Gaussian Process Regression with Heteroscedastic Residuals and Fast MCMC Methods

by

Chunyi Wang

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Statistics
University of Toronto

Abstract

Gaussian Process Regression with Heteroscedastic Residuals and Fast MCMC Methods

Chunyi Wang

Doctor of Philosophy

Graduate Department of Statistics

University of Toronto

2014

Gaussian Process (GP) regression models typically assume that residuals are Gaussian and have the same variance for all observations. However, applications with input-dependent noise (heteroscedastic residuals) frequently arise in practice, as do applications in which the residuals do not have a Gaussian distribution. In this thesis, we propose a GP regression model with a latent variable that serves as an additional unobserved covariate for the regression. This model (which we call GPLC) allows for heteroscedasticity since it allows the function to have a changing partial derivative with respect to this unobserved covariate. With a suitable covariance function, our GPLC model can handle (a) Gaussian residuals with input-dependent variance, or (b) non-Gaussian residuals with input-dependent variance, or (c) Gaussian residuals with constant variance. We compare our model, using synthetic datasets, with a model proposed by Goldberg, Williams and Bishop (1998), which we refer to as GPLV, which only deals with case (a), as well as a standard GP model which can handle only case (c). Markov Chain Monte Carlo methods are developed for the GPLC and GPLV models. Experiments show that when the data is heteroscedastic, both GPLC and GPLV give better results (smaller mean squared error and smaller negative log-probability density) than standard GP regression. In addition, if we do not assume Gaussian residuals, our GPLC model (as in case (b) above) is still generally nearly as good as GPLV when the residuals are in fact Gaussian. When the residuals are non-Gaussian, our GPLC model is better than GPLV.

Evaluating the posterior probability density function is the most costly operation when Markov Chain Monte Carlo (MCMC) is applied to many Bayesian inference problems. For GP models, computing the posterior density involves computing the covariance matrix, and then inverting the covariance matrix. The computation time for computing the covariance matrix is proportional to pn^2, and for the inversion is proportional to n^3, where p is the number of covariates and n is the number of training cases. We introduce MCMC methods based on the "temporary mapping and caching" framework (Neal, 2006), using a fast approximation, π^*, as the distribution needed to construct the temporary space. We propose two implementations under this scheme: "mapping to a discretizing chain", and "mapping with tempered transitions", both of which are exactly correct MCMC methods for sampling π, even though their transitions are constructed using an approximation. These methods are equivalent when their tuning parameters are set at the simplest values, but differ in general. We compare how well these methods work when using several approximations, finding on synthetic datasets that a π^* based on the "Subset of Data" (SOD) method is almost always more efficient than standard MCMC using only π. On some datasets, a more sophisticated π^* based on the "Nyström-Cholesky" method works better than SOD.

Acknowlegements

First and foremost I would like to express my deep and sincere gratitude to my Ph.D. advisor, Professor Radford Neal for his guidance, encouragement and support. I couldn't have completed the thesis work without his inspiration, patience and insightful direction.

I'd like to thank my PhD committee, Professor Radu Craiu and Professor Jeff Rosenthal, for their guidance, support and helpful suggestions. I'd also like to show my sincere gratitude to my final oral exam committee, Professor Patrick Brown, Professor John Davies, Professor Carl Rassmusen and Professor Ruslan Salakhutdinov. Their helpful comments and suggestions strengthen this thesis tremendously.

During my 8 years at the University of Toronto, I have had many great Professors who taught me interesting courses, directed me for projects, supervised me on teaching/research activities. It is these professors who kept me going forward for my study and research. I'd like to offer my sincere appreciation to David Brenner, Almut Burchard, Mike Evans, Don Fraser, Andrey Feuerverger, Alison Gibbs, Ian Graham, Sebastian Jamungal, Boris, Khesin, Keith Knight, Sheldon Lin, Philip McDunnough, Pierre Milman, Nancy Reid, James Stafford, Balint Virag, Chris Williams, Fang Yao and Zhou Zhou.

I'm also very lucky to have many intellegent fellow classmates, who shared their study and research experiences with me, helped me in many ways not only in my research but also in everyday life. These great friends made me miss Toronto every day: Yan Bai, Zeynep Baskurt, Yuxiang Chong, Andriy Derkach, Wei Lin, Madeleine Thompson, Avideh Sabeti, Cody Severinski, Alex Shestopaloff, Shivon Sue-Chee, Ramya Thinniyam, Panpan Wu, Weichi Wu, and Lizhen Xu.

Last but not least, I'd like to thank my family — my wife Zhi Li, my father Jian Wang, my mother Fuzhi Jin and my sister Junyi Wang — for their unconditional love, trust and support. Completing my PhD degree is the most challenging activity in my life so

far. Without the support from my family I couldn't have achieved what I have achieved. Thank you, mom, dad, sister and my dear wife. I know I'll make you proud.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Gaussian Process (GP) regression models have become popular in recent years in the machine learning community, mainly because these models are very flexible — one can choose from many covariance functions to achieve different degrees of smoothness or different degrees of additive structure, and for any given covariance function with unknown parameters, the values of the parameters can be automatically determined by the model. Standard GP regression models typically assume that the residuals are i.i.d., with Gaussian distributions that do not depend on the input covariates. However in many applications, the variances of the residuals actually depend on the inputs, and the distributions of the residuals are not necessarily Gaussian. For example, Silverman's motorcycle accident data (Silverman, 1985) shown in Figure 1.1 exhibits heteroscedastic residuals. Schmidt, *et al.* (1981) has the detailed description of the original experiment, a simulated motorcyle crash. The standard GP model assumes constant residual variance, and clearly overestimates the variance of acceleration when time is small (less than 10ms) and underestimates the variance when time is around 30 - 40 ms. We discuss the details of this example dataset in Section 2.4.2.

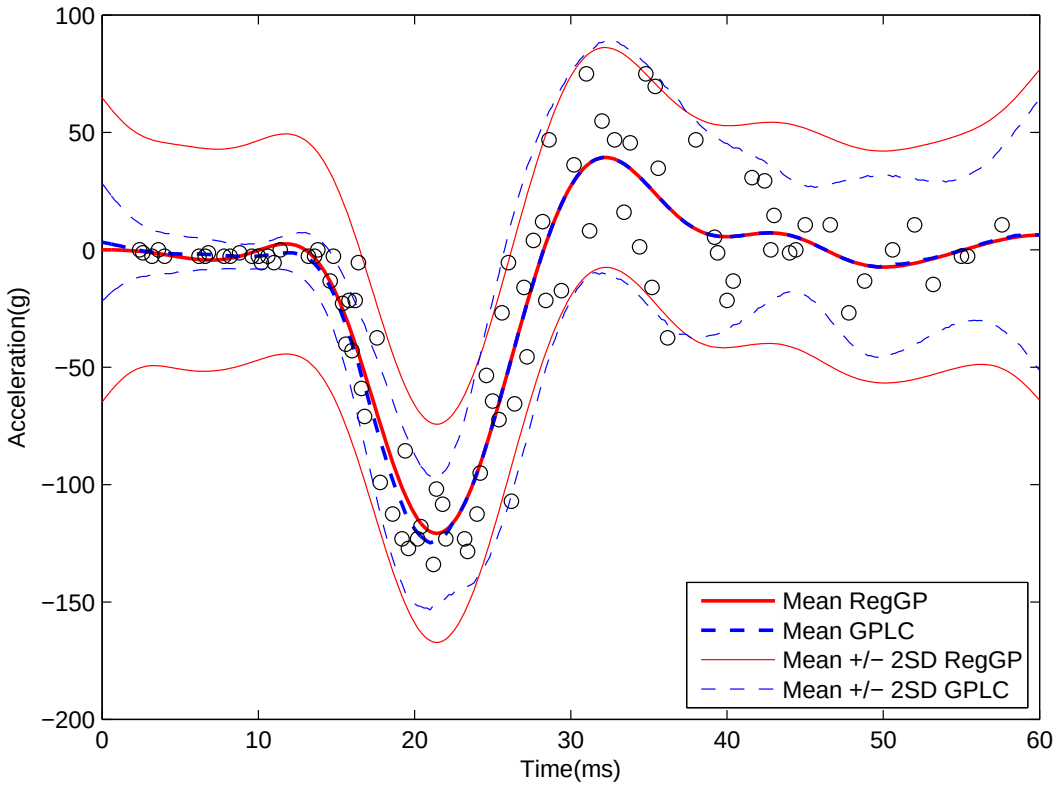

Figure 1.1: Silverman's motorcycle data. These 94 observations were obtained from `http://www.stat.cmu.edu/~larry/all-of-statistics/=data/motor.dat`
Note that the two mean curves are almost identical, and so are somtimes not separately visible. The mean curves are produced by making predictions on 300 time points evenly spaced between 0 and 60 ms. The standard deviations are the predictive standard deviations corresponding to the 300 test cases.

We present a GP regression model which can deal with input-dependent residuals. This model includes a latent variable as an unobserved input covariate with a fixed distribution. When the partial derivative of the response with respect to this unobserved covariate changes across observations, the variance of the residuals will change. We call this model the "Gaussian Process with a Latent Covariate" (GPLC) regression model. In the motorcycle example, our GPLC model (blue) is able to give different estimates of variance according to the observed data, in contrast to the standard GP model (red), whose predictive variances do not vary greatly among different times. The Bayesian treatment of GP regression typically requires using Markov Chain Monte Carlo methods to obtain posterior samples of the hyperparameters, as the posterior distributions are usually not analytically tractable. Alternatively, one can simply find a point estimate of the hyperparameters (e.g. a maximum *a posteriori* (MAP) estimate), or, to take a non-Bayesian approach, find an estimate of the parameters (e.g. a maximum likelihood estimate (MLE)) using the likelihood function without specifying a prior. Finding MAP and MLE may take less computation, but the estimate can be a local maximum and therefore not very accurate. Also, a point estimate doesn't capture uncertainty. We only consider the "full" Bayesian approach in this work, i.e. the method which involves using MCMC to obtain posterior samples of the hyperparameters.

Despite their long history, simplicity and flexibility, GP models haven't been widely used until recent years. One reason for this is the intensive computation required, mainly due to the n^3 operation of inverting the covariance matrix.

Several fast but approximate methods for Gaussian Process models have been developed. We show in this work how such approximations to the posterior distribution for parameters of the covariance function in a Gaussian process model can be used to speed up sampling, using either of two schemes, based on "mapping to a discretizing chain" or "mapping with tempered transitions", following the general scheme of constructing efficient MCMC methods using temporary mapping and caching techniques due to Neal

(2006). Both schemes produce an exactly correct MCMC method, despite using an approximation to the posterior density for some operations.

1.2 The standard GP regression model

Standard GP regression models are widely used for non-linear regression problems in various fields, including machine learning, finance, engineering, and statistics. We give a brief introduction below. Detailed discussion can be found in Rasmussen and Williams (2006), Neal (1998), and Bishop (2007).

In a non-linear regression problem, the aim is to find the relationship between a vector of covariates x of length p and a scalar response y, using n observed pairs $(x_1, y_1), ..., (x_n, y_n)$, and then make predictions for $y_{n+1}, y_{n+2}, ...$ corresponding to $x_{n+1}, x_{n+2}....$ We can express this relationship in terms of a regression function, f, and random residuals, ϵ_i:

$$y_i = f(x_i) + \epsilon_i \tag{1.1}$$

In the standard Gaussian process regression model, the random residuals, ϵ_i, are assumed to have i.i.d. Gaussian distributions with mean 0 and a constant variance σ^2.

Bayesian GP models assume that the noise-free regression function f comes from a Gaussian Process which has prior mean function zero and some specified covariance function. Note that a zero mean prior is not a requirement — we could specify a non-zero prior mean function $m(x)$ if we have *a priori* knowledge of the mean structure. Using a zero mean prior just reflects our prior knowledge that the function is equally likely to be positive or negative. It doesn't mean we believe the actual function will have an average over its domain of zero.

The covariance function could be fixed *a priori*, but more commonly is specified in terms of unknown hyperparameters, θ, which are then estimated from the data. Given the values of the hyperparameters, the vector of responses, y, follows a multivariate

Gaussian distribution with zero mean and a covariance matrix given by

$$\text{Cov}(y_i, y_j) = K(x_i, x_j) + \delta_{ij}\sigma^2 \tag{1.2}$$

where $K(x_i, x_j)$ is the covariance function of f, so that $\text{Cov}(f(x_i), f(x_j)) = K(x_i, x_j)$, and $\delta_{ij} = 0$ when $i \neq j$, with $\delta_{ii} = 1$. Any function K that always leads to a positive semi-definite covariance matrix can be used as a covariance function. One example is the squared exponential covariance function with isotropic length-scale:

$$K(x_i, x_j) = c^2 + \eta^2 \exp\left(-\frac{\|x_i - x_j\|^2}{\rho^2}\right) \tag{1.3}$$

Here, c is a fairly large constant (not excessively large, to avoid numerical singularity). η, ρ, and σ (from (1.2)) are hyperparameters — η controls the magnitude of variation of f, ρ is a length scale parameter for the covariates, and σ is the residual standard deviation. We can instead assign a different length scale to each covariate, which leads to the squared exponential covariance function with automatic relevance determination (ARD):

$$K(x_i, x_j) = c^2 + \eta^2 \exp\left(-\sum_{k=1}^{p} \frac{(x_{ik} - x_{jk})^2}{\rho_k^2}\right) \tag{1.4}$$

We will use the squared exponential form of covariance function from (1.3) or (1.4) in most of this work.

When the values of the hyperparameters are known, the predictive distribution of y_* for a test case x_* based on observed values $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ is Gaussian with the following mean and variance (Rasmussen and Williams, 2006):

$$E(y_*|x, y, x_*) = k^T C^{-1} y \tag{1.5}$$

$$\text{Var}(y_*|x, y, x_*) = v - k^T C^{-1} k \tag{1.6}$$

In the equations above, k is the vector of covariances between y_* and each of $y_1, \ldots, y_n$, C is the $n \times n$ covariance matrix of the observed y, and v is the prior variance of y_*, which is $\mathrm{Cov}(y_*, y_*) = K(x_*, x_*) + \sigma^2$ from (1.2).

When the values of the hyperparameters (denoted as θ) are unknown and therefore have to be estimated from the data, we put priors on them (typically independent Gaussian priors on the logarithm of each hyperparameter), and obtain the posterior distribution $p(\theta|x, y) \propto \mathcal{N}(y|0, C(\theta))p(\theta)$. The predictive mean of y_* can then be computed by integrating over the posterior distribution of the hyperparameters:

$$E(y_*|x, y, x_*) = \int_\Theta k(\theta)^T C(\theta)^{-1} y \cdot p(\theta|x, y) d\theta \tag{1.7}$$

Letting $\mathcal{E} = E(y_*|x, y, x_*)$, the predictive variance is given by

$$\begin{aligned} \mathrm{Var}(y_*|x, y, x_*) &= E_\theta[\mathrm{Var}(y_*|x, y, x_*, \theta)] + \mathrm{Var}_\theta[E(y_*|x, y, x_*, \theta)] \qquad (1.8) \\ &= \int_\Theta \left[v(\theta) - k(\theta)^T C(\theta)^{-1} k(\theta)\right] p(\theta|x, y) d\theta \\ &\quad + \int_\Theta \left[k(\theta)^T C(\theta)^{-1} y - \mathcal{E}\right]^2 p(\theta|x, y) d\theta \qquad (1.9) \end{aligned}$$

Finding C^{-1} directly takes time proportional to n^3, but we do not have to find the inverse of C explicitly. Instead we find the Cholesky decomposition of C, denoted as $R = \mathrm{chol}(C)$, for which $R^T R = C$ and R is an "upper" triangular matrix (also called a "right" triangular matrix). This also takes time proportional to n^3, but with a much smaller constant factor. We then solve $R^T u = y$ for u using a series of forward subsititutions (taking time proportional to n^2). From R and u, we can compute the likelihood for θ, which is needed to compute the posterior density, by making use of the expressions

$$y^T C^{-1} y = y^T (R^T R)^{-1} y = y^T R^{-1} \left(R^T\right)^{-1} y = u^T u \tag{1.10}$$

and

$$\det(C) = \det(R)^2 = \prod_{i=1}^{n} r_{ii}^2 \tag{1.11}$$

where r_{ii} is the ith diagonal element on R. Similarly, equations (1.5) and (1.6) can be reformulated to use R rather than C^{-1}.

1.3 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo methods are a class of algorithms for sampling from probability distributions (usually ones that are difficult to sample from using other methods) based on constructing Markov Chains that are defined such that their equilibrium distribution is the desired probability distribution. MCMC methods are extensively used in fields that require scientific computation, such as Bayesian inference.

Neal (1993), Casella and Robert (2005), Liu, J. (2009), as well as many other textbooks provide detailed discussions on this topic. We only give a very brief introduction below.

Suppose we wish to draw samples from a target distribution $\pi(x)$. We can construct a Markov Chain with transition probabilities $T(x'|x)$, such that

- This transition leaves π invariant. In other words,

$$\int \pi(x)T(x'|x)dx = \pi(x') \tag{1.12}$$

- This transition is irreducible, meaning that it is possible to get to any state from any state.

- This transition is aperiodic, meaning that the chain does not explore the state space in a cyclic way.

If all the above conditions are met, then after the chain is run for a sufficiently long time,

the distribution of the states x will be close to the target distribution $\pi(x)$, also called the "equilibirum" distribution.

We can easily construct such a chain using the Metropolis algorithm (Metropolis *et al.*, 1953). Let $x^{(k)}$ be the state of the kth time step of a Markov chain, then

- Starting from $x^{(k)}$, propose to move to x^* according to some proposal density $q(x^*|x)$ which satisfies

  $$q(x^*|x) = q(x|x^*)$$

  i.e. $q(x^*|x)$ must be symmetric.

- Accept x^* with probability

  $$A(x^*|x^{(k)}) = \min\left(1, \frac{\pi(x^*)}{\pi(x^{(k)})}\right) \tag{1.13}$$

  by drawing a sample U from Unif(0,1) and setting

  $$x^{(k+1)} = \begin{cases} x^* & \text{if } U \leq A(x^*|x^{(k)}) \\ x^{(k)} & \text{if } U > A(x^*|x^{(k)}) \end{cases}$$

It's easy to show that a Markov chain constructed by the Metropolis algorithm satisfies the "detailed balance" condition, which is that for all x and x',

$$\pi(x)T(x'|x) = \pi(x')T(x|x') \tag{1.14}$$

Starting from x, if we accept a proposal $x' = x^*$, then

$$\begin{aligned}
\pi(x)T(x'|x) &= \pi(x)q(x^*|x)A(x^*|x) \\
&= \pi(x)q(x^*|x)\min\left(1, \frac{\pi(x^*)}{\pi(x)}\right) \\
&= q(x^*|x)\min\left(\pi(x), \pi(x^*)\right) \\
&= \pi(x^*)q(x|x^*)\min\left(1, \frac{\pi(x)}{\pi(x^*)}\right) \\
&= \pi(x^*)q(x|x^*)A(x|x^*) \\
&= \pi(x')T(x|x')
\end{aligned}$$

It's trivial to show that (1.14) holds when $x' = x$, i.e. when the proposal is rejected.

A Markov chain that satisfies the detailed balance condition is called a reversible Markov chain. Reversibility implies invariance:

$$\begin{aligned}
\int \pi(x)T(x'|x)dx &= \int \pi(x')T(x|x')dx \\
&= \pi(x')\int T(x|x')dx \\
&= \pi(x')
\end{aligned}$$

Whether a Metropolis Markov chain is irreducible and aperiodic depends on π and q.

If we wish to use a non-symmetric proposal density $q(x'|x)$, then we should accept the proposal x^* with the following probability:

$$A(x^*|x) = \min\left(1, \frac{\pi(x^*)q(x|x^*)}{\pi(x)q(x^*|x)}\right) \tag{1.15}$$

This generalization of the Metropolis algorithm is called the Metropolis-Hastings algorithm (Hastings, 1970).

One way to sample from a univariate distribution with density function $\pi(x)$ is to uniformly draw points $(x_1, y_1), ..., (x_n, y_n)$ from a rectangle that covers the support of x

and the range of $\pi(x)$, then discard the points where $y_i > \pi(x_i)$. The remaining x values will then form a random sample of $\pi(x)$. However, there are two major drawbacks of this method: first, the density function $\pi(x)$ may not be easily bounded (especially if it is not normalized) and therefore the rectangle cannot be easily defined; second, when the support of x is $(-\infty, \infty)$, this method is not applicable.

Slice sampling (Neal, 1997) is a way to construct a Markov chain that will "sample from under the density curve". Given a univariate distribution with density $\pi(x) \propto \rho(x)$, with $\rho(x)$ being the unnormalized density, the algorithm operates this way:

- Given a current point, $x^{(k)} = x_0$, we draw a height, y_0, uniformly from (0,$\rho(x_0)$).
- From (x_0, y_0), we attempt to "step out" of the density curve with interval

$$(x_0 - wu, x_0 + w(1-u))$$

  where u is drawn uniformly from (0,1), and w is a pre-set length.

- If both ends of this interval have stepped out of the density curve (i.e., both $\rho(x_0 - wu)$ and $\rho(x_0 + w(1-u))$ are less than y_0), then we stop. Otherwise, we step outwards from the ends by a distance w at each step until both ends are outside. Call this interval (L, R).
- Draw x^* uniformly from (L, R). If x^* is under the density curve, i.e. $\rho(x^*) \geq y_0$ then x^* is our new point. Otherwise, we narrow the interval by setting either L or R to x^*, keeping x_0 inside the interval, and then repeat this step.

