital part of further research into Gaussian processes.

As well as being mathematically simple and easily interpretable, Gaussian processes are also computationally inexpensive. For small amounts of training data ($N < 400$), the inversion of the covariance matrix is a minimal cost. As the size of the training data set increases, the approximate methods of Skilling (1993) described in chapter 3 keep the cost of inversion down to manageable levels. Unfortunately, if the number of hyperparameters and the amount of data increases significantly then the cost of evaluating the derivatives of the covariance matrix far outweighs the cost of inversion.

The examples given in chapter 3 show the flexibility and power of the Gaussian process framework. The modelling of the zebra finch neuron spike data, a difficult task for a standard regression model, is handled with ease by a Gaussian process. Both the non-stationary covariance function and the non-linear map approaches produce good results although the behaviour of the non-linear map is more interpretable than that of the non-stationary approach. The use of Gaussian processes to interpolate potential energy surfaces demonstrates how the models can be used on non-trivial problems to obtain high quality solutions. The more efficient parameterization of the Gaussian process allows us accurately to deal with higher dimensional surfaces than is possible using neural networks, opening the way to the investigation of more complex molecules.

The application of variational methods to the Gaussian processes approach to classification discussed in chapters 4 and 5 shows great promise. The binary case produces results that are comparable to the best of present classification techniques. The multi-class case is less straightforward but initial results are also promising. It is a great pity that no analytic proof for the lower bound for the multi-class case could be found. Variational methods are a powerful ally for Gaussian processes

and hopefully further research into multi-dimensional bounds will allow the application of Gaussian processes to a wider range of problems.

The connections between random matrix theory and Gaussian processes discussed in chapter 6 also offer promising directions of research. The idea of expressing a prior over a set of interacting functions may have many applications in the area of solid state physics and beyond. It will be interesting to see if the ideas and mathematical tools of random matrix theory can be carried across to the Gaussian process framework with significant benefit.

As the power of computers steadily increases, the properties we look for in our models will change. For problems where even minute gains in performance are of paramount importance, the computational cost of a model will be superceded in importance by the model's flexibility and its ability to capture complex behaviour and prior beliefs. For tasks which need to be repeated many times and in environments where expert knowledge is not always available, ease of use and interpretability will become more important. Gaussian processes have a great deal to offer in both of these scenarios. While they do not provide an optimal model for all problems, Gaussian processes represent a significant step forward in the search for general regression and classification techniques. After many years of neglect by the Bayesian modelling community, I hope that Gaussian processes and other associated infinite models will now receive the recognition they deserve.

After three minutes of fumbling around in the dark, Brad is confused. The light switch should definitely be around here somewhere. Reaching out into the blackness, his hand collides with something solid. Brad reaches forward with his other hand, trying to make out the form of the new object. It has a long, thick handle and on one end is a large piece of sharp metal

Appendix A

Principal Gaussian Process Equations

Given a set of data $\mathcal{D} = \{\mathbf{x}_n, t_n\}_{n=1}^N$, a covariance function, $C(\mathbf{x}_n, \mathbf{x}_m; \Theta)$, and a set of hyperparameters, Θ, we can use a Gaussian process model to calculate the predictive posterior distribution

$$\begin{aligned} P(t_{N+1}|\mathcal{D}, C(\mathbf{x}_n, \mathbf{x}_m; \Theta), \mathbf{x}_{N+1}) &= \frac{P(\mathbf{t}_{N+1}|C(\mathbf{x}_n, \mathbf{x}_m; \Theta), \mathbf{x}_{N+1}, \{\mathbf{x}_n\})}{P(\mathbf{t}_N|C(\mathbf{x}_n, \mathbf{x}_m; \Theta), \{\mathbf{x}_n\})} & \text{(A.1)} \\ &= \frac{Z_N}{Z_{N+1}} \exp\left[-\frac{1}{2}\left(\mathbf{t}_{N+1}^T \mathbf{C}_{N+1}^{-1} \mathbf{t}_{N+1} - \mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N\right)\right] & \text{(A.2)} \end{aligned}$$

where $\mathbf{t}_{N+1}$, $\mathbf{t}_N$, $\mathbf{C}_{N+1}$ and $\mathbf{C}_N$ are defined as in Section 2.2. This distribution is a Gaussian with respect to t_{N+1}. In order to to find its mean and variance, consider the partitioned form of $\mathbf{C}_{N+1}$:

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_{N+1} \\ \mathbf{k}_{N+1}^T & \kappa \end{bmatrix} \tag{A.3}$$

Defining its inverse as

$$\mathbf{C}_{N+1}^{-1} = \begin{bmatrix} \mathbf{M}_N & \mathbf{m}_{N+1} \\ \mathbf{m}_{N+1}^T & \mu \end{bmatrix} \tag{A.4}$$

and using the fact that $\mathbf{C}_{N+1}\mathbf{C}_{N+1}^{-1} = \mathbf{I}_{N+1}$, we can write

$$\begin{aligned} \mathbf{C}_N\mathbf{M}_N + \mathbf{k}_{N+1}\mathbf{m}_{N+1}^T &= \mathbf{I}_N & \text{(A.5)} \\ \mathbf{C}_N\mathbf{m}_{N+1} + \mu\mathbf{k}_{N+1} &= \mathbf{0} & \text{(A.6)} \\ \mathbf{k}_{N+1}^T\mathbf{M}_N + \kappa\mathbf{m}_{N+1}^T &= \mathbf{0}^T & \text{(A.7)} \\ \mathbf{k}_{N+1}^T\mathbf{m}_{N+1} + \kappa\mu &= 1 & \text{(A.8)} \end{aligned}$$

Pre-multiplying equation A.5 by $\mathbf{C}_N^{-1}$ and substituting the resulting expression into equation A.7, we obtain

$$\mathbf{m}_{N+1} = \frac{-\mathbf{C}_N^{-1}\mathbf{k}_{N+1}}{\left(\kappa - \mathbf{k}_{N+1}^T\mathbf{C}_N^{-1}\mathbf{k}_{N+1}\right)} \tag{A.9}$$

Substituting this into equation A.6 then gives us

$$\mu = \left(\kappa - \mathbf{k}_{N+1}^T\mathbf{C}_N^{-1}\mathbf{k}_{N+1}\right)^{-1} \tag{A.10}$$

Returning to equation A.2, we can write down the dependence of the exponent on t_{N+1} in terms of the elements of $\mathbf{C}_{N+1}^{-1}$ defined in equation A.4:

$$\mathbf{t}_{N+1}^T \mathbf{C}_{N+1}^{-1} \mathbf{t}_{N+1} - \mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N = \mu t_{N+1}^2 + 2(\mu \mathbf{m}_{N+1}^T \mathbf{t}_N) t_{N+1} + \text{const.} \tag{A.11}$$

Using the expressions derived above for μ and $\mathbf{m}_{N+1}$, it is straightforward to calculate the mean and variance of the Gaussian predictive distribution $P(t_{N+1}|\mathcal{D}, C(\mathbf{x}_n, \mathbf{x}_m; \Theta), \mathbf{x}_{N+1})$:

$$\hat{t}_{N+1} = \mathbf{k}_{N+1}^T \mathbf{C}_N^{-1} \mathbf{t}_N \tag{A.12}$$

$$\sigma_{\hat{t}_{N+1}}^2 = \kappa - \mathbf{k}_{N+1}^T \mathbf{C}_N^{-1} \mathbf{k}_{N+1} \tag{A.13}$$

Appendix B

The Approximate Inversion Algorithm

Here is a brief summary of the tridiagonal approximate inversion algorithm. This algorithm was used for all of the examples given in this thesis which use approximate techniques. The prime consideration when structuring this algorithm was computational cost. It was assumed that the matrix $\mathbf{A}$ was not so large that it could not be explicitly represented.

The algorithm given below calculates an approximation, $\mathbf{y}_K$, to $\mathbf{C}_N^{-1}\mathbf{u}$ by performing a conjugate gradient algorithm based on $\mathbf{A}$ where $\mathbf{C}_N = \mathbf{A} + \theta_3\mathbf{I}$. The (A) superscript has been suppressed for brevity. The algorithm is set to terminate when the ratio $\Delta Q/Q(\mathbf{y}_K)$ is smaller than ϵ or the number of iterations K exceeds K_{max}. The second case should be handled as a failure of the algorithm.