The detailed balance condition holds for this algorithm. For multivariate distributions, we can simply update one variate at a time. Figure 1.2 illustrates this procedure.

Both the Metropolis sampler and the slice sampler have "tuning" parameters. For Metropolis, the parameters for the proposal distribution can be adjusted (for instance, the standard deviations for a Gaussian proposal). For slice sampling, the step-out length,

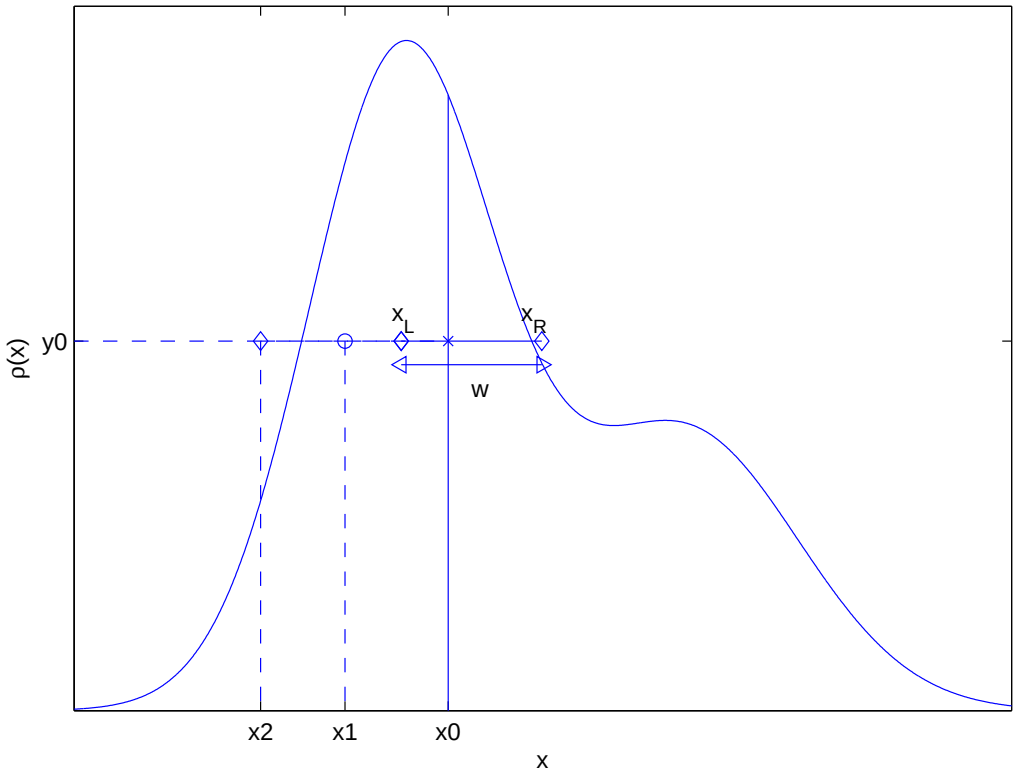

Figure 1.2: Univariate step-out slice sampling. Starting at x_0, a first interval ($x_L = x_0 - wu, x_R = x_0 + w(1-u)$) is formed. x_R steps out of the curve, but x_L remains "in" the curve. The interval then gets extended to the left by a length of w. Now $x_2 = x_0 - wu - w$ steps "out" of the curve. x_1 is uniformly drawn from this interval. Since it is under the density curve, it becomes the next state.

w, as well as the maximum number of step-outs can be tuned. The values of these tuning parameters can affect the performance of the Markov chain. However, there is usually no good way to find the optimal values of these tuning parameters other than trial and error.

The Metropolis sampler is generally hard to tune. If we update one parameter at a time based on a univariate proposal distribution, to construct an efficient MCMC method, we have to assign an appropriate value for the proposal standard deviation for each hyperparameter or latent variable so that the acceptance rate on each variable is neither too big nor too small (generally, between 20% and 80% is acceptable, though the optimal value is typically unknown). There is generally no good way to find out what tuning parameter value is the best for each variable other than trial and error. For high-dimensional problems, tuning the chain is very difficult.

Slice sampling is relatively easier to tune. It does also have tuning parameters, but the performance of the chain is not very sensitive to the tuning parameters. Figure 2.7 of Thompson (2011) demonstrates that step-out sizes from 1 to 1000 all lead to similar computation time, while a change in proposal standard deviation from 1 to 1000 for a Metropolis sampler can result in a MCMC which is 10 to 100 times slower.

1.4 Measuring and Comparing the efficiency of samplers

The efficiency of a MCMC sampler is usually measured by the autocorrelation time τ of its chain (see Neal, 1993):

$$\tau = 1 + 2\sum_{i=1}^{\infty} \rho_i \tag{1.16}$$

where ρ_i is the lag-i autocorrelation for some function of interest. Roughly speaking, the autocorrelation time can be viewed as the number of steps of a Markov chain we need to

simulate in order to obtain an independent sample. In practice, with an MCMC sample of size M, we can only find estimates, $\hat{\rho}_i$, of autocorrelations up to lag $i = M - 1$. To avoid excessive variance from summing many noisy estimates, we typically estimate τ by

$$\hat{\tau} \;=\; 1 + 2\sum_{i=1}^{k} \hat{\rho}_i \tag{1.17}$$

where k is a point where for all $i > k$, $\hat{\rho}_i$ is not significantly different from 0.

To compare the efficiency of two different samplers (that are used to sample from the same target distribution), we have to adjust for the possibility that an iteration (Markov chain step) of sampler A takes a different amount of time than that of sampler B. Time per iteration can be judged by counting the number of evaluations of the probability density of the target distribution, as it is often the dominating factor of MCMC computation. However, sometimes auxiliary operations can take a significant portion of time. For example, our MCMC with temporary mapping methods use another distribution, π^*, to help create the temporary discretizing chain. Even though the computation of $\pi^*(x)$ is typically much faster than that of $\pi(x)$, it cannot be ignored. In this paper, we will compare methods with respect to autocorrelation time of the log likelihood. For a fair comparison, we multiply the estimate of each method's autocorrelation time by the average CPU time it needs to obtain a new sample point.

1.5 Outline of the Remainder of the Thesis

We propose a Gaussian Process regression model which can deal with heteroscedastic residuals in Chapter 2. We also discuss in detail a related model by Goldberg *et al.* (1998), and show how full Bayesian inference can be implemented for this model. In Chapter 3, we discuss a general Markov Chain Monte Carlo framework using temporary mapping and caching, and present two implementations of GP regression under this framework. We give the conclusion in the last chapter, and discuss possibilities for future work.

Chapter 2

Heteroscedastic GP Regression Models

Standard non-linear regression models (including the standard GP regression models we discussed in Chapter 1) typically assume that the variance of the residuals is constant across all observations. Many of these models also assume that the distribution of the residuals is Gaussian. However, these assumptions are not always realistic in practice (Silverman's motorcycle data in Chapter 1 is a good example of this). In this chapter, we present a GP regression model which not only addresses the non-constant variance issue, but also doesn't make assumptions on the distribution of the residuals. We compare our model with that of Goldberg *et al.* (1998), which addresses only non-constant variance, and extend their earlier work to a full Bayesian version.[1]

[1]Part of this chapter originally appeared in Wang and Neal (2012).

2.1 A GP regression model with a latent covariate

In this work, we consider adding a latent variable, w, into the model as an unobserved input. The regression equation then becomes

$$y_i = g(x_i, w_i) + \zeta_i. \tag{2.1}$$

where $\zeta_i \sim N(0, \sigma^2)$

In this setting, the latent value w_i has some known random distribution, the same for all i. If we view $g(x_i, w_i)$ as a function of x_i only, its value is random, due to the randomness of w_i. So g is not the regression function giving the expected value of y for a given value of x — that is given by the average value of g over all w, which we write as $f(x)$:

$$f(x) = E(y|x) = \int g(x, w)p(w)dw \tag{2.2}$$

where $p(w)$ is the probability density of w. Note that (2.2) implies that the term ζ_i, which we assume has i.i.d. Gaussian distribution with constant variance, is not the real residual of the regression, since

$$\zeta_i = y_i - g(x_i, w_i) \;\neq\; y_i - f(x_i) = \epsilon_i$$

where ϵ_i is the true residual.

We could have omitted ζ_i, letting w_i express all randomness in y_i. We put ζ_i in the regression for two reasons. First, the covariance function for g can sometimes produce nearly singular covariance matrices, which are computationally non-invertible because of round-off error. Adding a small diagonal term can avoid this computational issue without significantly changing the properties of the covariance matrix. Secondly, the function g will produce a probability density function for ϵ that has singularities at points where the derivative of g with respect to w is zero, which is probably not desired in most

applications. Adding a jitter term ζ_i smooths away such singularities.

We will model $g(x, w)$ using a GP with a squared exponential covariance function using ARD, for which the covariance between training cases i and j, with latent values w_i and w_j, is

$$\begin{aligned} \mathrm{Cov}(y_i, y_j) &= K((x_i, w_i), (x_j, w_j)) + \sigma^2 \delta_{ij} \\ &= c^2 + \eta^2 \exp\left(- \sum_{k=1}^{p} \frac{(x_{ik} - x_{jk})^2}{\rho_k^2} - \frac{(w_i - w_j)^2}{\rho_{p+1}^2} \right) + \sigma^2 \delta_{ij} \end{aligned} \tag{2.3}$$

We choose independent standard normals as the distributions for $w_1, ..., w_n$. The mean for the w_i is chosen to be zero, but since the squared exponentianl covariance function depends on w only through $|w_i - w_j|$, any mean would work just as well. The variance of the w_i is fixed at 1 because the effect of a change of scale of w_i can be achieved instead by a change in the length scale parameter ρ_{p+1}.

We write $p(w)$ for the density for the vector of latent variables w, and $p(\theta)$ for the prior density of all the hyperparameters (denoted as a vector θ). The posterior joint density for the latent variables and the hyperparameters is

$$p(w, \theta | x, y) \propto \mathcal{N}(y | 0, C(\theta, w)) p(w) p(\theta) \tag{2.4}$$

where $\mathcal{N}(a | \mu, \Sigma)$ denotes the probability density of a multivariate Gaussian distribution with mean μ and covariance matrix Σ, evaluated at a. $C(\theta, w)$ is the covariance matrix of y, which depends on θ and w (as well as the covariates).

The prediction formulas for GPLC models are similar to (1.7) and (1.8), except that in addition to averaging over the hyperparameters, we also have to average over the posterior distribution of the latent variables in the observed cases, $w = (w_1, ..., w_n)$, and the latent variable in the new case, w_*:

$$E(y_*|x,y,x_*) = \int_{\mathcal{W}_*}\int_{\mathcal{W}}\int_{\Theta} k(\theta,w,w_*)^T C(\theta,w)^{-1} y\, p(\theta,w|x,y)p(w_*)d\theta dw dw_* \tag{2.5}$$

$$\begin{aligned}\mathrm{Var}(y_*|x,y,x_*) &= E_{\theta,w}[\mathrm{Var}(y_*|x,y,x_*,\theta,w)] + \mathrm{Var}_{\theta,w}[E(y_*|x,y,x_*,\theta,w)] \\ &= \int_{\mathcal{W}_*}\int_{\mathcal{W}}\int_{\Theta} \left[v(\theta,w_*) - k(\theta,w,w_*)^T C(\theta,w)^{-1} k(\theta,w,w_*)\right] p(w,\theta|x,y)p(w_*)d\theta dw dw_* \\ &\quad + \int_{\mathcal{W}_*}\int_{\mathcal{W}}\int_{\Theta} \left[k(\theta,w,w_*)^T C(\theta,w)^{-1} y - \mathcal{E}\right]^2 p(w,\theta|x,y)p(w_*)d\theta dw dw_*\end{aligned} \tag{2.6}$$

where $\mathcal{E} = E(y_*|x,y,x_*)$

Note that the vector of covariances of the response in a test case with the responses in training cases, written as $k(\theta,w,w_*)$ in (2.5) and (2.6), depends on, w_*, the latent value for the test case. Since we do not observe w_*, we randomly draw values from the prior distribution of w_*, compute the corresponding expectation or variance and take the average. Similarly, the prior variance for the response in a test case, written $v(\theta,w_*)$ above, depends in general on w_* (though not for the squared exponential covariance function that we use in this work).

To see that this model allows residual variances to depend on x, and that the residuals can have non-Gaussian distributions, we compute the Taylor-expansion of $g(x,w)$ at $w = 0$:

$$g(x,w) = g(x,0) + g_2'(x,0)w + \frac{w^2}{2}g_2''(x,0) + ... \tag{2.7}$$

where g_2' and g_2'' denotes the first and second order partial derivatives of g with respect to its second argument (w). If we can ignore the second and higher order terms, i.e. the linear approximation is good enough, then the response given x is Gaussian, and

$$\mathrm{Var}[g(x,w)] \approx 0 + [g_2'(x,0)]^2\mathrm{Var}(w) = [g_2'(x,0)]^2 \tag{2.8}$$

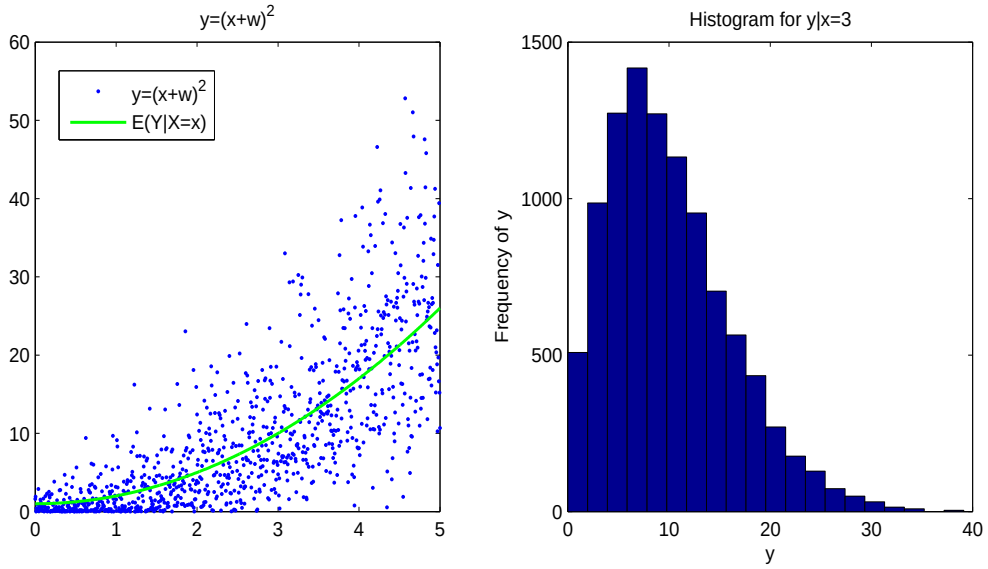

Figure 2.1: How GPLC can produce a non-Gaussian distribution of residuals: Since We do not observe w, the association between x and y is represented by the thick curve $E[(x+w)^2|x] = x^2 + 1$. One can clearly see that y is non-symmetrical with respect to its mean (therefore non-Guassian). Actually y has a Non-central Chi-squared distribution (Sankaran, 1959)

.

which depends on x when $g_2'(x, 0)$ depends on x (which typically is the case when g is drawn from a GP prior). Thus in this case, the model produces Gaussian residuals with input-dependent variances.

If the high-order terms in (2.7) cannot be ignored, then the model will have non-Gaussian, input-dependent residuals. For example, consider $g(x, w) = (x + w)^2$, where the second order term in w clearly cannot be ignored. Conditional on x, $g(x, w)$ follows a non-central Chi-Squared distribution. Figure 2.1 illustrates that at $x = 3$, an unobserved normally distributed input w (with zero mean and standard deviation 1) translates into a non-Gaussian output y.

Figure 2.2 illustrates how an unobserved covariate can produce heteroscedasticity. The data in the left plot are generated from a GP, with x_i drawn uniformly from [0,5] and w_i drawn from $N(0, 1)$. The hyperparameters of the squared exponential covariance function were set to $\eta = 3$, $\rho_x = 0.8$, and $\rho_w = 3$. Supposing we only observe (x, y), the

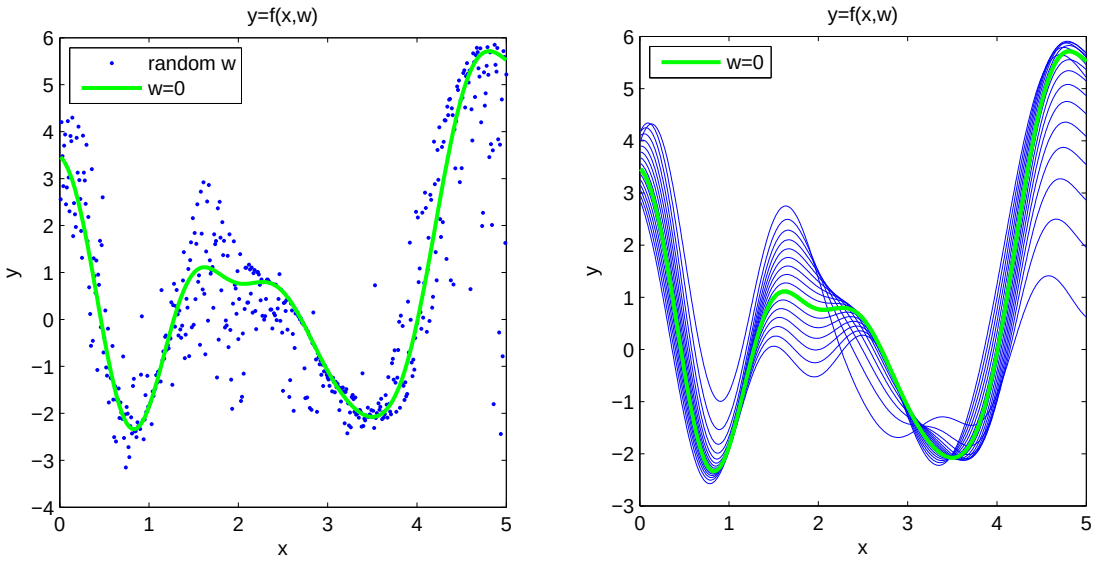

Figure 2.2: Heteroscedasticity produced by an unobserved covariate. The left plot shows a sample of x and y from the GP prior, with w not shown. The right plot shows 19 thin curves of $g(x_i, w_i)$ (for the same g as on the left) where, for the jth curve, the w_i are fixed to the $5j$th percentile of the standard normal.

data is clearly heteroscedastic, since the spread of y against x changes when x changes. For instance, the spread of y looks much bigger when x is around 1.8 than it is when x is near 3.5. We also notice that the distribution of the residuals can't be Gaussian, as, for instance, we see strong skewness near $x = 5$. These plots show that if an important input quantity is not observed, the function values based only on the observed inputs will in general be heteroscedastic, and non-Gaussian (even if the noise term ζ_i is Gaussian). Note that although an unobserved input quantity will create heteroscedasticity, our model can work well even if no such quantity really exists. The model can be seen as just using the latent variable as a mathematical trick, to produce changing residual variances. Whether or not there really exists an unobserved input quantity doesn't matter (though in practice, unobserved quantities often do exist).

We've been focussing on regression problems with only one output (a scalar response), though the GPLC model can be easily extended to problems with multiple outputs (vector

response) by introducing multiple latent variables, e.g. one for each output:

$$y_{ij} = g_j(x_i, w_1, ..., w_q) + \zeta_{ij},\ j = 1, ..., q \tag{2.9}$$

2.2 A GP regression model with a latent variance

Goldberg, Williams, and Bishop (1998) proposed a GP treatment of regression with input-dependent residuals. In their scheme, a "main" GP models the mean of the response just like a standard GP regression model, except the residuals are not assumed to have constant variance — a secondary GP is used to model the logarithm of the standard deviation of the residuals, which depends on the input. The regression equation looks the same as in (1.1):

$$y_i = f(x_i) + \epsilon_i \tag{2.10}$$

but the residuals $\epsilon_1, ...\epsilon_n$ do not have the same variance — instead, the logarithm of the standard deviation $z_i = \log SD[\epsilon_i]$ depends on x_i through:

$$z_i = r(x_i) + J_i \tag{2.11}$$

$f(x)$ and $r(x)$ are both given (independent) GP priors, with zero mean and covariance functions C_y and C_z, which have different hyperparameters (e.g. (η_y, ρ_y) and (η_z, ρ_z)). J_i is a Gaussian "jitter" term (see Neal, 1997) which has i.i.d. Gaussian distribution with zero mean and standard deviation σ_J (a preset constant, usually a very small number, e.g. 10^{-3}). Writing $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n)$, $\theta_y = (\eta_y, \rho_y)$, $\theta_z = (\eta_z, \rho_z)$, and $z = (z_1, ..., z_n)$, the posterior density function of the latent values and the hyperparameters is

$$\begin{aligned} p(\theta_y, \theta_z, z|x, y) &\propto p(y|x, z, \theta_y)p(z|x, \theta_z)p(\theta_y, \theta_z) \\ &\propto \mathcal{N}(y|0, C_y(\theta_y, z))\mathcal{N}(z|0, C_z(\theta_z))p(\theta_y, \theta_z) \end{aligned} \tag{2.12}$$

where C_y is the covariance matrix for y (for the "main" GP), C_z is the covariance matrix for z (for the "secondary" GP) and $p(\theta_y, \theta_z)$ represents the prior density for the hyperparameters (typically independent Gaussian priors for their logarithms). The predictive mean can be computed in a similar fashion as the prediction of GPLC, but instead of averaging over w, we average over z. To compute the covariance vector k, we need values of z_{n+1}, which we can sample from $p(z_{n+1}|z_1, ..., z_n)$.

Alternatively, instead of using a GP to model the logarithm of the residuals standard deviations, we can set the standard deviations to the absolute values of a function modelled by a GP. That is, we let $SD(\epsilon_i) = |z_i|$, with $z_i = r(x_i) + J_i$. So the regression model can be written as

$$y_i = f(x_i) + r(x_i)u_i \tag{2.13}$$

where $u_i \overset{\text{iid}}{\sim} N(0, 1)$, which is symmetrical around zero, so $r(x_i)u_i$ has the same distribution as $|r(x_i)|u_i$.

This is similar to modelling the log of the standard deviation with a GP, but it does allow the standard deviation, $|z_i|$, to be zero, whereas $\exp(z_i)$ is always positive, and it is less likely to produce extremely large values for the standard deviation of a residual. A more general approach is taken by Wilson and Ghahramani (2010), who use a parametrized function to map values modelled by a GP to residual variances, estimating the parameters from the data.

In the original paper by Goldberg *et al.*, a toy example was given where the hyperparameters are all fixed, with only the latent values sampled using MCMC. In this work, we will take a full Bayesian approach, where both the hyperparameters and the z values are sampled from (2.12). In addition, we will discuss fast computation methods for this model in Section 2.4.

2.3 Relationships between GPLC and other models

We will show in this section that GPLC can be equivalent to the a standard GP regression model or to a GPLV model, when the covariance function is suitably specified.

Suppose the function $g(x, w)$ in (2.1) has the form

$$g(x_i, w_i) = h(x_i) + \sigma w_i \tag{2.14}$$

where $w_i \sim N(0, 1)$ and here we assume $\zeta_i = 0$. If we only observe x_i but not w_i, then (2.14) is a regression model with i.i.d. Gaussian residuals with mean 0 and variance σ^2, which is equivalent to the standard GP regression model (1.1), if we give a GP prior to h. If we specify a covariance function that produces such a $g(x, w)$, then our GPLC model will be equivalent to the standard GP model. Below, we compute the covariance between training cases i and j (with latent values w_i and w_j) to find out the form of the appropriate covariance function.

Let's put a GP prior with zero mean and covariance function $K_1(x_i, x_j)$ on $h(x)$, and an independent Gaussian prior with zero mean and variance λ^2 on σ. Since the values of $g(x, w)$ are a linear combination of independent Gaussians, they will have a Gaussian process distribution, conditional on the hyperparameters defining K_1. Now given x and w, the covariance between cases i and j is

$$\begin{aligned} \mathrm{Cov}[g(x_i, w_i), g(x_j, w_j)] &= E[(h(x_i) + \sigma w_i)(h(x_j) + \sigma w_j)] \\ &= E[h(x_i)h(x_j)] + w_i w_j E(\sigma^2) \\ &= K_1(x_i, x_j) + \lambda^2 w_i w_j \end{aligned} \tag{2.15}$$

Therefore, if we put a GP prior on $g(x, w)$ with zero mean and covariance function

$$K[(x_i, w_i), (x_j, w_j)] = K_1(x_i, x_j) + \lambda^2 w_i w_j \tag{2.16}$$

the results given by GPLC will be equivalent to standard GP regression with covariance function K_1 plus residuals of unknown standard deviation having a Gaussian prior. In practice, if we are willing to make the assumption that the residuals have equal variances (or know this as a fact), this modified GPLC model is not useful, since the complexity of handling latent variables computationally is unnecessary. However, consider a more general covariance function

$$K[(x_i, w_i), (x_j, w_j)] = K_1[(x_i, w_i), (x_j, w_j)] + K_2[(x_i, w_i), (x_j, w_j)] \tag{2.17}$$

where $K_1[(x_i, w_i), (x_j, w_j)] = \exp\left(-\sum_{k=1}^{p}(x_{ik} - x_{jk})^2/\rho_k^2 - (w_i - w_j)^2/\rho_{p+1}^2\right)$ is a squared exponential covariance function with ARD, and $K_2[(x_i, w_i), (x_j, w_j)] = \sum_{k=1}^{p} \gamma_k^2 x_{ik} x_{jk} + \gamma_{p+1}^2 w_i w_j$ is a linear covariance function with ARD. Then the covariance function (2.16) can be obtained as a limiting case of (2.17), when ρ_{p+1} goes to infinity in K_1 and $\gamma_1, ..., \gamma_p$ all go to zero. Therefore, we could use this more general model, and let the data choose whether to (nearly) fit the simpler standard GP model.

Similarly, if we believe that the function $g(x, w)$ is of the form

$$g(x, w) = h_1(x) + w h_2(x) \tag{2.18}$$

then with h_1 and h_2 independently having Gaussian Process priors with zero mean and covariance functions K_1 and K_2, conditional on x and w, the covariance between case i and case j is

$$\begin{aligned} \mathrm{Cov}[g(x_i, w_i), g(x_j, w_j)] &= E[(h_1(x_i) + w_i h_2(x_i))(h_1(x_j) + w_j h_2(x_j))] \\ &= E[h_1(x_i) h_1(x_j)] + w_i w_j E[(h_2(x_i) h_2(x_j)] \\ &= K_1(x_i, x_j) + w_i w_j K_2(x_i, x_j) \end{aligned} \tag{2.19}$$

Therefore, we can use a GPLC model with a covariance function of the form

$$K[(x_i, w_i), (x_j, w_j)] = K_1(x_i, x_j) + w_i w_j K_2(x_i, x_j) \tag{2.20}$$

to model the function in (2.18).

Now consider the GPLV model (2.13): if we put independent GP priors on $f(x_i)$ and $r(x_i)$, each with zero mean, and covariance functions K_1 and K_2, respectively, then model (2.13) is equivalent to the modified GPLC model above with covariance function (2.20). The hyperparameters of K_1 of both models should have the same posterior distribution, as would the hyperparameters of K_2. Notice that the two models have different latent variables: the absolute value of the latent variable in GPLC, $|w_i|$, is the absolute value of the ith (normalized) residual; the latent variable in GPLV is $z_i = r(x_i)$, which is plus or minus the standard deviation of the ith residual.

2.4 Computation

Bayesian inference for GP models is based on the posterior distribution of the hyperparameters and the latent variables. Unfortunately this distribution is seldom analytically tractable. We usually use Markov Chain Monte Carlo to sample the hyperparameters and the latent values from their posterior distribution.

Metropolis sampling and slice sampling are among the most commonly used MCMC algorithms. Since slice samplers are generally easier to tune than Metropolis samplers, in this work, we use univariate step-out slice sampling for regular GP regression models and GPLC models (for both the hyperparameters and the latent variables). For GPLV, since the latent values are highly correlated, regular Metropolis and slice samplers do not work well. We will give a modified Metropolis sampler that works better than both of these simpler samplers.

2.4.1 Major computations for GP models

Evaluating the log of the posterior probability density of a GP model is typically dominated by computing the covariance matrix, C, and finding the Cholesky decomposition of C, with complexities pn^2 and n^3, respectively.

For both standard GP models and GPLV models, updates of most of the hyperparameters require that the covariance matrix C be recomputed, and hence also the Cholesky decomposition (denoted as chol(C)). For GPLC, when the ith latent variable is updated, the ith row and ith column need to be updated. For this we also need to find the Cholesky decomposition of the new covariance matrix.

Things are slightly more complicated for GPLV, since the model consists of two GPs, with two covariance matrices. When one of the hyperparameters for the main GP (denoted as θ_y) is changed, the covariance matrix for the main GP, C_y, is changed, and thus chol(C_y) has to be recomputed. However, C_z, the covariance matrix for the secondary GP, remain unchanged. When one of θ_z, the hyperparameters of the secondary GP, is changed, C_y (and chol(C_y)) remain unchanged, but C_z and chol(C_z) must be recomputed. When one of the latent values, say the ith, is changed, C_z remains unchanged as it only depends on x and θ_z, but the ith entry on the diagonal of C_y is changed. This minor change to C_y requires only a rank-1 update of chol(C_y) (Sherman and Morrison, 1950), with complexity n^2. We list the major operations for the GP models discussed in this chapter in Table 2.1.

2.4.2 Silverman's motorcycle-impact dataset

In this section, we report how we fit the standard GP, the GPLC and the GPLV model for Silverman's motorcycle-impact data to produce Figure 1.1.

There are 94 observations in the dataset, which we to train a standard GP regression model and a GPLC model. For the standard GP model, we use a GP prior with zero mean and a squared exponential covariance function, setting the prior as follows:

		one hyperparameter	latent variable(s)
STD	Operation	C, chol(C)	-
	Complexity	pn^2, n^3	-
	# of such operations	$p+2$	-
GPLC	Operation	C, chol(C)	$1/n$ of C, all of chol(C)
	Complexity	pn^2, n^3	pn, n^3
	# of such operations	$p+3$	n
GPLV with Standard Metropolis/Slice	Operation	C_y, chol(C_y) or C_z, chol(C_z)	rank-1 update C_y
	Complexity	pn^2, n^3	n^2
	# of such operations	$2p+2$	n
GPLV with Modified Metropolis	Operation	C_y, chol(C_y) or C_z, chol(C_z)	C_y, chol(C_y)
	Complexity	pn^2, n^3	n^3
	# of such operations	$2p+2$	1

Table 2.1: Major operations needed when hyperparameters and latent variables change in GP models. We also include a modified Metropolis method described in Section 2.4.3. Note for each method, the first row "operation" gives the operation needed to update one hyperparameter/latent variable (for GPLV with Modified Metropolis, all latent variables). The third row "# of such operations" gives the number of the corresponding operations needed to have all the hyperparameters and latent variables updated for models with ARD covariance functions.

$$\log \eta \sim N(4, 2^2)$$

$$\log \rho \sim N(0, 2^2)$$

$$\log \sigma \sim N(0, 2^2)$$

For GPLC, we use a GP prior of the form (2.3), with the prior for the hyperparameters and latent variables being

$$\log \eta \sim N(4, 2^2)$$

$$\log \rho_x \sim N(0, 2^2)$$

$$\log \rho_w \sim N(-1, 2^2)$$

$$\log \sigma \sim N(-1, 2^2)$$

$$w_i \sim N(0, 1) \text{ for } i = 1, 2, ..., 94$$

For both standard GP and the GPLC models, we set the constant $c = 50$.

For GPLV, we use a GP prior with squared exponential covariance function, and set the priors for the hyperparameters as follows.

$$\log \eta_y \sim N(4, 2^2)$$
$$\log \rho_y \sim N(0, 2^2)$$
$$\log \eta_z \sim N(0, 2^2)$$
$$\log \rho_z \sim N(0, 2^2)$$

We use the slice sampler to obtain the posterior samples for both standard GP and the GPLC models, updating one parameter/latent variable at a time. We use the modified Metropolis sampler described in Section 2.4.3 for the GPLV model. We run the MCMC for 10000 iterations for each of the standard GP, the GPLC and the GPLV models. For all models, we trim the first 2000 obtained samples as burn-in. Using the posterior samples obtained, we then make predictions on time = 0 to 60 ms in 0.2 ms increments to produce the predictive mean and predictive variance. For each test case x_* of GPLC, we randomly generate 10 latent values of w_* from its prior, for each MCMC iteration, to make the prediction using (2.5) and (2.6). For each test case x_* of GPLV, we randomly generate 10 latent values of z_* from the conditional distribution of z_* given values of $z_1, ..., z_{94}$, for each MCMC iteration, to make the prediction. For the modified Metropolis sampler, we set $a = 0.05$, resulting in a 17% acceptance rate on the latent values of z_i.

Figures 2.3 and 2.4 show that the chains for GPLC and GPLV appear to have converged.

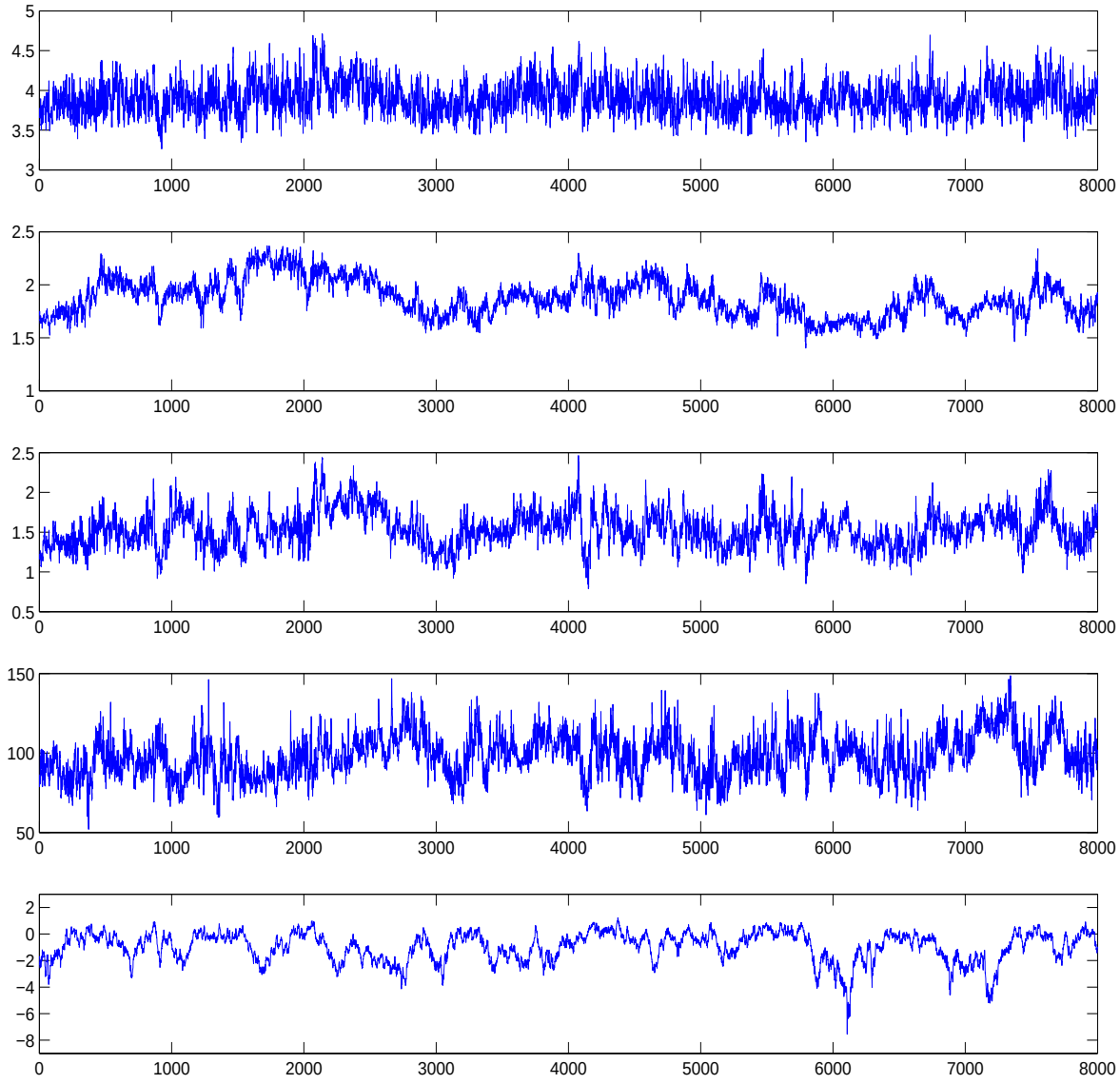

Figure 2.3: Trace plots for GPLC on the Silverman's motorcycle data. The plots (from the top to the bottom) are trace plots for $\log \eta$, $\log \rho_x$, $\log \rho_w$, $\sum_i w_i^2$ and $\log \sigma$, respectively. All the trace plots are plotted with the first 2000 iterations trimmed off.

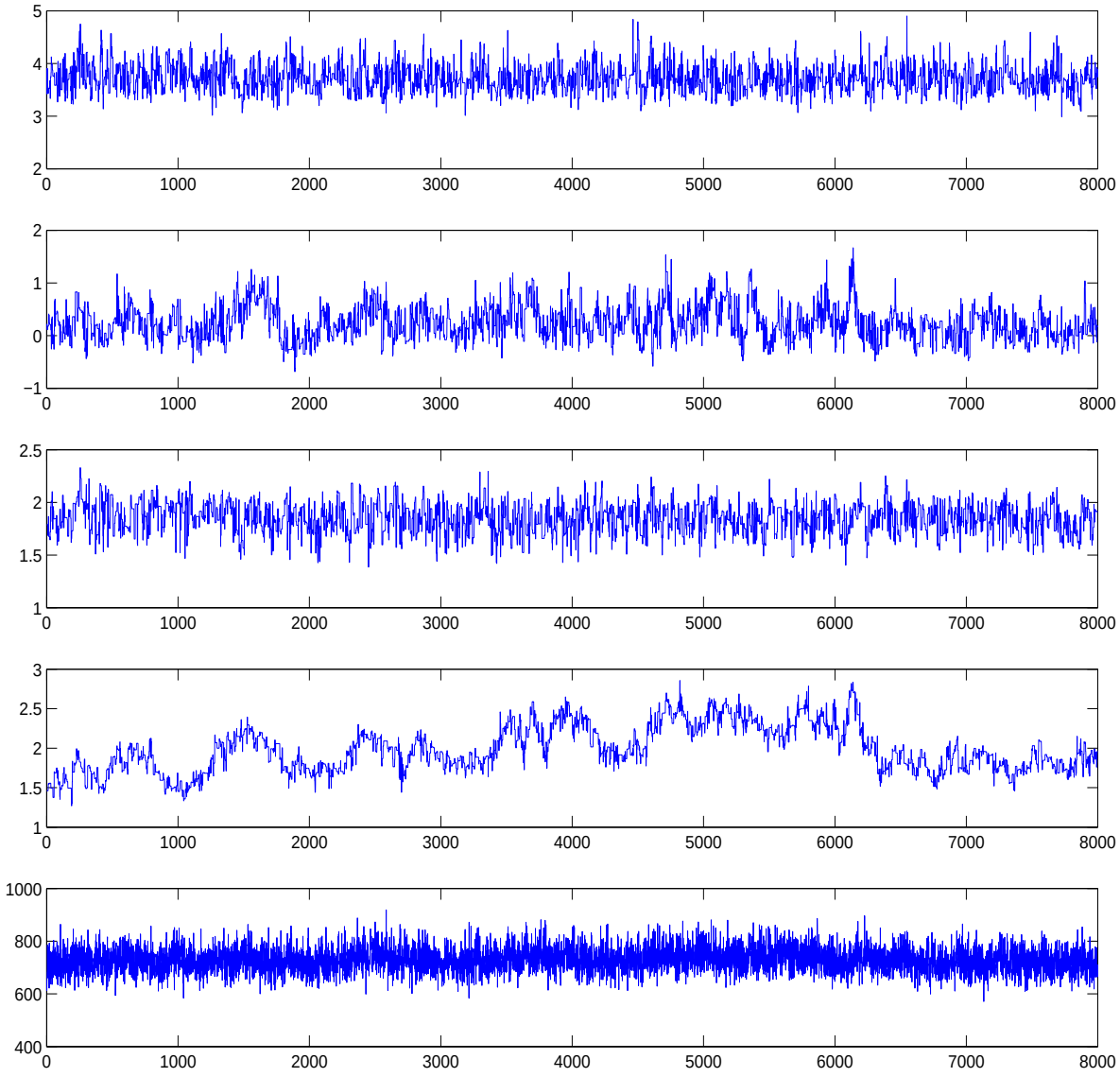

Figure 2.4: Trace plots for GPLC on the Silverman's motorcycle data. The plots (from the top to the bottom) are trace plots for $\log \eta_y, \log \rho_y, \log \eta_z, \log \rho_z$ and $\sum_i z_i^2$, respectively. All the trace plots are plotted with the first 2000 iterations trimmed off.

2.4.3 A modified Metropolis sampler for GPLV

Neal (1998) describes a method for updating latent variables in a GP model that uses a proposal distribution that takes into account the correlation information. This method proposes to change the current latent values, z, to a z' obtained by

$$z' = (1 - a^2)^{1/2} z + aR^T u \tag{2.21}$$

where a is a small constant (a tuning parameter, typically slightly greater than zero), R is the upper triangular Cholesky decomposition for C_z, the covariance matrix for the $N(0, C_z(\theta_z))$ prior for z, and u is a random vector of i.i.d. standard normal values. The transition from z to z' is reversible with respect to the $N(0, C_z(\theta_z))$ prior for z, and leaves the prior for z invariant. Because of this, the Metropolis-Hastings acceptance probability for these proposals depends only on the ratio of likelihoods for z' and z, given by $N(y|0, C_y(\theta_y, z))$. We will use this method to develop a sampling strategy for GPLV. Recall the unnormalized posterior distribution for the hyperparameters and latent values is given by

$$p(\theta_y, \theta_z, z|x, y) \propto \mathcal{N}(y|0, C_y(\theta_y, z))\mathcal{N}(z|0, C_z(\theta_z))p(\theta_y, \theta_z)$$

To obtain new values θ_y', θ_z' and z' based on current values θ_y, θ_z and z, we can do the following:

1. For each of the hyperparameters in θ_y (i.e. those associated with the "main" GP), do an update of this hyperparameter (for instance a Metropolis or slice sampling update). Notice that for each of these updates we need to recompute chol(C_y), but not chol(C_z), since C_z does not depend on θ_y.

2. For each of the hyperparameters in θ_z (i.e. those for the "secondary" GP):

   (a) Do an update of this hyperparameter (e.g. with Metropolis or slice sampling). We need to recompute chol(C_z) for this, but not chol(C_y), since C_y does not

depend on θ_z.