Set tolerance, ϵ, and iteration limit, K_{max}.
$\mathbf{g}_1 = \mathbf{u}$
$K = 1$
$\alpha_1 = |\mathbf{g}_1|^2$
do {
    if($K = 1$){
        $\mathbf{h}_1 = \mathbf{g}_1$
    } else {
        $\mathbf{h}_K = \mathbf{g}_K + \gamma_{K-1}\mathbf{h}_{K-1}$
    }

    $\mathbf{d} = \mathbf{A}\mathbf{h}_K$
    $\lambda_K = \alpha_K/\mathbf{h}_K^T\mathbf{d}$
    $\mathbf{g}_{K+1} = \mathbf{g}_K - \lambda_K\mathbf{d}$
    $\alpha_{K+1} = |\mathbf{g}_{K+1}|^2$
    $\gamma_K = \frac{\alpha_{K+1}}{\alpha_K}$

    if($K = 1$){
        $T_{11} = \frac{1}{\lambda_1} + \theta_3$
        $\overline{T}_{11} = \frac{1}{\lambda_1^2}(1+\gamma_1) + \theta_3\frac{\alpha_1}{\lambda_1}$
    } else {
        $T_{KK} = \frac{1}{\lambda_K} + \frac{\gamma_{K-1}}{\lambda_{K-1}} + \theta_3$
        $T_{(K-1)K} = -\frac{\alpha_K}{\alpha_{K-1}\lambda_{K-1}}$
        $T_{K(K-1)} = T_{(K-1)K}$
        $\overline{T}_{KK} = \frac{1}{\lambda_K^2}(1+\gamma_K) + \theta_3\frac{\alpha_K}{\lambda_K}$

$\overline{T}_{(K-1)K} = -\frac{\alpha_K}{\lambda_{K-1}\lambda_K}$
$\overline{T}_{K(K-1)} = \overline{T}_{(K-1)K}$
}

if($K = 1$){
$\hat{u}_1 = |\mathbf{u}|$
$(\hat{\mathbf{v}}_K)_1 = \alpha_1/\lambda_1$
} else {
$\hat{u}_K = 0$
$(\hat{\mathbf{v}}_K)_K = 0$
}

$Q(\mathbf{y}_K) = \frac{1}{2}\hat{\mathbf{u}}^T\mathbf{T}_K^{-1}\hat{\mathbf{u}}$
$Q^*(\mathbf{y}_K^*) = \frac{1}{2}\hat{\mathbf{v}}^T\overline{\mathbf{T}}_K^{-1}\hat{\mathbf{v}}$
$\Delta Q = \frac{1}{\theta_3}\left(\frac{1}{2}\mathbf{u}^T\mathbf{u} - Q^*(\mathbf{y}_K^*)\right) - Q(\mathbf{y}_K)$

$K = K + 1$

} while { $(\Delta Q/Q(\mathbf{y}_K) > \epsilon)$ and $(K < K_{max})$ }

$\mathbf{y}_K = \mathbf{E}_K\mathbf{T}_K^{-1}\hat{\mathbf{u}}$

To implement this algorithm we need to store the elements of $\mathbf{T}_K$,$\overline{\mathbf{T}}_K$,$\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}_K$. We also need to store all the gradient vectors $\{\mathbf{g}_k\}$ in order to calculate $\mathbf{y}_K$ when the algorithm terminates. Assuming we perform at most K_{max} iterations, we will require $\mathcal{O}(NK_{max})$ numbers to be stored (excluding the storage requirements for $\mathbf{A}$ itself). The algorithm also requires a scratch vector $\mathbf{d}$ of length N in which to store $\mathbf{A}\mathbf{h}_K$ temporarily.

Appendix C

Joint Probability Density Function of $\mathbf{H}$

Consider a matrix $\mathbf{H}$ which belongs to either the GOE, GUE or GSE. In order to calculate the joint probability density function (j.p.d.f.) of the eigenvalues of $\mathbf{H}$, we need to find an expression for the j.p.d.f. of the matrix elements. To do this, we rely on the following two lemmas:

Lemma 1 *All the invariants of a $K \times K$ matrix $\mathbf{H}$ under nonsingular similarity transformations $\mathbf{A}$ ($\mathbf{H} \to \mathbf{H}' = \mathbf{AHA}^{-1}$) can be expressed in terms of the traces of the first K powers of $\mathbf{H}$.*

Lemma 2 *If three continuous and differentiable functions $f_k(x)$, $x = 1, 2, 3$, satisfy the equation*

$$f_1(xy) = f_2(x) + f_3(y) \tag{C.1}$$

then they are necessarily of the form

$$f_k(x) = a\ln(x) + b_k \quad \textit{for} \quad k = 1, 2, 3 \tag{C.2}$$

where $b_1 = b_2 + b_3$.

Let us consider the transformation

$$\mathbf{H} = \mathbf{U}^{-1}\mathbf{H}'\mathbf{U} \tag{C.3}$$

where

$$\mathbf{U} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & \cdots & 0 \\ -\sin\theta & \cos\theta & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{C.4}$$

The matrix $\mathbf{U}$ is orthogonal, unitary and self-dual. In order to investigate the consequences of statistical independence on the j.p.d.f. of the matrix elements, let us consider the differential of $\mathbf{H}$ with respect to θ. We can show that

$$\frac{\partial \mathbf{H}}{\partial \theta} = \mathbf{AH} + \mathbf{HA}^T \tag{C.5}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{C.6}$$

The probability density function of $\mathbf{H}$ can be written

$$P(\mathbf{H}) = \prod_{(\alpha)} \prod_{i \le j} f_{ij}^{(\alpha)} \left(H_{ij}^{(\alpha)} \right) \tag{C.7}$$

where the summation over α represents either a summation over only real parts, over real and imaginary parts or over quaterions for the GOE, GUE and GSE respectively. $P(\mathbf{H})$ must be invariant under $\mathbf{U}$ and so

$$\sum_{\alpha,i,j} \frac{1}{f_{ij}^{(\alpha)}} \frac{\partial f_{ij}^{(\alpha)}}{\partial H_{ij}^{(\alpha)}} \frac{\partial H_{ij}^{(\alpha)}}{\partial \theta} = 0 \tag{C.8}$$

Let us explore the above equation in more detail for the orthogonal case (the unitary and symplectic cases are similar if notationally more complicated). Using equation C.5, equation C.8 for the $K \times K$ matrix $\mathbf{H}$ becomes

$$\begin{aligned} \left[\left(-\frac{1}{f_{11}^{(0)}} \frac{\partial f_{11}^{(0)}}{\partial H_{11}^{(0)}} + \frac{1}{f_{22}^{(0)}} \frac{\partial f_{22}^{(0)}}{\partial H_{22}^{(0)}} \right) \left(2H_{12}^{(0)} \right) + \frac{1}{f_{12}^{(0)}} \frac{\partial f_{12}^{(0)}}{\partial H_{12}^{(0)}} \left(H_{11}^{(0)} - H_{22}^{(0)} \right) \right] \\ + \sum_{i=3}^{K} \left[-\frac{1}{f_{1i}^{(0)}} \frac{\partial f_{1i}^{(0)}}{\partial H_{1n}^{(0)}} H_{2i}^{(0)} + \frac{1}{f_{2i}^{(0)}} \frac{\partial f_{2i}^{(0)}}{\partial H_{2i}^{(0)}} H_{1i}^{(0)} \right] = 0 \end{aligned} \tag{C.9}$$

The terms in the square brackets all depend on mutually exclusive sets of variables and their sum is zero. Hence each term must be a constant; for example,

$$-\frac{1}{f_{1i}^{(0)}} \frac{\partial f_{1i}^{(0)}}{\partial H_{1i}^{(0)}} H_{2i}^{(0)} + \frac{1}{f_{2i}^{(0)}} \frac{\partial f_{2i}^{(0)}}{\partial H_{2i}^{(0)}} H_{1i}^{(0)} = C \tag{C.10}$$

Dividing through the above equation by $H_{1i}^{(0)} H_{2i}^{(0)}$ and applying Lemma 2, we can show that

$$\frac{1}{f_{1i}^{(0)} H_{1i}^{(0)}} \frac{\partial f_{1i}^{(0)}}{\partial H_{1i}^{(0)}} = \frac{1}{f_{2i}^{(0)} H_{2i}^{(0)}} \frac{\partial f_{2i}^{(0)}}{\partial H_{2i}^{(0)}} = \text{constant} \tag{C.11}$$

and so, on integration,

$$f_{1i}^{(0)}(H_{1i}^{(0)}) = \exp\left(-a \left(H_{1i}^{(0)} \right)^2 \right) \tag{C.12}$$