(b) Update all of z with the proposal described in (2.21). We need to recompute chol(C_y) to do this, but not chol(C_z), since C_z depends only on θ_z but not z. We repeat this step for m times (a tuning parameter) before moving to the next hyperparameter in θ_z.

In this scheme, the hyperparameters θ_y and θ_z are not highly correlated and hence are relatively easy to sample using the Metropolis algorithm. The latent variables z are highly correlated. The z-values are difficult to sample (since they are correlated and therefore the chain would have relatively large autocorrelation time), but each update to the z-values are relatively cheap, so we try to update them as much as possible. Notice that C_z depends only on x and θ_z, so a change of z will not result in a change of C_z. Hence once we update a component of θ_z (and obtain a new C_z), it makes sense to do $m > 1$ updates on z before updating another component of θ_z, or of θ_y.

2.5 Experiments

We will compare our GPLC model with Goldberg *et al.*'s GPLV model, and with a standard GP regression model having Gaussian residuals of constant variance.

2.5.1 Experimental setup

We use six types of synthetic datasets, with one or three covariates, and Gaussian or non-Gaussian residuals, as summarized below:

Dataset	p	True function	Residual SD	Residual distribution
U0	1	$f(x)$	0.2	Gaussian
U1	1	$f(x)$	$r(x)$	Gaussian
U2	1	$f(x)$	$r(x)$	non-Gaussian
M0	3	$g(x)$	0.3	Gaussian
M1	3	$g(x)$	$s(x)$	Gaussian
M2	3	$g(x)$	$s(x)$	non-Gaussian

Datasets U0, U1 and U2 (shown in Figure 2.5) all have one covariate, which is uniformly drawn from [0,1], and the true function is

$$f(x_i) = [1 + \sin(4x_i)]^{1.1}$$

The response $y_i = f(x_i) + \epsilon_i$ for U0 is contaminated with Gaussian residuals, ϵ_i, with constant standard deviation 0.2.

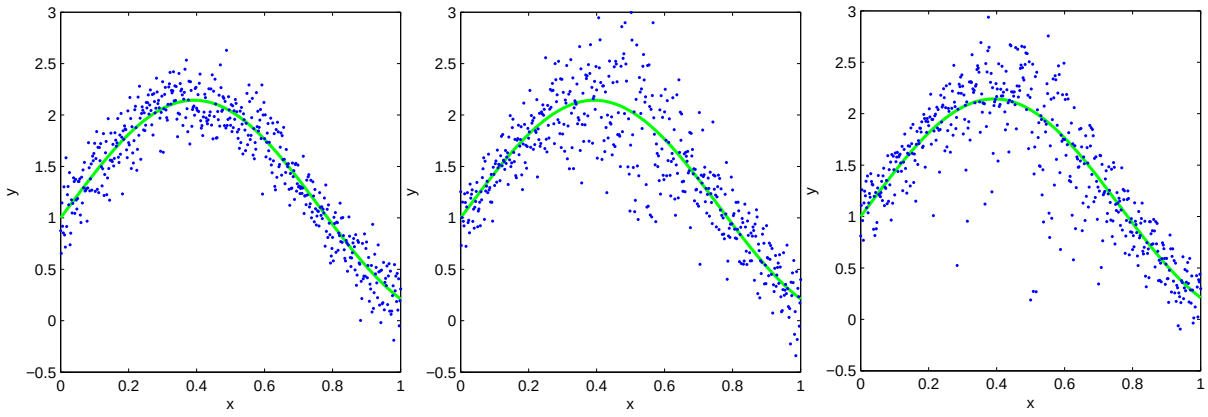

Figure 2.5: Datasets U0, U1 and U2. In each plot, we show the true function as the solid curve, as well as 500 observations with random noise. One can see that the residuals for U0 have a constant variance, whereas for U1 and U2 the variance of the residuals change according to the input. Also, the residuals for U1 have a symmetric distribution, and the residuals for U2 have a skewed distribution.

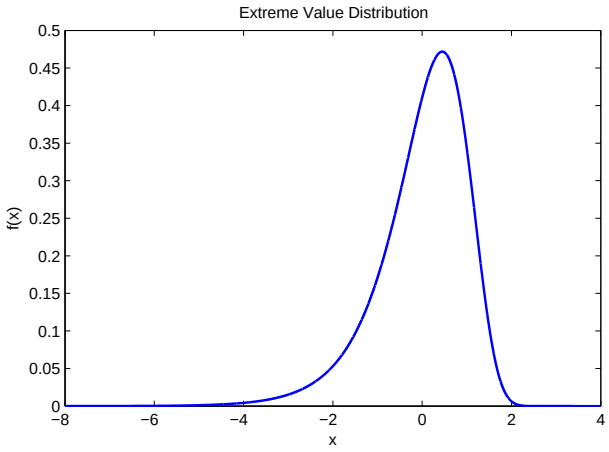

Figure 2.6: Density curve of extreme value residual with mean 0 and variance 1.

For U1, the Gaussian residuals ϵ_i have input-dependent standard deviation

$$SD(\epsilon_i) = r(x_i) = 0.2 + 0.3\exp[-30(x_i - 0.5)^2]$$

For U2, the response has a non-Gaussian residual, ω_i, with a location-scale extreme value distribution, $EV(\mu_i, \sigma_i)$ (see Leadbetter *et al.*, 2011), with probability density

$$\pi(\omega) = (1/\sigma)e^{(\omega-\mu)/\sigma}\exp\left(-e^{(\omega-\mu)/\sigma}\right).$$

The mean and variance of ω are

$$E(\omega) = \mu + \sigma\gamma,$$
$$\text{Var}(\omega) = \frac{\pi^2}{6}\sigma^2.$$

where $\gamma = 0.5772\ldots$ is Euler's constant. We set $\mu_i = -\sqrt{6}\gamma r(x_i)/\pi$ and $\sigma_i^2 = 6/\pi^2 r(x_i)^2$ so that the mean of the ω_i is zero and its standard deviation is $r(x_i)$ (same as those of ϵ in U1). The density curve of a EV residual with mean 0 and variance 1 is shown in Figure 2.6.

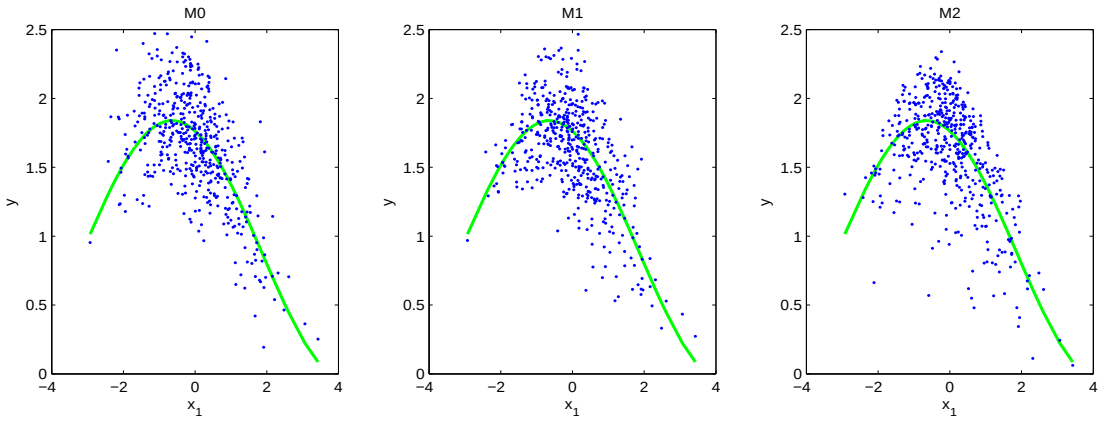

Figure 2.7: 1-D plots for datasets M1 and M2 when $x_2 = x_3 = 0$: similar to Figure 2.5, solid curves are the true functions when $x_2 = x_3 = 0$, and dots are the noisy observations.

Datasets M0, M1 and M2 all have three independent, standard normal covariates, denoted as $x = (x_1, x_2, x_3)$. The true function is

$$g(x) = [1 + \sin(x_1/1.5 + 2)]^{0.9} - [1 + \sin(x_2/2 + x_3/3 - 2)]^{1.5}$$

As we did for U0, U1 and U2, we add Gaussian residuals with constant standard deviation (0.3) to M0, input-dependent Gaussian residuals to M1 and input-dependent extreme value residuals to M2. For both M1 and M2, the standard deviations of these residuals depend on the input covariates as follows:

$$s(x) = 0.1 + 0.4\exp[-0.2(x_1 - 1)^2 - 0.3(x_2 - 2)^2] + 0.3\exp[-0.3(x_3 + 2)^2]$$

The reason why we use these particular data sets is that we would like to have some multi-dimensional datasets that are neither too simple nor too hard to work with. We illustrate datasets M1 and M2 in Figures 2.7 and 2.8.

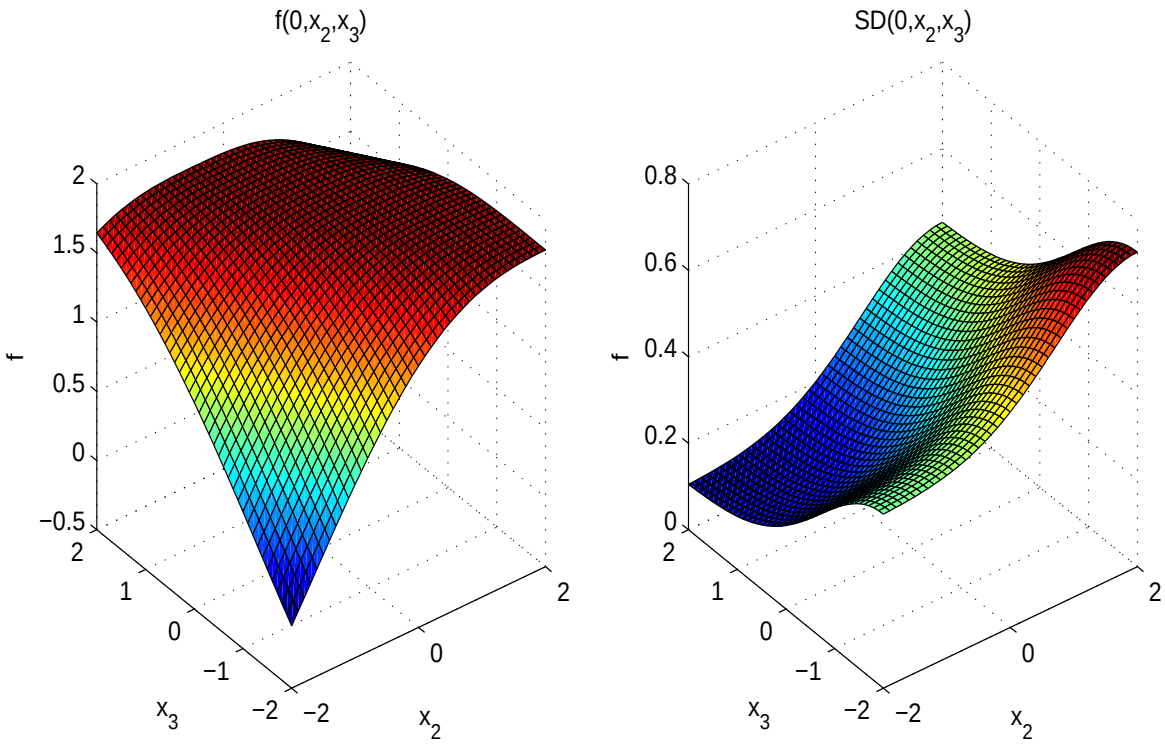

Figure 2.8: 2-D plots for datasets M1 and M2 when $x_1 = 0$: With $x_1 = 0$, the left plot shows the 2-D surface of the true function; the right plot shows how the standard deviation of the residuals change as x_2 and x_3 change.

2.5.2 Predictive performance of the methods

For each dataset (U1, U2, M1, and M2), we randomly generated 10 different training sets (using the same program but different random seeds), each with $n = 100$ observations, and a test dataset with $N = 5000$ observations. We obtained MCMC samples using the methods described in the previous section, dropping the initial 1/4 of the samples as burn-in, and used them to make predictions for the test cases. For each dataset, we run 10000 iterations of slice sampler for the standard GP model, 5000 iterations of slice sampler for the GPLC model and 3000 iterations of the modified Metropolis sampler for the GPLV model. Trace plots suggest that these number of iterations are enough for the chains to converge.

In order to evaluate how well each model does in terms of the mean of its predictive distribution, we computed the mean squared error (MSE) with respect to the true function values $f(x_i)$ as follows

$$\text{MSE}(\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{M}\sum_{j=1}^{M}\frac{1}{L}\sum_{k=1}^{L}\hat{y}_{ijk} - f(x_i)\right)^2 \tag{2.22}$$

where $\hat{y}_{ijk}$ is the predicted value for test case i based on the jth sample of hyperparameters and latent variables corresponding to the training cases and the kth latent variable corresponding to test case i (for GPLC, it's randomly sampled from its prior, for GPLV, it's sampled from $p(z_*|z_1, ..., z_n)$). M is the number of MCMC iterations after discarding the burn-in iterations, and L is the number of latent variables we randomly sample from its prior to make prediction for each test case for each MCMC iteration. We also computed the average negative log-probability density (NLPD) of the responses in the test cases, as follows

$$\text{NLPD} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{1}{M}\sum_{j=1}^{M}\frac{1}{L}\sum_{k=1}^{L}\psi(y^{(i)}|\hat{\mu}_{ijk}, \hat{\sigma}^2_{ijk})\right) \tag{2.23}$$

where $\psi(\cdot|\mu, \sigma^2)$ denotes the probability density for $N(\mu, \sigma^2)$, $\hat{\mu}_{ijk}, \hat{\sigma}^2_{ijk}$ are the predictive mean and variance for test case $y^{(i)}$ using the hyperparameters and latent variables from the jth MCMC iteration and the kth randomly sampled latent variable for test case i.

For standard GP, there is no latent variables, so the formulas for MSE and NLPD reduce to

$$\text{MSE}(\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{M}\sum_{j=1}^{M}\hat{y}_{ij} - f(x_i)\right)^2$$
$$\text{NLPD} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{1}{M}\sum_{j=1}^{M}\psi(y^{(i)}|\hat{\mu}_{ij}, \hat{\sigma}^2_{ij})\right)$$

where $\hat{y}_{ij}$ is the predicted value for test case i based on the jth sample of hyperparameters, and $\hat{\mu}_{ij}$ and $\hat{\sigma}^2_{ij}$ are the predictive mean and variance for test case $y^{(i)}$ using the hyperparameters from the jth MCMC iteration.

We give pairwise comparison of the MSE and the NLPD in Figures 2.9 through 2.14. For homoscedastic datasets U0, all three models give very similar results. For M0, the standard GP model gives slightly better results than the other two models (it gives the smallest MSE 7 out of 10 times, and the smallest NLPD 8 out of 10 times, though the differences are small). For heteroscedastic datasets (U1, U2, M1 and M2), the plots show that both GPLC and GPLV give smaller NLPD values than the standard GP model for almost all datasets with only one exception for U1 and GPLC. At least for the multivariate datasets, GPLC and GPLV also usually give smaller MSEs than the standard GP model. This shows that both GPLC and GPLV can be effective for heteroscedastic regression problems.

Comparing GPLC and GPLV on heteroscedastic datasets, we notice that for datasets with Gaussian residuals (datasets U1 and M1), GPLC gives very similar MSE and NLPD values as GPLV (GPLC is slightly worse then GPLV only for NLPD values for U1, where 7 out 10 times it gives bigger NLPD values than GPLV). For non-Gaussian residuals,

GPLC is the clear winner, giving MSEs and NLPDs that are smaller than for GPLV most of the time. The numerical MSE and NLPD values are listed in the Appendix.

We ran these experiments on a workstation with an Intel Xeon (R) E31270 CPU @ 3.40GHz. It takes around 10 minutes to run 10000 iterations of slice sampler for standard GP, about 3 to 3.5 hours to run 5000 iterations of slice sampler for GPLC, and about 15 minutes to run 3000 iterations of a modified Metropolis sampler for GPLV.

2.5.3 Comparison of MCMC methods for GPLV models

To test whether or not the modified Metropolis sampler described in Section 2.4.3 is effective, we compare it to two standard samplers, which update the latent values one by one using either the Metropolis algorithm or the univariate step-out slice sampler. The Metropolis algorithm (with Gaussian proposal, updating one parameter at a time) is used to update the hyperparameters in all of the three samplers. The significant computations are listed in Table 2.1.

We adjust the tuning parameters so that the above samplers are reasonably efficient. For the slice sampler, we use a slice width of 1, and allow infinite number of stepping-out. For the univariate Metropolis sampler, we adjust the standard deviations of the Gaussian proposals so that the acceptance rate of each parameter variable is around 50%. For the modified Metropolis sampler, we set $a = 0.3$ and $m = 40$. These tuning parameter values are found by trial and error.

The efficiency of an MCMC method is usually measured by the autocorrelation time, τ (see Section 1.4). Using the first simulated dataset for U1 as an example, we record the autocorrelation time of both the hyperparameters and the latent variables. The model has four hyperparameters (η_y, ρ_y and η_z, ρ_z), so it is not too difficult to look at all of them. But there are $n = 100$ latent variables, each with its own autocorrelation time. Instead of comparing all of them one by one, we will compare the autocorrelation time of the sum of the latent variables as well as the sum of the squares of the latent variables.

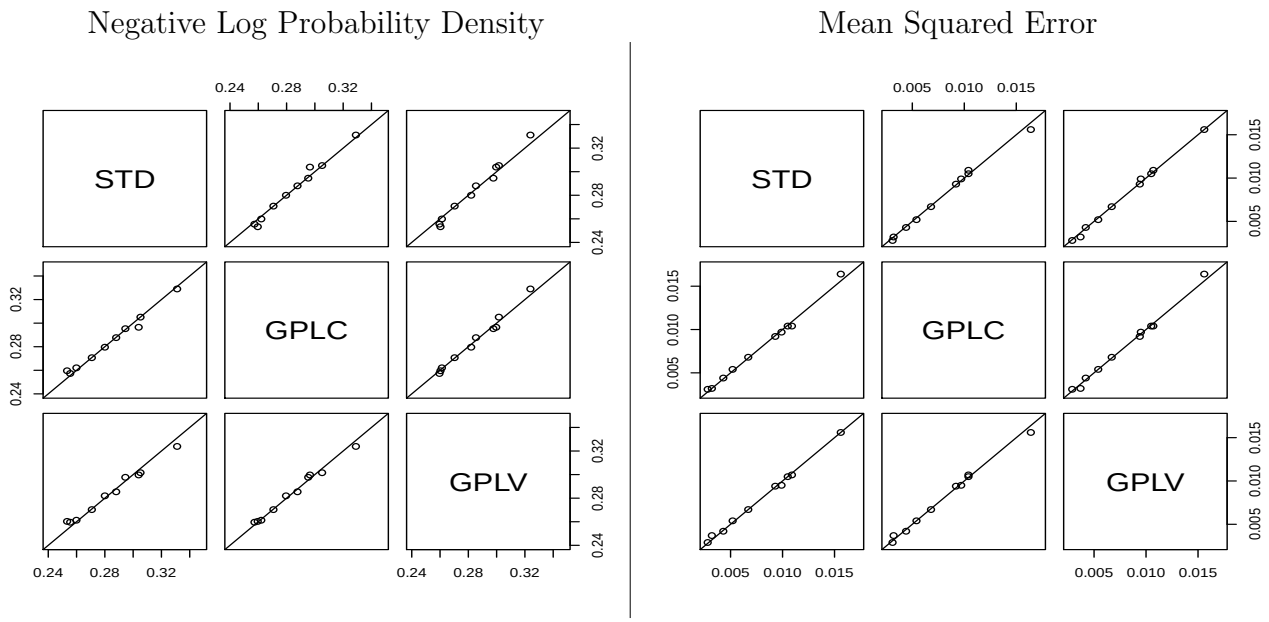

Figure 2.9: Dataset U0: Pairwise comparison of methods using NLPD(Left) and MSE(right)

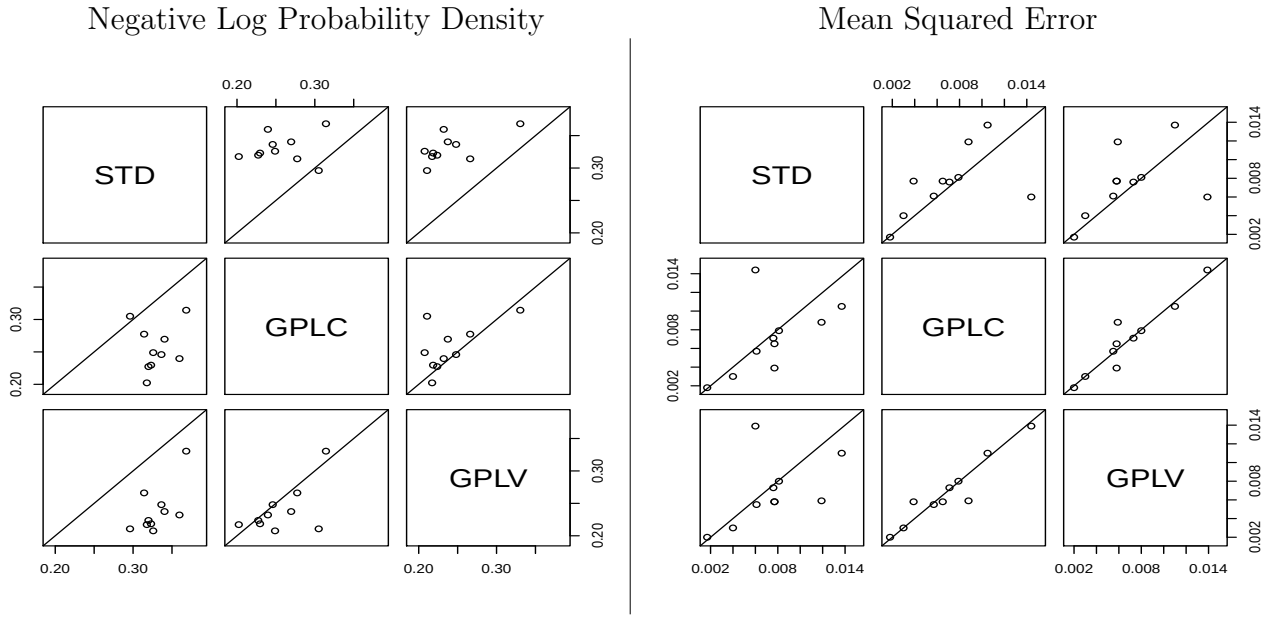

Figure 2.10: Dataset U1: Pairwise comparison of methods using NLPD(Left) and MSE(right)

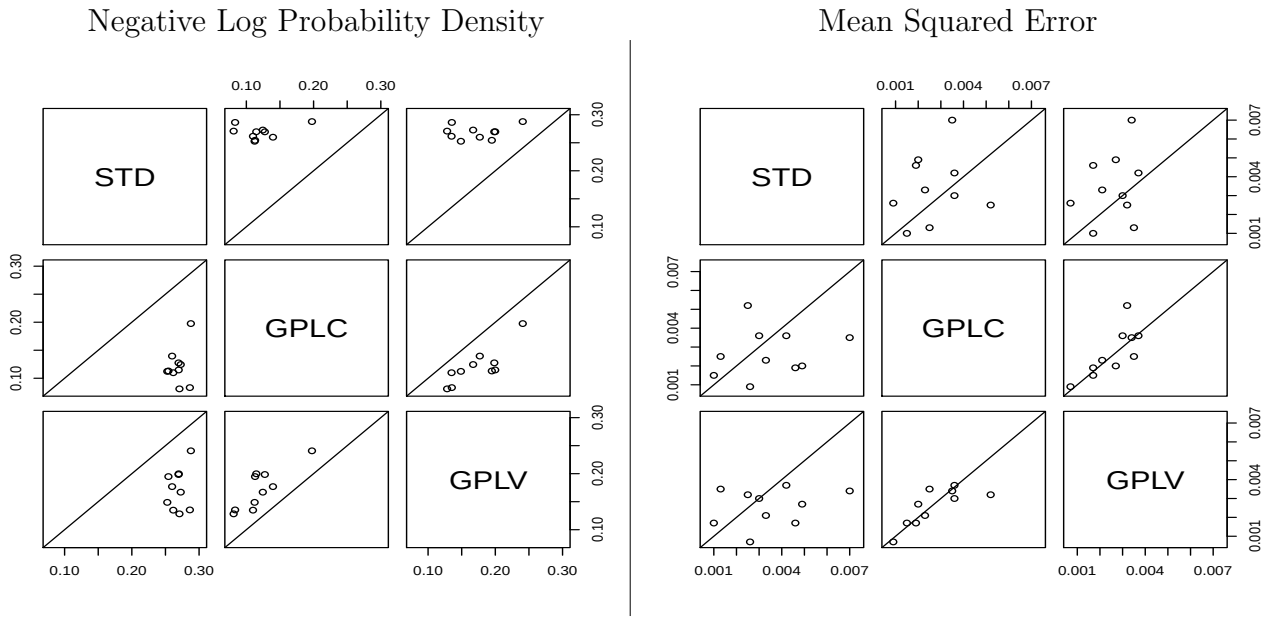

Figure 2.11: Dataset U2: Pairwise comparison of methods using NLPD(Left) and MSE(right)

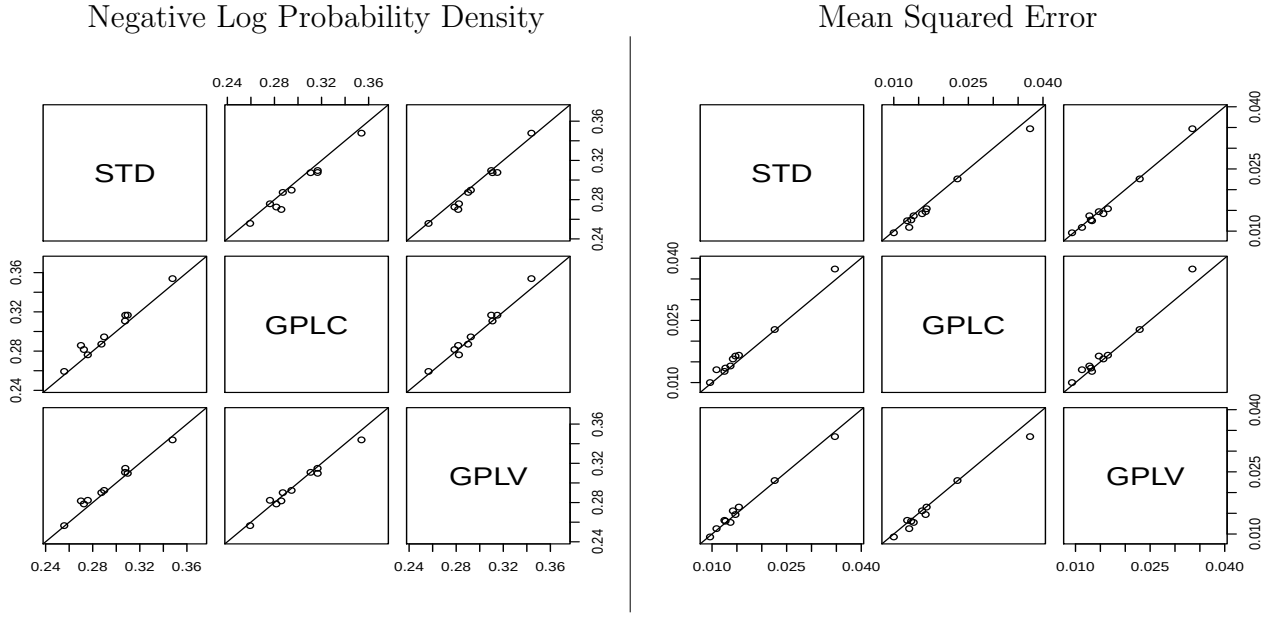

Figure 2.12: Dataset M0: Pairwise comparison of methods using NLPD(Left) and MSE(right)

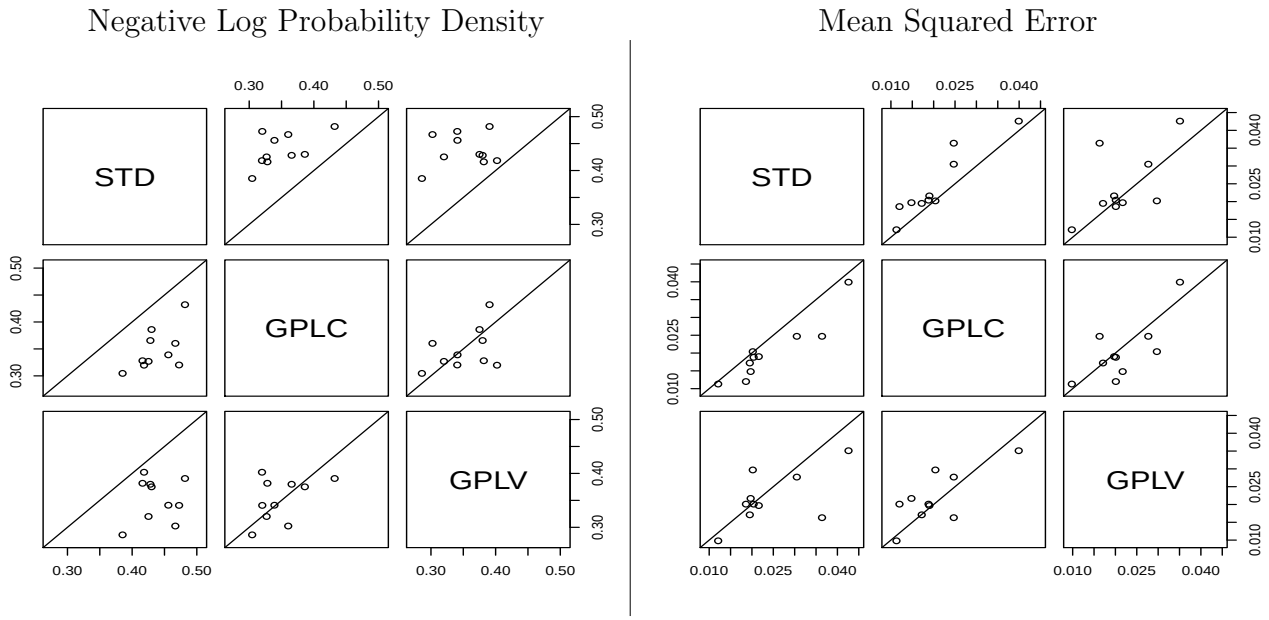

Figure 2.13: Dataset M1: Pairwise comparison of methods using NLPD(Left) and MSE(right)

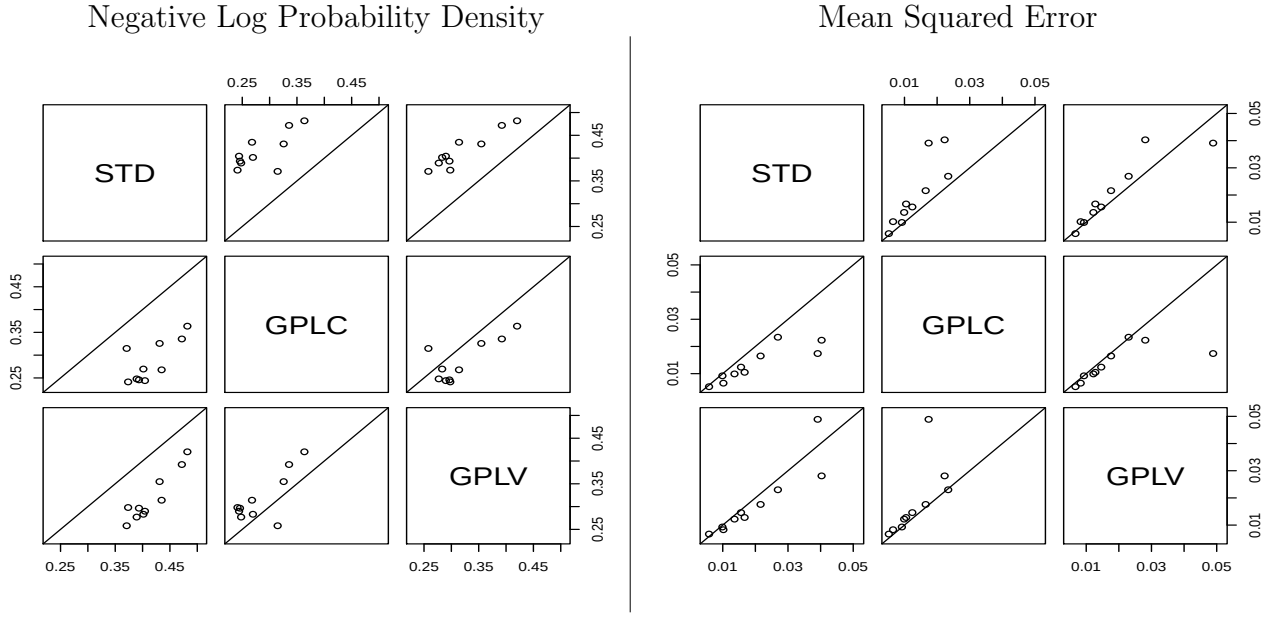

Figure 2.14: Dataset M2: Pairwise comparison of methods using NLPD(Left) and MSE(right)

Another measure of sampler efficiency is the time it takes for a sampler to reach equilibrium, i.e. the stationary distribution of the Markov chain. It is common practice to look at trace plots of log-probability density values to decide whether or not a chain has reached equilibrium (this is usually how the number of "burn-in" iterations is decided). A sampler that takes less time to get a new uncorrelated sample can usually achieve equilibrium faster, and vice versa, though this is not always the case.

Note that both autocorrelation time and time to reach equilibrium take the number of iterations of the a Markov chain as their unit of measurement. However, the CPU time required for an iteration differs between samplers. For a fair comparison, we adjust τ by multiplying it with the average CPU time (in seconds) per iteration. The result, which we denote as $\tilde{\tau}$, measures the CPU time a sampler needs to obtain an uncorrelated sample. Similarly, to fairly compare time to reach equilibrium using trace plots, we will adjust the number of iterations in the plots so that each entire trace takes the same amount of time.

We set the initial values of the hyperparameters to be their prior means, and randomly sample n values from their priors to initialize the latent variables (once these numbers are set, they are fixed for all the test-runs). We then run the three samplers five times, all from this fixed set of initial values but with different random seeds. The average adjusted autocorrelation times are listed in Table 2.2. The modified Metropolis sampler significantly outperforms the others at sampling the latent variables: it is about 50 to 100 times faster than the regular Metropolis sampler and slice sampler. For the hyperparameters, the modified Metropolis sampler performs roughly the same as the standard Metropolis for η_y and η_z, and more than 2 times better than the standard Metropolis for ρ_y and ρ_z. Both of them seems to work better than slice sampler, but the difference is much smaller than the difference in sampling latent variables.

Figure 2.15 shows selected autocorrelation plots from one of the five runs (adjusted for computation time). Figure 2.16 gives the trace plots of the three methods for one

	$\tilde{\tau}$						t
	η_y	ρ_y	η_z	ρ_z	$\sum_i z_i$	$\sum_i z_i^2$	
Modified Metropolis	3.06	4.92	3.09	10.63	0.32	0.46	0.103
Metropolis	2.81	11.21	2.73	27.26	13.78	13.92	0.073
Slice	13.37	16.71	11.85	23.65	37.81	53.81	0.389

Table 2.2: Autocorrelation times multiplied by computation time per iteration of MCMC methods for GPLV. The last column t is the computation time (in seconds) per iteration.

of the five runs (other runs are similar). It is clear that the modified Metropolis takes the least time to reach equilibrium. Starting from the prior mean (which seem to be a reasonable initial point), with log-probability density (LPD) value of approximately -13, the modified Metropolis method immediately pushes the LPD to 70 at the second step, and then soon declines slightly to what appears to be the equilibrium distribution. The other two methods both take much more time to reach this equilibrium distribution.

We conclude that the modified Metropolis is the best of these MCMC method — the fastest to reach equilibrium, the best at sampling latent values thereafter, and at least as good at sampling hyperparameters.

2.6 Related work

While standard Gaussian Process regression models assume constant variance of residuals, there is also a growing number of works that address the non-constant variance problem.

Goldberg *et al.* (1998) first tackled this problem with their GPLV approach, which we discussed extensively in this chapter. Several methods proposed later are based on the GPLV approach.

Kersting *et al.* (2007) develop an EM-like algorithm that gives the approximate "most likely" estimates of residuals as well as the hyperparameters. The biggest advantage of this approach is its fast computation, as it avoids the time-consuming MCMC process. However, since this approach has the same foundation as Goldberg *et al.*'s model, it also

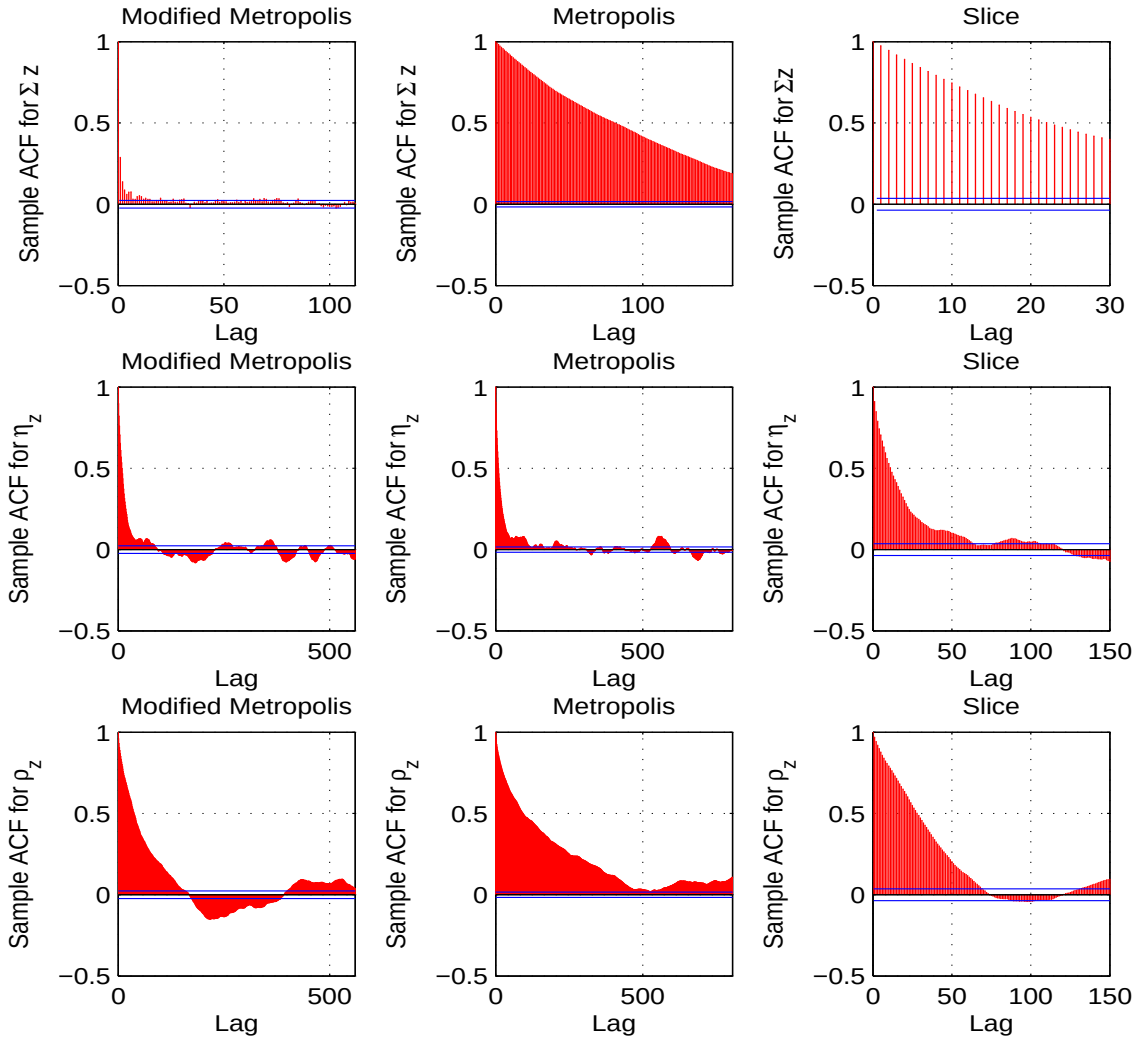

Figure 2.15: Selected autocorrelation plots of MCMC methods for GPLV (with horizontal scales that adjust for computation time).

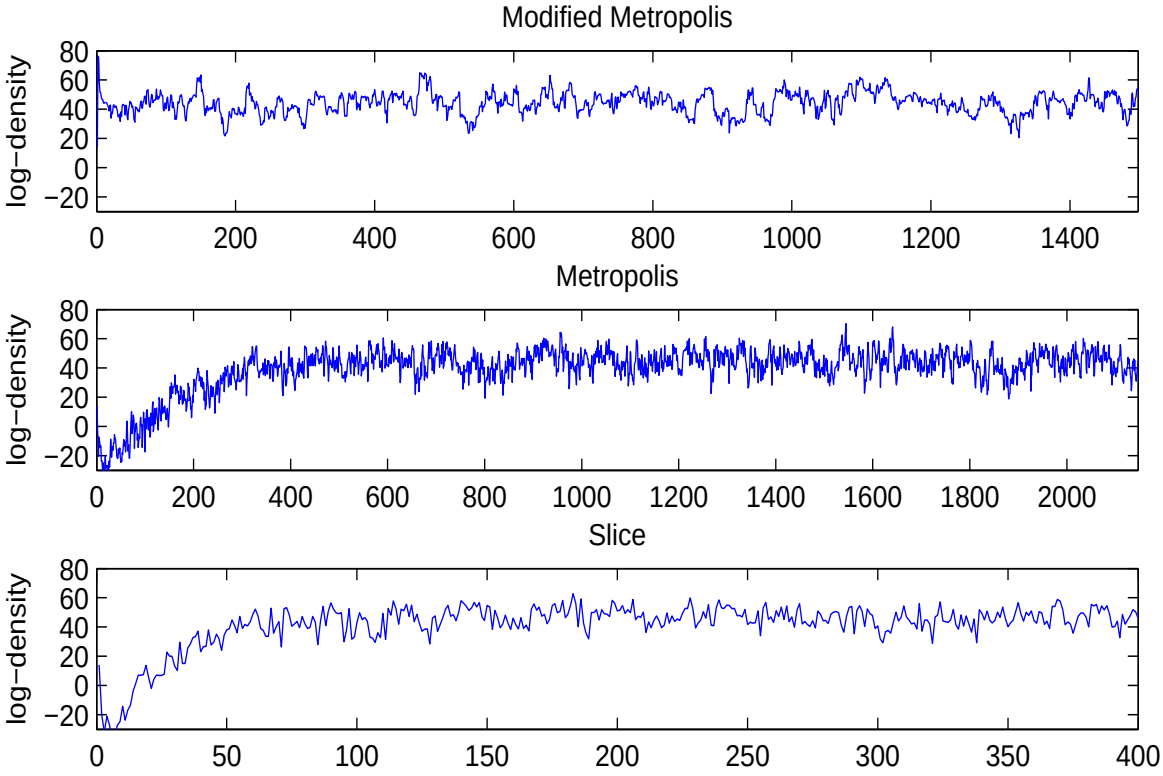

Figure 2.16: Trace plots of log posterior density for MCMC methods for GPLV. The top plot shows the trace of log-probability density values of the the first 1500 iterations of the modified Metropolis sampler. The middle plot shows the initial iterations of the regular Metropolis sampler, with the number adjusted to take the same time as the 1500 standard Metropolis sampler iterations. The bottom plot is the trace of the initial iterations of the slice sampler, also with the number adjusted to take the same computation time as the others.

shares the same limitations. For instance, this approach also assumes Gaussian residuals, which means when the residuals are not Gaussian, we expect it will not outperform our GPLC model in terms of MSE and NLPD. The authors also point out that the algorithm does not guarantee convergence, and may instead oscillate amongst local maxima. This problem is later addressed in a related work by Quadrianto *et al.* (2009), where a Maximum *a posteriori* (MAP) type of estimator is used in place of the most likely estimator. However, this approach still assumes Gaussian residuals, and tends to overfit for a large number of latent variables and large values of the latent variables, as Lazaro-Gredilla and Titsias (2011) point out.