Similar expressions can be obtained for the other terms in equation C.9. Recalling Lemma 1 and observing that the off-diagonal elements of $\mathbf{H}$ are raised to, at most, the second power in the exponential, we conclude that $P(\mathbf{H})$ is an exponential which contains traces of, at most, the second power of $\mathbf{H}$. Hence $P(\mathbf{H})$ can be written

$$P(\mathbf{H}) = \exp\left(-a \ \mathrm{tr}\mathbf{H}^2 + b \ \mathrm{tr}\mathbf{H} + c \right) \tag{C.13}$$

where a is real and positive and b and c are real. In terms of the eigenvalues of $\mathbf{H}$, $\{\lambda_k\}$,

$$P(\mathbf{H}) = \exp\left(-a\sum_{k=1}^{K}\lambda_k^2 + b\sum_{k=1}^{K}\lambda_k + c\right) \tag{C.14}$$

Bibliography

Amos, R. D. (1982) *CADPAC: Cambridge analytic derivatives package (Version 5.0).* Cambridge,UK: University of Cambridge.

Anderson, J. B. (1975) A random-walk simulation of the Schrodinger equation. *Journal of Chemical Physics* **63**: 1499.

Barber, D., and Williams, C. K. I. (1996) Gaussian Processes for Bayesian classification via Hybrid Monte Carlo. In *Advances in Neural Information Processing Systems 9*, ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press.

Barton, A. E., and Howard, B. J. (1982) *Discoveries of the Faraday Society* **73**: 45.

Bayes, T. (1763) An essay towards solving a problem in the doctrine of chances. *Philos. Trans. R. Soc. London* **53**: 370–418.

Bernoulli, J. (1713) *Ars Conjectandi*. Basel: Thurnisiorum.

Blochl, P. E., Jepsen, O., and Andersen, O. K. (1994) Improved tetrahedron method for Brillouin zone integrations. *Physical Review B - Condensed Matter* **49**: 16223–16233.

Bochner, S. (1979) *Harmonic analysis and the theory of probability.* Berkeley.

Bowman, J. M., and Kuppermann, A. (1975) *Chemical Physics Letters* **34**: 523.

Box, G. E. P., and Tiao, G. C. (1973) *Bayesian inference in statistical analysis*. Addison–Wesley.

Bridle, J. S. (1989) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neuro-computing: algorithms, architectures and applications*, ed. by F. Fougelman-Soulie and J. Hérault. Springer–Verlag.

Brown, D. F. R., Gibbs, M. N., and Clary, D. C. (1996) Combining *ab-initio* computations, neural networks and Diffusion Monte Carlo - an efficient method to treat weakly-bound molecules. *Journal of Chemical Physics* **105**: 7597–7604.

Bunker, P. R., Kofranek, M., Lischka, H., and Karpfen, A. (1988) An analytical 6-dimensional potential energy surface for $(hf)_2$ from *ab initio* calculations. *Journal of Chemical Physics* **89**: 3002–3007.

Connor, J. N. L., Jakubetz, W., and Manz, J. (1975) Exact quantum transition probabilities by the state path sum method: colinear $F + H_2$ reaction. *Molecular Physics* **29**: 347.

Cox, R. (1946) Probability, frequency, and reasonable expectation. *Am. J. Physics* **14**: 1–13.

Cressie, N. (1993) *Statistics for Spatial Data.* Wiley.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987) Hybrid Monte Carlo. *Physics Letters B* **195**: 216–222.

Dyson, F. J. (1962) Statistical theory of energy levels of complex systems i, ii and iii. *Journal of Mathematical Physics* **3**: 140–156,157–165,166–175.

Golub, G. H., and Loan, C. V. (1990) *Matrix Computation*. Baltimore: John Hopkins University Press.

Green, P. J. (1995) Reversible jump markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82**: 711–732.

Gregory, J. K., and Clary, D. C. (1995) 3-Body Effects on Molecular Properties in the Water Trimer. *Chemical Physics Letters* **237**: 39.

Gull, S. F. (1988) Bayesian inductive inference and maximum entropy. In *Maximum Entropy and Bayesian Methods in Science and Engineering, vol. 1: Foundations*, ed. by G. Erickson and C. Smith, pp. 53–74, Dordrecht. Kluwer.

Hestenes, M. R., and Stiefel, E. (1952) Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards* **49**: 409.

Hinton, G. E., and van Camp, D. (1993) Keeping neural networks simple by minimizing the description length of the weights. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pp. 5–13. ACM Press, New York, NY.

Hinton, G. E., and Zemel, R. S. (1994) Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, ed. by J. D. Cowan, G. Tesauro, and J. Alspector, San Mateo, California. Morgan Kaufmann.

Ho, T. S., and Rabitz, H. (1996) A general method for constructing multi-dimensional molecular potential energy surfaces from Ab-Initio Calculations. *Journal of Chemical Physics* **104**: 2584.

Ichikawa, K., Bhadeshia, H. K. D. H., and MacKay, D. J. C. (1996) Model for hot cracking in low-alloy steel weld metals. *Science and Technology of Welding and Joining* **1**: 43–50.

Jaakkola, T. S., and Jordan, M. I. (1996) Computing upper and lower bounds on likelihoods in intractable networks. In *Proceedings of the Twelfth Conference on Uncertainty in AI*, ed. by E. Horvitz. Portland, Oregon.

Jeffreys, H. (1939) *Theory of Probability*. Oxford Univ. Press. 3rd edition reprinted in paperback 1985.

Jennings, A. (1977) Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method. *Journal of the Institute of Mathematics and its Applications* **20**: 61–72.

Kitanidis, P. K. (1986) Parameter uncertainty in estimation of spatial functions: Bayesian analysis. *Water Resources Research* **22**: 499–507.

Kohn, W., and Sham, L. J. (1965) *Physical Review A* **140**: 1133.

Lanczos, C. (1950) An iteration method for the solution of the eigenvalue problem for linear differential and integral operators. *Journal of Research (National Bureau of Standards)* **45**: 255–282.

Laplace, P. S. d. (1812) *Théorie analytique des probabilités*. Paris: Courcier Imprimeur.

Lewicki, M. (1994) Bayesian modeling and classification of neural signals. *Neural Computation* **6** (5): 1005–1030.

Lowe, D. G. (1995) Similarity metric learning for a variable kernel classifier. *Neural Computation* **7**: 72–85.

MacKay, D. J. C. (1992a) Bayesian interpolation. *Neural Computation* **4** (3): 415–447.

MacKay, D. J. C. (1992b) The evidence framework applied to classification networks. *Neural Computation* **4** (5): 698–714.

MacKay, D. J. C. (1992c) Information based objective functions for active data selection. *Neural Computation* **4** (4): 589–603.

MacKay, D. J. C. (1992d) A practical Bayesian framework for backpropagation networks. *Neural Computation* **4** (3): 448–472.

MacKay, D. J. C. (1994) Bayesian methods for backpropagation networks. In *Models of Neural Networks III*, ed. by E. Domany, J. L. van Hemmen, and K. Schulten, chapter 6. New York: Springer-Verlag.

MacKay, D. J. C. (1995a) Free energy minimization algorithm for decoding and cryptanalysis. *Electronics Letters* **31** (6): 446–447.

MacKay, D. J. C. (1995b) Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems* **6**: 469–505.

MacKay, D. J. C. (1996) Hyperparameters: Optimize, or integrate out? In *Maximum Entropy and Bayesian Methods, Santa Barbara 1993*, ed. by G. Heidbreder, pp. 43–60, Dordrecht. Kluwer.

MacKay, D. J. C., and Miller, K. D. (1989) Analysis of Linsker's simulations of Hebbian rules. In *Advances in Neural Information Processing Systems II*, ed. by D. Touretzky, pp. 694–701.

MacKay, D. J. C., and Takeuchi, R., (1994) Interpolation models with multiple hyperparameters. Submitted to Statistics and Computing.

Matheron, G. (1963) Principles of geostatistics. *Economic Geology* **58**: 1246–1266.

McLachlan, G. J., and Basford, K.-E. (1988) *Mixture Models*. New York and Basel: Dekker.

Mehta, M. L. (1991) *Random Matrices*. San Diego: Academic Press, 2nd edition.

Mendel, J. M. (1994) Fuzzy logic systems for engineering: A tutorial. *Proceedings of the IEEE* **83**: 1293.