Also based on Goldberg *et al.*'s model, Lazaro-Gredilla and Titsias (2011) take a variational inference approach, producing approximate estimates (MAP) of the residuals. This method doesn't have the overfitting issue the Quadrianto *et al.*'s approach has. It also seems to produce better results (in terms of MSE and NLPD) than the previous two methods on the same benchmark datasets.

Another variational method, proposed by Adams and Stegle (2008), models the data as the point-wise product of two latent GPs so that the non-stationary variations of amplitude can be inferred. They propose an approximate inference scheme using Expectation Propagation to reduce computation.

Gaussian Process Regression Network (GPRN), proposed by Wilson, *et al.* (2012), combines the ideas of Bayesian neural networks and Gaussian processes. The scalar response version of the model can be expressed as:

$$y_i = \mathbf{w}(x_i)^T[\mathbf{f}(x_i) + \sigma_f \boldsymbol{\epsilon_i})] + \sigma_y z_i \tag{2.24}$$

where $\boldsymbol{\epsilon}_i \sim N(0, I_{q\times q})$ and $z_i \sim N(0,1)$ are indepdent random noises, $\mathbf{w}(x_i)$ is a vector of independent GPs, with $\mathbf{w}_j(x_i)$ having zero mean and covariance function k_w, and $\mathbf{f}(x_i)$ is a vector of independent GPs, with component j having zero mean and covariance

function $k_{\mathbf{f}_j}$.

If we explicitly separate the dynamic signal and noise correlations, equation (2.24) can be rewritten as

$$y_i = \mathbf{w}(x_i)^T \mathbf{f}(x_i) + \sigma_f \mathbf{w}(x_i)^T \boldsymbol{\epsilon}_i + \sigma_y z_i \tag{2.25}$$

We can see that the model effectively uses $\mathbf{w}(x_i)^T \mathbf{f}(x_i)$ to model the mean function, and $\sigma_f \mathbf{w}(x_i)^T \boldsymbol{\epsilon}_i + \sigma_y z_i$ to model the residuals. Since the variance of the "noise" for case i is $\sigma_f^2 \sum_{j=1}^{q} w_j(x_i)^2 + \sigma_y^2$, it is input-dependent. The covariance functions of each of k_{f_i} can be entirely different, so the overall covariance function (of the mean function we are trying to learn) may be switching between region with very different covariance structures.

Note that GPRN still assumes Gaussian residuals, as the "noise" part of the decomposition is a linear combination of independent Gaussian random variables. Moreover, this model is practical when q is small. When $q = 1$, the model reduces to

$$y_i = w(x_i) f(x_i) + \sigma_f w(x_i) \epsilon_i + \sigma_y z_i \tag{2.26}$$

where w, f and ϵ are all scalars. If we put a GP prior on $w(x)$, the model will behave similarly to a GPLV model described in (2.13), with the main function being $\tilde{f}(x) = w(x)f(x)$, and the secondary function being $\tilde{r}(x) = \sqrt{\sigma_f^2 w(x)^2 + \sigma_y^2}$.

Some financial time series don't require a "main" GP, since the mean response can be taken to be always zero. In this context, Wilson and Ghahramani (2010) use the "elliptical slice sampling" method of Murray *et al.* (2010) to sample latent values that determine the variances of observations. Elliptical slice sampling is related to the modified Metropolis method used in this thesis. It would be interesting to see how they compare in a general regression context.

Mooij *et al.* (2010) presented a model which is very similar to our GPLC model, where they use a "probability latent variable model" to infer causal association. To

represent that "X causes Y", Mooij *et al.* assume that the relationship between X and Y is not deterministic, but disturbed by a latent variable E. They further assume that E is independent of X, and has a standard normal distribution — an unobserved noise, which summarizes all other causes of Y. Values for the latent variables are obtained using a MAP estimator. Although this model and the GPLC model are essentially the same, they differ substantially in motivation and in the method of implementation. The Mooij *et al.* paper is motivated by casual inference, while our model is proposed for general non-linear regression problems. Mooij *et al.*'s approach makes inferences based on a point estimate of the latent variables, while our GPLC is based on the posterior samples obtained using MCMC methods.

Chapter 3

MCMC with Temporary Mapping and Caching

3.1 Introduction

Evaluating the posterior probability density function is the most costly operation when Markov Chain Monte Carlo (MCMC) is applied to many Bayesian inference problems. One example is the Gaussian Process regression model (see Section 1.2 for a brief introduction), for which the time required to evaluate the posterior probability density increases with the cube of the sample size. However, several fast but approximate methods for Gaussian Process models have been developed. We show in this chapter how such an approximation to the posterior distribution for parameters of the covariance function in a Gaussian process model can be used to speed up sampling, using either of two schemes, based on "mapping to a discretizing chain" or "mapping with tempered transitions". Both schemes produce an exactly correct MCMC method, despite using an approximation to the posterior density for some operations. [1]

In the next section, we describe a general scheme for contructing efficient MCMC

[1] Part of this chapter originally appeared in Wang and Neal (2012).

methods using temporary mapping and caching techniques, first introduced by Neal (2006), which is the basis for both of the schemes for using approximations that are introduced in this paper.

One possibility for a space to temporarily map to is the space of Markov chain realizations that leave a distribution π^* invariant. Our hope is that if we use such a space with a π^* that is a good approximation to π, but faster to compute, then MCMC with temporary mapping and caching will be faster than MCMC methods using only π.

We then consider how the tempered transiton method due to Neal (1996b) can also be viewed as mapping temporary to another space. Using this view, we give a different proof that detailed balance holds for tempered transitions. We then discuss how the sequence of transitions $\hat{T}_1, \hat{T}_2, ..., \check{T}_2, \check{T}_1$ (which collectively form the tempered transition) should be chosen when they are defined using fast approximations, rather than (as in the original context for tempered transtions) by modifying the original distribution, π, in a way that does not reduce computation time.

We apply these two proposed schemes to Gaussian process regression models that have a covariance function with unknown hyperparameters, whose posterior distribution must be sampled using MCMC. We discuss several fast GP approximation methods that can be used to contruct an approximate π^*. We conclude by presenting experiments on synthetic datasets using the new methods that show that these methods are indeed faster than standard methods using only π.

3.2 MCMC with Temporary Mapping and Caching

To start, we present two general ideas for improving MCMC — temporarily mapping to a different state space, and caching the results of posterior density computations for possible later use.

3.2.1 Creating Markov transitions using temporary mappings

As is introduced in Section 1.3, we can obtain samples of a target distribution π from space $\mathcal{X}$ using a Markov chain with transition probabilities $T(x'|x)$, such that

$$\int \pi(x)T(x'|x)dx = \pi(x') \tag{3.1}$$

i.e., $T(x'|x)$ leaves the target distribution π invariant. There are many ways to form such a transition, e.g. the Metropolis algorithm (Metropolis *et al.*, 1953) and slice sampling (Neal, 1993).

The temporary mapping technique (Neal, 2006) defines such a transition via three other stochastic mappings, $\hat{T}$, $\bar{T}$ and $\check{T}$, as follows:

$$x \xrightarrow{\hat{T}} y \xrightarrow{\bar{T}} y' \xrightarrow{\check{T}} x' \tag{3.2}$$

where $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$. Starting from x, we obtain a value y in the temporary space $\mathcal{Y}$ by $\hat{T}(y|x)$. The target distribution for y has probability mass/density function $\rho(y)$. We require that

$$\int \pi(x)\hat{T}(y|x)dx = \rho(y) \tag{3.3}$$

We then obtain another sample y' using $\bar{T}(y'|y)$, which leaves ρ invariant:

$$\int \rho(y)\bar{T}(y'|y)dy = \rho(y') \tag{3.4}$$

Finally, we map back to $x' \in \mathcal{X}$ using $\check{T}(x'|y)$, which we require to satisfy

$$\int \rho(y')\check{T}(x'|y')dy' = \pi(x') \tag{3.5}$$

It's easy to see that the combined transition $T(x'|x) = \int\int \hat{T}(y|x)\bar{T}(y'|y)\check{T}(x'|y')dydy'$

leaves π invariant:

$$\begin{aligned}
\int \pi(x)T(x'|x)dx &= \int\int\int \pi(x)\hat{T}(y|x)\bar{T}(y'|y)\check{T}(x'|y')dydy'dx && (3.6)\\
&= \int\int \rho(y)\bar{T}(y'|y)\check{T}(x'|y')dydy' && (3.7)\\
&= \int \rho(y')\check{T}(x'|y')dy' && (3.8)\\
&= \pi(x') && (3.9)
\end{aligned}$$

Quite a few existing methods can be viewed as mapping to temporary spaces. For instance, the technique of temporarily introducing auxiliary variables can be considered as mapping from x to $y = (x, z)$, where z is a set of auxiliary variables.

3.2.2 Caching values for future re-use

Many MCMC transitions require evalulating the probability density of π, up to a possibly unknown normalizing constant. For example, each iteration of the Metropolis algorithm needs the probability density values of both the current state x and the candidate state x^*. Since these evaluations typically dominate the MCMC computation time, it may be desirable to save ('cache') computed values of $\pi(x)$ so they can be re-used when the same state x appears in the chain again.

Caching is always useful for the Metropolis algorithm, since if we reject a proposal x^*, we will need $\pi(x)$ for the next transition, and if we instead accept x^* then it becomes the current state and we will need $\pi(x^*)$ for the next transition.

When the proposal distribution is discrete (as it will always be when the state space is discrete), the probability of proposing an x^* that was previously proposed can be positive, so saving the computed value of $\pi(x^*)$ may be beneficial even if x^* is rejected. When the state space is continuous, however, the proposal distributions commonly used are also continuous, and we will have zero probability of proposing the same x^* again. But in this case, as we will see next, caching can still be beneficial if we first map to another

space with a "discretizing chain".

3.3 Mapping to a discretizing chain

To take full advantage of both mapping and caching, we propose a temporary mapping scheme where the temporary space is continuous, but is effectively discrete with regard to transitions $\bar{T}$.

Let $R(x'|x)$ be the transition probabilities for a Markov chain which leaves π^* invariant. Let $\tilde{R}(x|x') = R(x'|x)\pi^*(x)/\pi^*(x')$ be the reverse transition probabilities, which are easily shown to be well-defined and to also leave π^* invariant.

We map from $\mathcal{X}$ to $\mathcal{Y}$, a space of realizations of this Markov chain of length K, where one time step of this chain is "marked". To map $x \in \mathcal{X}$ to $y \in \mathcal{Y}$, we use a $\hat{T}$ that operates as follows:

- Choose k uniformly from $0, ..., K-1$.
- Simulate $K-1-k$ forward transition steps using R starting at $x_k = x$, producing states $x_{k+1}, ..., x_{K-1}$.
- Simulate k reverse transitions using $\tilde{R}$, starting at $x_k = x$, producing states $x_{k-1}, ..., x_0$.
- Set the "marked" time step to k.

The transition $\bar{T}$ moves the mark along the chain from k to another time step $k' \in \{0, \ldots, K-1\}$, while keeping the current chain realization, $(x_0, \ldots, x_{K-1})$, fixed. The transition $\check{T}$ just takes the marked state, so $x' = x_{k'}$. The actual implementation will not necessarily simulate all $K-1$ steps of the discretizing chain — a new step is simulated only when it is needed. We can then let K go to infinity, so that $\bar{T}$ can move the mark any finite number of steps forward or backward.

Figure 3.1 illustrates this scheme. Note that an element $y \in \mathcal{Y}$ is a chain realization with a mark placed on the time step k. We write $y = (k; x_0, ..., x_{K-1})$. When we

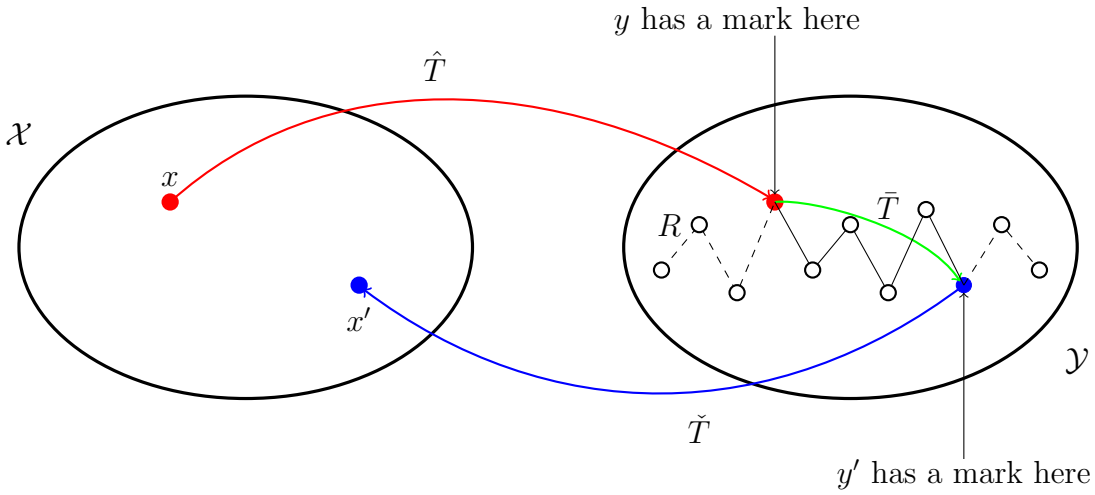

Figure 3.1: Mapping to a discretizing chain and back.

say we "move the mark from k to k'", we actually use a transition $\bar{T}$ to move from $y = (k; x_0, ..., x_{K-1})$ to $y' = (k'; x_0, ..., x_{K-1})$, where y and y' share the same chain realization and differ only on the marked position. We are free to choose the way $\bar{T}$ moves the mark in any way that leaves ρ invariant — for instance, we can pick an integer s and propose to move the mark from k to $k+s$ or $k-s$ with equal probabilities. We can make r such moves within each $\bar{T}$ update. The discretizing chain makes the state space effectively discrete, even though the space $\mathcal{Y}$ is continuous, and consequently, when we move the mark around the chain realization, there is a positive probability of hitting a location that has been visited before.

The transition $\bar{T}$ has to leave $\rho(y)$ invariant. We compute the ratio of $\rho(y')$ and $\rho(y)$ to see how we can construct a such a $\bar{T}$. ρ has been implicitly defined in (3.3) as the distribution resulting from applying $\hat{T}$ to x drawn from π. The probability to sample y is given by the simulation process described above (i.e. start from x, simulate $K-1-k$

forward steps using R and k backward steps using $\tilde{R}$), namely, if $y = (k; x_0, ..., x_{K-1})$,

$$\begin{aligned}\rho(y) &= \pi(x_k)\frac{1}{K}R(x_{k+1}|x_k)\cdots R(x_{K-1}|x_{K-2}) \times \tilde{R}(x_{k-1}|x_k)\cdots\tilde{R}(x_0|x_1) \\ &= \frac{\pi(x_k)}{\pi^*(x_k)}\frac{1}{K}\underbrace{\pi^*(x_k)R(x_{k+1}|x_k)\cdots R(x_{K-1}|x_{K-2}) \times \tilde{R}(x_{k-1}|x_k)\cdots\tilde{R}(x_0|x_1)}_{:=A} \end{aligned} \tag{3.10}$$

An expression for $\rho(y')$ can be similarly obtained for $y' = (k'; x_0, ..., x_{K-1})$:

$$\rho(y') = \frac{\pi(x_{k'})}{\pi^*(x_{k'})}\frac{1}{K}\underbrace{\pi^*(x_{k'})R(x_{k'+1}|x_{k'})\cdots R(x_{K-1}|x_{K-2}) \times \tilde{R}(x_{k'-1}|x_{k'})\cdots\tilde{R}(x_0|x_1)}_{:=A'} \tag{3.11}$$

We take out a factor of the ratio of densities π/π^* from both (3.10) and (3.11), and write the remaining term as A or A', as indicated in the respective equation. Since R and $\tilde{R}$ are reverse transitions with respect to π^*, if $k' > k$, then

$$\begin{aligned}&\pi^*(x_k)R(x_{k+1}|x_k)\cdots R(x_{k'}|x_{k'-1}) \\ &\quad= \tilde{R}(x_k|x_{k+1})\pi^*(x_{k+1})R(x_{k+2}|x_{k+1})\cdots R(x_{k'}|x_{k'-1}) \\ &\quad\vdots \\ &\quad= \tilde{R}(x_k|x_{k+1})...\tilde{R}(x_{k'-1}|x_{k'})\pi^*(x_{k'})\end{aligned} \tag{3.12}$$

It therefore follows that $A = A'$. A similar argument shows that $A = A'$ when $k' \leq k$. Thus the ratio of $\rho(y')$ and $\rho(y)$ is

$$\frac{\rho(y')}{\rho(y)} = \frac{\pi(x_{k'})/\pi^*(x_{k'})}{\pi(x_k)/\pi^*(x_k)} \tag{3.13}$$

Equation (3.13) implies that to leave ρ invariant we can use a Metropolis type transition,

$\bar{T}$, that proposes to move the mark from k to k' and accepts the move with probability

$$\min\left(1, \frac{\pi(x_{k'})/\pi^*(x_{k'})}{\pi(x_k)/\pi^*(x_k)}\right)$$

Note that if $\pi = \pi^*$, then the transition $\bar{T}$ will accept a move of the mark to any other time step on the discretizing chain, since the discretizing chain actually leaves the target distribution π^* invariant and therefore every time step of this chain is a valid sample of π. If $\pi^* \neq \pi$, but is very similar to π, we can hope the acceptance rate will be high. In addition, if the evaluation of $\pi^*(x)$ takes much less time than that of $\pi(x)$, mapping to the discretizing chain and then proposing large moves of the mark can save computation time, since it effectively replaces evaluations of π with evaluations of π^*, except for the acceptance decisions.. On the other hand, if π^* is completely arbitrary, the acceptance rate will be low, and if the evalution of π^* is not much faster than $\pi(x)$, we will not save computation time. These π^*'s are not useful. We need π^* to be a fast but good approximation to π. We will discuss this in the context of GP models in a later section.

Every time we map into a temporary space, we can make multiple $\bar{T}$ updates (move the "mark" several times). This way we can take advantage of the "caching" idea, since sometimes the mark will be moved to a state where π has already been computed, and therefore no new computation is needed. The number of updates is a tuning parameter, which we denote as "r". Another tuning parameter, which we denote as "s", is the number of steps of transition R to "jump" when we try to move the mark. Note that although we only "bring back" (using $\check{T}$) the last updated sample as x', all of the marked states are valid samples of $\pi(x)$, and can be used for computing expectations with respect to π if desired.

3.4 Tempered transitions

The "tempered transitions" method of Neal (1996b) can also be viewed as mapping to a temporary space. This method aims to sample from π using a sequence of distributions $\pi_0,\ \pi_1, \ldots,\ \pi_n$, with $\pi = \pi_0$.

For $i = 1, \ldots, n$, let $\hat{T}_i$ (called the "up" transition) and $\check{T}_i$ (the "down" transition) be mutually reversible transitions with respect to the density π_i — i.e. for any pair of states x and x',

$$\pi_i(x)\hat{T}_i(x'|x) = \check{T}_i(x|x')\pi_i(x') \tag{3.14}$$

This condition implies that both $\hat{T}_i$ and $\check{T}_i$ have π_i as their invariant distribution. If $\hat{T}_i = \check{T}_i$ then (3.14) reduces to the detailed balance condition. If $\hat{T}_i = S_1 S_2 ... S_k$ with all of S_i being reversible transitions, then $\check{T}_i = S_k S_{k-1} ... S_1$ would satisfy condition (3.14).

We map from $x \in \mathcal{X}$ to $y \in \mathcal{Y}$, a space of realizations of tempered transitions, using a $\hat{T}$ that operates as follows:

Generate $\hat{x}_1$ from x using $\hat{T}_1$;

Generate $\hat{x}_2$ from $\hat{x}_1$ using $\hat{T}_2$;

$\vdots$

Generate $\bar{x}_n$ from $\hat{x}_{n-1}$ using $\hat{T}_n$.

Generate $\check{x}_{n-1}$ from $\bar{x}_n$ using $\check{T}_n$;

Generate $\check{x}_{n-2}$ from $\check{x}_{n-1}$ using $\check{T}_{n-1}$;

$\vdots$

Generate x^* from $\check{x}_1$ using $\check{T}_1$.