Methfessel, M. S., Boon, M. H., and Mueller, F. M. (1983) Analytic-quaratic method of calculating the density of states. *Journal of Physics C: Solid State Physics* **16**: 1949–1954.

Mucciolo, E. R., Capaz, R. B., Altshuler, B. L., and Joannopoulos, J. D. (1994) Manifestations of quantum chaos in electronic band structures. *Physical Review B - Condensed Matter* **50**: 8245–8251.

Neal, R. M. (1992) Bayesian mixture modelling. In *Maximum Entropy and Bayesian Methods, Seattle 1991*, ed. by C. Smith, G. Erickson, and P. Neudorfer, pp. 197–211, Dordrecht. Kluwer.

Neal, R. M. (1993) Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG–TR–93–1, Dept. of Computer Science, University of Toronto.

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. New York: Springer.

Neal, R. M. (1997) Monte Carlo Implementation of Gaussian process Models for Bayesian Regression and Classification. Technical Report CRG–TR–97–2, Dept. of Computer Science, University of Toronto.

Odlyzko, A. M. (1987) On the distribution of spacings between zeros of the zeta function. *Mathematics of Computation* **48**: 273–308.

O'Hagan, A. (1978) On curve fitting and optimal design for regression. *Journal of the Royal Statistical Society, B* **40**: 1–42.

Omre, H. (1987) Bayesian kriging - merging observations and qualified guesses in kriging. *Mathematical Geology* **19**: 25–39.

Payne, M. C., Teter, M. P., Allan, D. C., Arias, T. A., and Joannopoulos, J. D. (1992) Iterative minimization techniques for ab-initio total energy calculations - molecular dynamics and conjugate gradients. *Reviews of Modern Physics* **64**: 1045–1097.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo: Morgan Kaufmann.

Pickard, C. J., (1997) *Ab-Initio EELS*. University of Cambridge dissertation.

Poggio, T., and Girosi, F. (1989) A theory of networks for approximation and learning. Technical Report A.I. 1140, M.I.T.

Porter, C. E., and Rosenzweig, N. (1960) Statistical properties of atomic and nuclear spectra. *Annaless Acamdemiae Scientiarum Fennicae. Serie A6 Physica* **44**: 1–66.

Press, W., Flannery, B., Teukolsky, S. A., and Vetterling, W. T. (1988) *Numerical Recipes in C*. Cambridge.

Rasmussen, C. E., (1996) *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. University of Toronto dissertation.

Ripley, B. D. (1994) Flexible non-linear approaches to classification. In *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*, ed. by V. Cherkassky, J. H. Friedman, and H. Wechsler, ASI Proceedings. Springer-Verlag.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. C.U.P.

Selberg, A. (1944) Bemerkninger om et multiplet integral. *Norsk Matematisk Tidsskrift* **26**: 71–78.

Skilling, J. (1993) Bayesian numerical analysis. In *Physics and Probability*, ed. by W. T. Grandy, Jr. and P. Milonni, Cambridge. C.U.P.

Suhm, M. A., and Watts, R. O. (1991) Quantum Monte Carlo studies of the Vibrational states in Molecules and Clusters. *Physics Reports* **204**: 293.

Tully, J. C. (1980) *Advanced Chemical Physics* **42**: 63.

Wahba, G. (1990) *Spline Models for Observational Data.* Society for Industrial and Applied Mathematics. CBMS-NSF Regional Conference series in applied mathematics.

Waterhouse, S. R., and Robinson, A. J. (1995) Non-linear prediction of acoustic vectors using hierarchical mixtures of experts. In *Advances in Neural Information Processing Systems 7*, ed. by G. Tesauro, D. S. Touretzky, and T. K. Leen.

Weaver, R. L. (1989) Spectral statistics in elastodynamics. *Journal of the Acoustic Society of America* **85**: 1005–1013.

Wigner, E. P. (1957) Statistical properties of real symmetric matrices with many dimensions. In *Canadian Mathematical Congress*, pp. 174–184, Toronto, Canada. University of Toronto Press.

Williams, C. K. I., and Rasmussen, C. E. (1996) Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*, ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press.

Young, A. S. (1977) A Bayesian approach to prediction using polynomials. *Biometrika* **64**: 309–317.