Note that $\hat{T}$ is distinct from $\hat{T}_1, \hat{T}_2, ..., \hat{T}_n$, and $\check{T}$ is also distinct from $\check{T}_1, \check{T}_2, ... \check{T}_n$.

We denote an element $y \in \mathcal{Y}$ as $y = (x, \hat{x}_1, ..., \bar{x}_n, ..., \check{x}_1, x^*)$.

$\bar{T}$ attempts to flip the order of y, accepting the flip with probability

$$\min\left(1, \frac{\pi_1(\hat{x}_0)}{\pi_0(\hat{x}_0)} \cdots \frac{\pi_n(\hat{x}_{n-1})}{\pi_{n-1}(\hat{x}_{n-1})} \cdot \frac{\pi_{n-1}(\check{x}_{n-1})}{\pi_n(\check{x}_{n-1})} \cdots \frac{\pi_0(\check{x}_0)}{\pi_1(\check{x}_0)}\right) \tag{3.15}$$

where $\hat{x}_0$ and $\check{x}_0$ are synonyms for x and x^*, respectively, to keep notations consistent. In other words, with this probability, we set y' to $y^* = (x^*, \check{x}_1, ..., \bar{x}_n, ..., \hat{x}_1, x)$ (the order is reversed); otherwise we sset $y' = y$ (the order is preserved).

Finally, $\check{T}$ maps back to $x' \in \mathcal{X}$ by taking the first coordinate of y' (either the original x or x^*, depending on whether or not the flip was accepted).

Using the temporary mapping perspective, we can show that tempered transitions are valid updates, leaving π invariant, by defining ρ to be the result of applying $\hat{T}$ to a point drawn from π, and then showing that $\bar{T}$ leaves ρ invariant, and that $\check{T}$ produces a point distributed as π from a point distributed as ρ.

The $\hat{T}$ mapping from $x = \hat{x}_0$ to $y = (\hat{x}_0, \hat{x}_1, ..., \bar{x}_n, ..., \check{x}_1, \check{x}_0)$ involves a sequence of transitions:

$$\hat{x}_0 \xrightarrow{\hat{T}_1} \hat{x}_1 \xrightarrow{\hat{T}_2} \hat{x}_2 \longrightarrow \cdots \longrightarrow \hat{x}_{n-1} \xrightarrow{\hat{T}_n} \bar{x}_n \xrightarrow{\check{T}_n} \check{x}_{n-1} \xrightarrow{\check{T}_{n-1}} \check{x}_{n-2} \longrightarrow \cdots \longrightarrow \check{x}_1 \xrightarrow{\check{T}_1} \check{x}_0$$

The probability density, ρ, for y can be computed from this as

$$\rho(y) = \pi_0(\hat{x}_0)\hat{T}_1(\hat{x}_1|\hat{x}_0) \cdots \hat{T}_n(\bar{x}_n|\hat{x}_{n-1})\check{T}_n(\check{x}_{n-1}|\bar{x}_n) \cdots \check{T}_1(\check{x}_0|\check{x}_1) \tag{3.16}$$

Similarly,

$$\rho(y^*) = \pi_0(\check{x}_0)\hat{T}_1(\check{x}_1|\check{x}_0) \cdots \hat{T}_n(\bar{x}_n|\check{x}_{n-1})\check{T}_n(\hat{x}_{n-1}|\bar{x}_n) \cdots \check{T}_1(\hat{x}_0|\hat{x}_1) \tag{3.17}$$

Also note that the transitions $\hat{T}_i$ and $\check{T}_i$ satisfy the mutual reversibility condition (3.14), which can be rewritten as

$$\frac{\hat{T}_i(x'|x)}{\check{T}_i(x|x')} = \frac{\pi_i(x')}{\pi_i(x)} \tag{3.18}$$

Now we compute the ratio of probability densities of y^* and y:

$$\begin{aligned}
\frac{\rho(y^*)}{\rho(y)} &= \frac{\pi_0(\check{x}_0)\hat{T}_1(\check{x}_1|\check{x}_0)\cdots\hat{T}_n(\bar{x}_n|\check{x}_{n-1})\check{T}_n(\hat{x}_{n-1}|\bar{x}_n)\cdots\check{T}_1(\hat{x}_0|\hat{x}_1)}{\pi_0(\hat{x}_0)\hat{T}_1(\hat{x}_1|\hat{x}_0)\cdots\hat{T}_n(\bar{x}_n|\hat{x}_{n-1})\check{T}_n(\check{x}_{n-1}|\bar{x}_n)\cdots\check{T}_1(\check{x}_0|\check{x}_1)} \\
&= \pi_0(\check{x}_0)\cdot\frac{\hat{T}_1(\check{x}_1|\check{x}_0)}{\check{T}_1(\check{x}_0|\check{x}_1)}\cdots\frac{\hat{T}_n(\bar{x}_n|\check{x}_{n-1})\check{T}_n(\hat{x}_{n-1}|\bar{x}_n)}{\check{T}_n(\check{x}_{n-1}|\bar{x}_n)\hat{T}_n(\bar{x}_n|\hat{x}_{n-1})}\cdots\frac{\check{T}_1(\hat{x}_0|\hat{x}_1)}{\hat{T}_1(\hat{x}_1|\hat{x}_0)}\cdot\frac{1}{\pi_0(\hat{x}_0)} \qquad (3.19) \\
&= \pi_0(\check{x}_0)\cdot\frac{\pi_1(\check{x}_1)}{\pi_1(\check{x}_0)}\cdots\frac{\pi_n(\bar{x}_n)}{\pi_n(\check{x}_{n-1})}\cdot\frac{\pi_n(\hat{x}_{n-1})}{\pi_n(\bar{x}_n)}\cdots\frac{\pi_1(\hat{x}_0)}{\pi_1(\hat{x}_1)}\cdot\frac{1}{\pi_0(\hat{x}_0)} \qquad (3.20) \\
&= \frac{\pi_1(\hat{x}_0)}{\pi_0(\hat{x}_0)}\cdots\frac{\pi_n(\hat{x}_{n-1})}{\pi_{n-1}(\hat{x}_{n-1})}\cdot\frac{\pi_{n-1}(\check{x}_{n-1})}{\pi_n(\check{x}_{n-1})}\cdots\frac{\pi_0(\check{x}_0)}{\pi_1(\check{x}_0)} \qquad (3.21)
\end{aligned}$$

(3.20) follows from (3.18). (3.19) and (3.21) are obtained simply by reordering terms.

From (3.21), we see that the probability of accepting the flip from y to y^* given by (3.15) is equal to $\min(1, \rho(y^*)/\rho(y))$, and thus $\bar{T}$ satisfies detailed balance with respect to ρ. It is also clear from (3.16) that the marginal distribution under ρ of the first component of y is $\pi_0 = \pi$, and thus $\check{T}$ maps from ρ to π.

The original motivation of the tempered transition method described by Neal (2006) is to move between isolated modes of multimodal distributions. The distributions $\pi_1, ..., \pi_n$ are typically of the same class as π, but broader, making it easier to move between modes of π (typically, as i gets larger, the distribution π_i gets broader, thus making it more likely that modes have substantial overlap). Evaluating the densities for $\pi_1, ..., \pi_n$ typically takes similar computation time as evaluating the density for π. Our mapping-caching scheme, on the other hand, is designed to reduce computation. Ideally, in our scheme the bigger i is, the faster is the evaluation of $\pi_i(x)$. One possibility for this is that each π_i is an approximation of π, and as i increases the computation of π_i becomes cheaper (but worse).

The two methods we propose in this work are equivalent if the following are all true:

- For mapping to a discretizing chain:

    1. The transition R which leaves π^* invariant is reversible.

2. $s = 2k$, i.e. $\bar{T}$ always attempts to move the mark over an even number of R updates.

3. $r = 1$, i.e. $\bar{T}$ attempts to move the mark only once within each mapping.

- For mapping by tempered transitions:

1. $n = 1$, i.e., there is only one additional distribution.

2. $\hat{T}_1 = \check{T}_1 = R^k$, i.e. these transitions consist of k updates using R.

When all above are true except that $n > 1$, so more than one additional distribution is used in the tempered transitions, we might expect tempered transitions to perform better, as they propose a new point through the guidance of these additional distributions, and computations for these additional distributions should be negligible, if they are faster and faster approximations. On the other hand, we might think that $r > 1$ will improve the performance when mapping to a discretizing chain, since then caching could be exploited. So each method may have its own advantages.

3.5 Application to Gaussian process models

We now show how these MCMC methods can be applied to Bayesian inference for Gaussian process models.

3.5.1 Approximating π for GP models

As discussed in Section 3.3, using a poor π^* for the discretizing chains on $\mathcal{Y}$, or poor π_i for tempered transitions, can lead to a poor MCMC method which is not useful. We would like to choose approximations to π that are good, but that can nevertheless be computated much faster than π. For GP regression models, π will be the posterior distribution of the hyperparameters, θ.

Quite a few efficient approximation methods for GP models have been discussed from a different perspective. For example, Quiñonero-Candela (2007) categorizes these approximations in terms of "effective prior". Any approximation method that can be used to create an approximation of the posterior density for the hyperparameters can be used here. Most methods that aim to approximate the training (i.e. finding approximated estimates for the parameters) should be useful. On the other hand, the approximation method for prediction (i.e. finding approximated response value y_* corresponding to some new input covariate x_*) are most likely not useful for the methods discussed in this chapter. Whether or not a particular method can work well depends on both efficiency and accuracy. A very fast method that's less accurate may very well be better than a slow but accurate method.

Below, we discuss two classes of approximation methods.

Subset of data (SOD)

The most obvious approximation is to simply take a subset of size m from the n observed pairs (x_i, y_i) and use the posterior distribution given only these observations as π^*:

$$\pi^*(\theta) \;=\; \mathcal{N}(y_{(m)}|0, \hat{C}_{(m)}(\theta))\, p(\theta) \tag{3.22}$$

where $p(\theta)$ is the prior for θ, the vector of hyperparameters, and $\mathcal{N}(a|\mu, \Sigma)$ denotes the probability density of a multivariate normal distribution $N(\mu, \Sigma)$ evaluated at a. $\hat{C}_{(m)}(\theta)$ is computed based on hyperparameters θ and the m observations in the subset.

Even though the SOD method seems quite naive, it does speed up computation of the Cholesky decomposition of C from time proportional to n^3 to time proportional to m^3. If a small subset (say 10% of the full dataset) is used to form π^*, we can afford to do a lot of Markov chain updates for π^*, since the time it takes to make these updates will be quite small compared to a computation of π. So a π^* formed by this method might

still be useful.

To form a π^* using SOD, we need the following major computations, if there are p covariates:

Operation	Complexity
Compute $\hat{C}_{(m)}$	pm^2
Find chol($\hat{C}_{(m)}$)	m^3

Using low-rank plus diagonal matrices

A covariance matrix in a GP model typically has the form $C = K + \sigma^2 I$, where K is the noise-free covariance matrix, and σ^2 is the residual variance. More generally, if the residual variance differs for different observations, the covariance matrix will be K plus a diagonal matrix giving these residual variances. If we approximate K by a matrix $\hat{K}$ with rank $m < n$, and let $\hat{C} = \hat{K} + \sigma^2 I$, then after writing $\hat{K} = BSB^T$, where B is n by m, we can quickly find $\hat{C}^{-1}$ by taking advantage of the matrix inversion lemma (Woodbury, 1950), which states that

$$(BSB^T + D)^{-1} = D^{-1} - D^{-1}B(S^{-1} + B^T D^{-1} B)^{-1} B^T D^{-1} \tag{3.23}$$

This can be simplified as follows when $D = dI$, where d is a scalar, B has orthonormal columns (so that $B^T B = I$), and S is a diagonal matrix with diagonal elements given by

the vector s, denoted by $\text{diag}(s)$:

$$(B\,\text{diag}(s)\,B^T + dI)^{-1} = d^{-1}I - d^{-1}IB(\text{diag}(1/s) + B^T d^{-1} IB)^{-1}B^T d^{-1}I \tag{3.24}$$

$$= d^{-1}I - d^{-2}B(\text{diag}(d/s)/d + B^T B/d)^{-1}B^T \tag{3.25}$$

$$= d^{-1}I - d^{-1}B(\text{diag}(d/s) + I)^{-1}B^T \tag{3.26}$$

$$= d^{-1}I - d^{-1}B(\text{diag}((s+d)/s)))^{-1}B \tag{3.27}$$

$$= d^{-1}I - B\,\text{diag}(s/(d(s+d)))\,B^T \tag{3.28}$$

Expressions above such as $1/s$ denote element-by-element arithmetic on the vector operands.

We can use the matrix determinant lemma (Press, *et al.*, 1992) to compute the determinant of $\hat{C}$.

$$\det(BSB^T + D) \;=\; \det(S^{-1} + B^T D^{-1} B)\det(D)\det(S) \tag{3.29}$$

When $D = dI$ with d being a scalar, $\det(D) = d^n$ is trivial, and $\det(S^{-1} + B^T D^{-1} B)$ can be found from the Cholesky decomposition of $S^{-1} + B^T D^{-1} B$.

Once we obtain $\hat{C}^{-1}$ and $\det(\hat{C})$, we can easily establish our π^*:

$$\pi^*(\theta) = \mathcal{N}(y|0, \hat{C})p(\theta) \tag{3.30}$$

The Eigen-exact approximation

Since the noise-free covariance matrix, K, is non-negative definite, we can write it as $K = E\Lambda E^T = \sum_i^n \lambda_i e_i e_i^T$, where E has columns $e_1, e_2, ..., e_n$, the eigenvectors of K, and the diagonal matrix Λ has the eigenvalues of K, $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ on its diagonal. This is known as the eigendecomposition. A natural choice of low-rank plus diagonal approximation would be $\hat{C} = \hat{K} + \sigma^2 I$ where $\hat{K} = BSB^T$ where B is an $n \times m$ matrix with columns $e_1, ..., e_m$, and S is a diagonal matrix with diagonal entries $\lambda_1, ..., \lambda_m$. We

expect this to be a good approximation if λ_{m+1} is close to zero.

With this approximation, $\hat{C}^{-1}$ can be computed rapidly from B and S using (3.28). However, the time needed to find the first m eigenvalues and eigenvectors (and hence B and S) is proportional to mn^2, with a much larger constant factor than for the n^3 computation of all eigenvalues and eigenvectors. In practice, depending on the values of m and n and the software implementation, a π^* formed by this method could even be slower than the original π. Since our experiments confirm this, we mention it here only because it is a natural reference point.

The Nyström-Cholesky approximation

In the Nyström method (Nyström, 1930; Reinhardt, 1985), we take a random m by m submatrix of the noise-free covariance matrix, K, which is equivalent to looking at the noise-free covariance for a subset of the data of size m, and then find its eigenvalues and eigenvectors. This takes time proportional to m^3. We will denote the submatrix chosen by $K^{(m,m)}$, and its eigenvalues and eigenvectors by $\lambda_1^{(m)}, ..., \lambda_m^{(m)}$ and $e_1^{(m)}, ..., e_m^{(m)}$. We can then approximate the first m eigenvalues and eigenvectors of the full noise-free covariance matrix by

$$\hat{\lambda}_i = (n/m)\lambda_i^{(m)} \tag{3.31}$$

$$\hat{e}_i = \frac{\sqrt{m/n}}{\lambda_i^{(m)}} K^{(n,m)} e_i^{(m)} \tag{3.32}$$

where $K^{(n,m)}$ is the n by m submatrix of K with only the columns corresponding to the m cases in the random subset.

The covariance matrix C can then be approximated in the same fashion as Eigen-exact, with the exact eigenvalues and eigenvectors replaced by the approximated eigenvalues $\hat{\lambda}_1, ..., \hat{\lambda}_m$ and eigenvectors $\hat{e}_1, ...\hat{e}_m$. However, a more efficient computational method for this approximation, requiring no eigenvalue/eigenvector computations, is available as

follows:

$$\hat{K} = K^{(n,m)}[K^{(m,m)}]^{-1}K^{(m,n)} \tag{3.33}$$

where $K^{(m,n)} = [K^{(n,m)}]^T$. We can find the Cholesky decomposition of $K^{(m,m)}$ as $R^T R$, in time proportional to m^3, with a much smaller constant factor than finding the eigenvalues and eigenvectors. Equation (3.33) can then be put in the form of BSB^T by letting $B = K^{(n,m)}R^{-1}$ and $S = I$. In practice, the noise free submatrix $K^{(m,m)}$ often has some very small positive eigenvalues, which can appear to be negative due to round-off error, making the Cholesky decomposition fail, a problem that can be avoided by adding a small jitter to the diagonal (Neal, 1993).

An alternative way of justifying the approximation in (3.33) is by considering the covariance matrix for the predictive distribution of all n noise-free observations from the random subset of m noise-free observations, which (from a generalization of (1.6)) is $K - K^{(n,m)}[K^{(m,m)}]^{-1}K^{(m,n)}$. When this is close to zero (so these m noise-free observations are enough to almost determine the function), $\hat{K}$ will be almost the same as K.

More sophisticated schemes for Nyström-Cholesky have been proposed. For instance, Drineas and Mahoney (2005) randomly select the m columns to construct $\hat{C}$ according to some "judiciously-chosen" and data-dependent probability distribution rather than uniformly choose the m columns.

To form a π^* using Nyström-Cholesky, we need the following major computations:

Operation	Complexity
Compute $K^{(n,m)}$	pmn
Find chol($K^{(m,m)}$)	m^3

3.6 Experiments

Here we report tests of the performance of the methods described in this chapter using synthetic datasets.

3.6.1 Experimental setup

The datasets we used in these experiments were randomly generated, with all covariates drawn independently from uniform distributions on the interval $[0, 1]$, and responses then generated according to a Gaussian process with specified hyperparameters.

We generated ten types of datasets in this way, with different combinations of the following:

- Number of observations: $n = 300$ or $n = 900$.
- Number of covariates: p=1 or $p = 5$.
- Type of covariance function: squared exponential covariance function with a single length scale (isotropic), or with multiple length scales (Automatic Relevance Determination, ARD). Note that these are identical when $p = 1$.
- Size of length scales: "short" indicates that a dataset has small length scales, "long" that it has large length scales.

The specific hyperparameter values that were used for each combination of covariance function and length scale are shown in Table 3.1. For each dataset, we use the follow priors for the hyperparameters:

$$\log \eta \sim N(0, 2^2)$$
$$\log \rho_i \sim N(0, 2^2)$$
$$\log \sigma \sim N(0, 2^2)$$

The efficiency of an MCMC method is usually measured by the autocorrelation time, τ, which is typically estimated by $\hat{\tau}$, the sample autocorrelation time (see section 1.4). Below, we will compare methods with respect to autocorrelation time of the log likelihood. For a fair comparison, we multiply the estimate of each method's autocorrelation times by the average CPU time it needs to obtain a new sample point.

3.6.2 Experiments with mapping to a discretizing chain

For each dataset, we tried the method of mapping to a discretizing chain using both a π^* formed with SOD and a π^* formed with Nyström-Cholesky. For comparison, we also ran a standard MCMC model. All the Markov chains were started from the hyperparameter values that were used to generate them, so these tests assess only autocorrelation time once the high-probability region of the posterior has been reached, not time needed for convergence when starting at a low-probability initial state. The adjustable parameters of each method were chosen to give good performance. All chains were run for 2000 iterations, and autocorrelation times were then computed based on the last two-thirds of the chain.

The standard MCMC method we used is a slice sampler (Neal, 2003), specifically a univariate slice sampler with stepping-out and shrinkage, updating parameters in sequence. For the discretizing Markov chain, the transition $R(x'|x)$ uses the same slice sampler. Although slice sampling has tuning parameters (the stepsize, w, and the upper limit on number of steps, M), satisfactory results can be obtained without extensive tuning (that is, the autocorrelation time of a moderately-well-tuned chain will not be much bigger than for an optimally-tuned chain). Because finding an optimal set of tuning parameters is generally hard (requiring much time for trial runs), we will accept the results using moderately-well-tuned chains.

There are many other MCMC method to choose from as the baseline method. Some more sophisticated method might perform better than the slice sampler we use here on

Length scale size	Length scale type	η	ρ
short	isotropic	5	$\rho = 0.1$
short	ARD	5	$\rho_i = 0.1i$
long	isotropic	5	$\rho = 2$
long	ARD	5	$\rho_i = 2i$

Table 3.1: Hyperparameter values used to generate the synthetic datasets. We set $c = 10$ and $\sigma = 0.2$ for all the datasets.

some particular datasets. In such cases, we can also use that MCMC method to construct the chain for π^*, and we would expect similar gain by mapping to such a discretizing chain. Given that there are a variety of good MCMC methods in the literature, it would not be possible to compare to all of them. But comparing to the slice sampler seems sufficient to show how we can construct efficient MCMC methods with temporary mapping and caching.

We found that $r = s = 1$ gives the best performance for the method of mapping to a discretizing chain when the slice sampler is used for $R(x'|x)$, at least if only fairly small values of r and s are considered. Recall that r is the number of $\bar{T}$ updates to do in each temporary mapping, and s is the number of steps of $R(x'|x)$ to propose to move the mark for each $\bar{T}$ update. Note that a single slice sampling update will usually evaluate π or π^* more than once, since an evaluation is needed for each outward step and for each time a point is sampled from the interval found by stepping out. Therefore if we didn't use a mapping method we would have to compute $\pi(x)$ several times for each slice sampling update. When a mapping method is used, $\pi(x)$ only needs to be evaluated once each update, for the new state (its value at the previous state having been saved), while meanwhile, $\pi^*(x)$ will be evaluated several times.

We tuned the remaining parameter m, the subset size for SOD, or the number of random columns for Nyström-Cholesky, by trial and error. Generally speaking, m should be between 10% and 50% of n, depending on the problem. For Nyström-Cholesky, quite good results are obtained if such a value for m makes π^* be very close to $\pi(x)$.

The results are in Table 3.2, which shows CPU time per iteration multiplied by the autocorrelation time for the standard MCMC method, and for other methods the ratio of this with the standard method. Table 3.3 shows actual autocorrelation time and CPU time per iteration for each experimental run.

From these results, we see that Subset of Data is overall the most reliable method for forming a π^*. We can almost always find a SOD type of π^* that leads to more

#	Length scale		p	n	m			Autocorrelation time × CPU time per iteration			
	size	type			SOD	NYS	TMP	T_{STD}	$T_{\text{SOD}}/T_{\text{STD}}$	$T_{\text{NYS}}/T_{\text{STD}}$	$T_{\text{TMP}}/T_{\text{STD}}$
1	small	isotropic	1	300	40	30	40, 20	0.76	0.45	0.51	1.05
2	small	isotropic	5	300	150	-	100, 50	1.62	0.81	-	0.14
3	small	ARD	5	300	100	-	90, 45	3.39	0.83	-	0.36
4	long	isotropic	5	300	150	120	130, 65	2.05	0.81	0.97	0.69
5	long	ARD	5	300	90	80	100, 50	5.23	0.66	0.85	0.51
6	small	isotropic	1	900	60	90	60, 30	9.06	0.27	0.23	0.28
7	small	isotropic	5	900	300	-	-	18.17	0.51	-	-
8	small	ARD	5	900	100	-	-	25.47	0.43	-	-
9	long	isotropic	5	900	100	110	-	16.86	0.34	0.40	-
10	long	ARD	5	900	300	90	-	47.46	0.67	0.34	-

Table 3.2: Results of experiments on the ten datasets.

#	CPU time (s) per iteration				Autocorrelation time			
	STD	SOD	NYS	TMP	STD	SOD	NYS	TMP
1	0.26	0.078	0.11	0.15	2.90	4.32	3.53	5.40
2	0.28	0.14	-	0.13	5.77	9.32	-	1.67
3	0.56	0.23	-	0.14	6.09	11.98	-	8.63
4	0.13	0.072	0.15	0.09	15.62	23.04	12.88	16.56
5	0.49	0.19	0.41	0.13	11.16	18.07	10.89	20.37
6	3.10	0.53	0.83	0.61	2.92	4.63	2.48	4.21
7	3.76	0.82	-	-	4.83	11.24	-	-
8	7.21	1.48	-	-	3.53	7.38	-	-
9	1.81	0.69	0.91	-	9.33	8.27	7.40	-
10	5.66	1.95	1.75	-	8.39	16.18	9.14	-

Table 3.3: CPU time per iteration and autocorrelation time for each run in Table 3.2.

efficient MCMC than the standard method. Depending on the problem, mapping to a discretizing chain using such a π^* can be two to four times faster than standard MCMC, for the Gaussian Process regression problems we tested. The computational savings go up when the size of the dataset increases. This is likely because when n is small, evaluation of π is fast, so overhead operations (especially those not related to n) are not trivial in comparison. The computational saving of π^* compared to π will be then less than the m^3 to n^3 ratio we expect from SOD for large n. Also when n is small, time to compute C (proportional to pn^2) may be significant, which also reduces the computational savings from a π^* based on SOD.

For some datasets, we can find a Nyström-Cholesky π^* with a small m that can approximate π well, in which case this method works very nicely. However, for datasets with small length scales with $p = 5$, in order to find a working π^* we have to set m to be around 95% of n or greater, making π^* as slow as, or even slower than π. This is due to the fact that when the length scale parameters for the GP are small, the covariance declines rapidly as the input variable changes, so x and x' that are even moderately far apart have low covariance. As a result, we were not able to find efficient mapping method using Nyström-Cholesky with performance even close to standard MCMC (so no result is shown in the table). On the other hand, when the length scale is large, a good approximation can be had with a small m (as small as 10% of n). For $n = 900$ and $p = 5$ with ARD covariance, Nyström-Cholesky substantially outperforms SOD.

3.6.3 Experiments with tempered transitions

We have seen in the previous section that the method of mapping to a discretizing chain has a lot of tuning parameters, and finding the optimal combination of these tuning parameters is not easy. The method of tempered transitions actually has more tuning parameters. To start with, we have to decide the number of "layers" (we call each of $\hat{T}_i$ or $\check{T}_i$ a "layer"). For each layer, (e.g. $\hat{x}_i \xrightarrow{\hat{T}_{i+1}} \hat{x}_{i+1}$), we have to decide how many MCMC

updates to simulate. This reduces the attraction of tempered transitions, but in some situations it does improve sampling efficiency.

In the experiments for the method of mapping to a discretizing chain, the results given by both SOD and Nyström-Cholesky for datasets with $n = 300$ and $p = 5$ are less satisfatory compared to others. We tried tempered transitions with these datasets. For simplicity, we used just two layers, each of which uses SOD to form the transition. We also use the slice sampler for these experiments. The number of observations in each subset (denoted as m_i for transition $\hat{T}_i$ and $\check{T}_i$) is listed in Table 3.2 under the column "TMP" and the time ratio results are under the column "$T_{\text{TMP}}/T_{\text{STD}}$". We can see that for all these datasets, tempered transitions outperform the method of mapping to a discretizing chain, sometimes substantially. We also tried the transition method on datasets with small, isotropic length scales (#1 and #6). The results from dataset #6 show great improvement of the mapping to tempered transitions method over others. For dataset #1, however, we were not able to find a combination of tuning parameters which lead to improved performance. This is likely because when n is relatively small and $p = 1$, the overhead operations of tempered transitions, compared to other methods, are too heavy to be ignored.

The advantage of tempered transitons is further illustrated in Figure 3.2, which shows the sample autocorrelation plots of the log likelihood for both methods, on dataset #2.

We also attempted to apply the tempered transitions method to datasets with $n = 900$. However, due to the fact that there are too many tuning paramters for these methods, and when n is large the number of choices increase enormously, we unfortunately were unable to obtain MCMC methods based on the tempered transition method that outperforms the other methods. Since we can relatively easily obtain better performance using the method of mapping to discretizing chains with SoD or Nyström approximation for $n = 900$, we recommend using these methods when n is large.

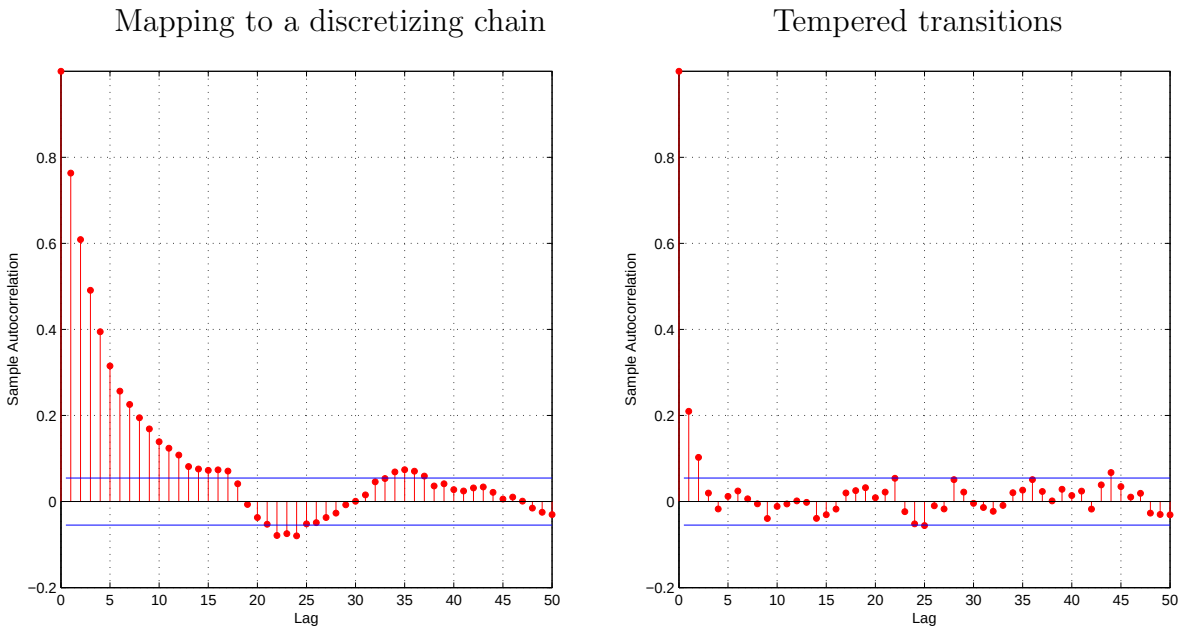

Figure 3.2: Comparison of autocorrelation times of the log likelihood for MCMC runs using mapping to a discretizing chain and using tempered transitions. Dataset #2 is used (with five covariates, small length scales, an isotropic covariance function, and 300 observations). Note that CPU time per iteration is almost the same for the two methods, so no adjustment is needed.

Chapter 4

Discussion and future work

This thesis makes two major contributions. In Chapter 2, we propose a new heteroscedastic Gaussian Process regression method (GPLC). By adding a latent variable as an unobserved input, GPLC has the ability to handle regression problems with residuals having non-constant variance. GPLC is also a more general model in that it doesn't require the assumption that the residuals are normally distributed. We also extend Goldberg *et al.* (1998)'s GPLV model to a full Bayesian version, and develop a new MCMC method for it. We demonstrate in Section 2.4 that, under certain conditions, the standard GP regression model and the GPLV model can be viewed as special cases of the GPLC model. Extensive experiments show that GPLC model is generally as good as the GPLV model in terms of mean squared error and negative log probability density when the residuals are normally distributed. When the residuals are non-normal, GPLC is better then GPLV. Both methods give better results than a standard homoscedastic Gaussian Process regression model.

Heteroscedasticity can be viewed as a form of non-stationarity — think of $y(x)$ as a random process — if the residual variance changes when x changes, then clearly $y(x)$ is non-stationary. In many scientific fields such as atmospheric science, the assumption of stationarity (which many models rely on) is often violated. Popular spatical statisti-

cal models that handle non-stationarity include "spatial deformation", due to Sampson and Guttorp (1992) and Schmidt and O'Hagan (2003), and "dimension expansion" by Bornn, *et al.* (2012). These methods, however, only focus on the non-stationarity in the propterties of the regression function which gives the expected value of the response, but not the non-stationarity in the residuals (heteroscedasticity). It would be interesting to combine these non-stationary methods with GPLC to get a model that can handle both kinds of non-stationarity.

In Chapter 3, we introduce two classes of MCMC methods using the "mapping and caching" framework: the method of mapping to a discretizing chain, and the tempered transition method. Our experiments indicate that for the method of mapping to a discretizing chain, when an appropriate π^* is chosen (e.g. SOD approximation of π with an appropriate m), an efficient MCMC method can be constructed by making "local" jumps (e.g. setting $r = s = 1$). A good MCMC method can also be constructed using tempered transitions, with a small number of π_i, where each $\hat{T}_i$ and $\check{T}_i$ makes only a small update.

These results are understandable. Though π^* and π_i, are broader than π, making small adjustments a small number of times will have a good chance to still stay in a high probability area of π. However, even though the acceptance rate is high, this strategy of making small adjustments cannot bring us very far from the previous state. On the other hand, if we make large jumps, for instance, by using large values for r and s in the method of mapping to a discretizing chain, the acceptance rate will be low, but when a proposal is accepted, it will be much further away from the previous state, which is favourable for a MCMC method. We haven't had much success using this strategy so far, perhaps due to the difficulty of parameter tuning, but we believe this direction is worth pursuing. The tempered transition method may be more suitable for this direction, because moving from one state to another state further away is somewhat similar to moving among modes — the sequence of $\hat{T}_i$ and $\check{T}_i$ should be able to "guide" the transition back to a region with high probability under π.

Bibliography

Adams, R. and Stegle, O. (2008) "Gaussian process product models for nonparametric nonstationarity", *Proceedings of the 25th International Conference on Machine Learning*, pp. 1–8.

Bishop, C. M., (2007) *Pattern Recognition and Machine Learning* Springer. ISBN: 0387310738.

Bornn, L., Shaddick, G. and Zidek, J. V. (2012) "Modeling nonstationary processes through dimension expansion", *Journal of the American Statistical Association*, Vol 107, No. 497

Casella, G. and Robert, C. (2005) *Monte Carlo Statistical Methods*, Springer, ISBN: 0387212396.

Drineas, P. and Mahoney, M. W., (2005) On the Nystrom method for approximating a Gram matrix for improved kernel-based learning, *Journal of Machine Learning Research*, vol 6, pp. 2153–2175.

Gangadharan, M., Soares, C. G., and Lucas, C. S. G. (2011), "Sea Level Rise, Coastal Engineering, Shorelines and Tides", Linda. L.Wright (editor) *Characteristic and Moment Generating Functions of Generalised Extreme Value Distribution*, Nova Science Publishers, pp. 269–276.

Goldberg, P. W. and Williams, C. K. I. and Bishop, C.M. (1998) "Regression with Input-

dependent Noise: A Gaussian Process Treatment", *Advances in Neural Information Processing Systems 10*, pp. 493–499.

Hastings, W.K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications", *Biometrika* 57 (1) pp. 97–109.

Kersting, K, Plagemann, C, Pfaff, P., and Burgard, W. (2007) "Most likely heteroskedastic gaussian process regression", *Proceedings of the 24th International Conference on Machine Learning*, pp 393–400.

Kolar, M. and Sharpnack, J. (2012) "Variance Function Estimation in High-dimensions", *Proceedings of the 29th International Conference on Machine Learning*, pp 1447–1454.

Lazaro-Gredilla, M. and Titsias, M. (2011) "Variational heteroscedastic gaussian process regression", *Proceedings of the 28th International Conference on Machine Learning*, pp 841–848.

Leadbetter, M.R., Lindgren, G. and Rootzén, H. (1983). *Extremes and Related Properties of Random Sequences and Processes*, Springer-Verlag, ISBN: 0387907319.

Liu, J. (2009) *Monte Carlo Strategies in Scientific Computing*, Springer, ISBN: 0387763694.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. (1953) "Equations of State Calculations by Fast Computing Machines", *Journal of Chemical Physics*, vol. 21, pp. 1087–1092.

Mooij, J. M., Stegle, O., Janzing, D., Zhang, K., and Schölkopf, B. (2010) "Probabilistic latent variable models for distinguishing between cause and effect", *Advances in Neural Information Processing Systems 23*, pp. 1687–1695.

Murray, I., Adams, R. P., and MacKay, D. J. C. (2010) "Elliptical slice sampling",

The Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), vol 9, pp541–548.

Neal, R. M. (1993), "Probabilistic Inference Using Markov Chain Monte Carlo Methods", Technical Report, Dept. of Computer Science, University of Toronto, CRG-TR-93-1. Available from `http://www.utstat.utoronto.ca/~radford`

Neal, R. M. (1996a) *Bayesian Learning for Neural Networks*, Lecture Notes in Statistics No. 118, New York: Springer-Verlag

Neal, R. M. (1996b) "Sampling from multimodal distributions using tempered transitions", *Statistics and Computing*, vol. 6, pp. 353–366.

Neal, R. M. (1997), "Monte Carlo implementation of Gaussian process models for Bayesian regression and classification", Technical Report, Dept. of Statistics, University of Toronto, no. 9702. Available from `http://www.utstat.utoronto.ca/~radford`

Neal, R. M. (1998), "Regression and Classification Using Gaussian Process Priors", Bernardo, J. M. (editor) *Bayesian Statistics 6*, Oxford University Press, pp. 475–501.

Neal, R. M. (2003), "Slice sampling", *Annals of Statistics*, vol. 11, pp. 125–139.

Neal, R. M. (2006), "Constructing Efficient MCMC Methods Using Temporary Mapping and Caching", Talk at Columbia University, December 2006. Available from `http://www.utstat.utoronto.ca/~radford`

Neal, R. M. (2010), "MCMC using ensembles of states for problems with fast and slow variables such as Gaussian Process Regression", Technical Report, Department of Statistics, University of Toronto. Available from `http://www.utstat.utoronto.ca/~radford`

Neal, R. M. (2011), "MCMC using Hamiltonian dynamics", Chapter 5 of *The Handbook of Markov Chain Monte Carlo*, Chapman & Hall / CRC Press, ISBN: 1420079417.

Nyström, E. J., (1930) Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben, *Acta Mathematica*, 54, pp. 185–204.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., (1992) *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press. p. 73. ISBN 0521431085.

Quadrianto, N., Kersting, K., Reid, M., Caetano, T., and Buntine, W. (2009), "Kernel conditional quantile estimation via reduction revisited". *9th IEEE International Conference on Data Mining*.

Quiñonero-Candela, J., Rasmussen, C. E., and Williams, C. K. I. (2007), "Approximation Methods for Gaussian Process Regression", Technical Report MSR-TR-2007-124, Microsoft Research.

Rasmussen, C. E. and Williams, C. K. I. (2006), *Gaussian Processes for Machine Learning*, the MIT Press, ISBN 026218253X.

Reinhardt, H. J. (1985) *Analysis of Approximation Methods for Differential and Integral Equations*, Springer, New York.

Sampson, P. D. and Guttorp, P. (1992), "Nonparametric estimation of nonstationary spatial covariance structure", *Journal of the American Statistical Association*, Vol 87, pp 108–119.

Schmidt, A., M. and O'Hagan, A. (2003), "Bayesian inference for nonstationary spatial covariance structure via spatial deformation", *Journal of the Royal Statistical Society, B 65. part 3*, pp. 743–758

Schmidt, G., Mattern, R. and Schueler, F. (1981) "Biomechanical investigation to determine physical and traumatological differentiation criteria for the maximum load capacity of head and vertebral column with and without protective helmet under the

effects of impact", *EEC Research Program on Biomechanics of Impacts, Final report, Phase III, Project G5*, Institut fur Rechtsmedizin, University of Heidelberg, West Germany.

Sankaran , M. (1959). "On the non-central chi-squared distribution", *Biometrica* vol. 46, no. 1/2, June, 1959.

Silverman, B.W. (1985) "Some aspects of the spline smoothing approach to non-parametric curve fitting", *Journal of the Royal Statistical Society* B, 47, no. 1, pp. 1–52.

Sherman, J. and Morrison, W. J. (1950). "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix", *Annals of Mathematical Statistics*, vol 21, no. 1, pp. 124–127.

Thompson, M., (2011). "Slice sampling with multivariate steps", Doctoral Thesis, Department of Statistics, University of Toronto.

Wang, C. and Neal, R. M. (2012) "Gaussian process regression with heteroscedastic or non-Gaussian residuals", Technical Report, Department of Statistics, University of Toronto.

Wang, C. and Neal, R. M. (2013) "MCMC methods for Gaussian process models using fast approximations for the likelihood", Technical Report, Department of Statistics, University of Toronto.

Williams, C.K.I. and Seeger, M. (2001), "Using the Nyström Method to Speed up Kernel Machines", *Advances in Neural Information Processing Systems 13*, pp. 682–688.

Wilson, A. G. and Ghahramani, Z. (2010) "Copula processes", *Advances in Neural Information Processing Systems 23*, pp. 2460–2468.

Wilson, A. G., Knowles, D. A. and Ghahramani, Z. (2012) "Gaussian Process Regression Network", *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*

Woodbury, M.A., (1950) "Inverting modified matrices", Memorandum Rept, no. 42, Statistical Research Group, Princeton University, Princeton, NJ.

Appendix A

MSE and NLPD values for all experiments

		NLPD			MSE		
	Training Set	REG	GPLC	GPLV	REG	GPLC	GPLV
Dataset U0:	1	**0.2945**	0.2953	0.2977	0.0105	**0.0104**	0.0105
	2	**0.2599**	0.2621	0.2613	**0.0052**	0.0054	0.0054
	3	0.3311	0.3290	**0.3239**	**0.0156**	0.0164	**0.0156**
	4	0.2800	**0.2796**	0.2820	0.0093	**0.0092**	0.0094
	5	0.2708	0.2707	**0.2703**	**0.0067**	0.0068	**0.0067**
	6	0.2880	0.2877	**0.2854**	0.0099	0.0097	**0.0095**
	7	0.3052	0.3051	**0.3016**	**0.0028**	0.0031	0.0029
	8	0.3039	**0.2965**	0.2997	0.0109	**0.0104**	0.0107
	9	**0.2555**	0.2573	0.2596	0.0043	0.0044	**0.0042**
	10	**0.2533**	0.2596	0.2603	**0.0032**	**0.0032**	0.0037
Dataset U1:	1	0.3364	**0.2459**	0.2480	0.0119	0.0088	**0.0059**
	2	0.3259	0.2489	**0.2076**	0.0077	**0.0039**	0.0058
	3	0.3142	0.2774	**0.2661**	**0.0060**	0.0144	0.0139
	4	0.3198	0.2273	**0.2237**	0.0077	0.0065	**0.0058**
	5	0.3231	0.2296	**0.2185**	0.0076	**0.0071**	0.0073
	6	0.3596	0.2397	**0.2321**	0.0137	**0.0105**	0.0110
	7	0.3404	0.2696	**0.2374**	0.0040	**0.0030**	**0.0030**
	8	0.3683	**0.3143**	0.3305	0.0081	**0.0079**	0.0080
	9	0.3177	**0.2023**	0.2172	0.0061	0.0057	**0.0055**
	10	0.2961	0.3050	**0.2107**	**0.0017**	0.0018	0.0020
Dataset U2:	1	0.2878	**0.1976**	0.2408	0.0070	0.0035	**0.0034**
	2	0.2616	**0.1099**	0.1349	0.0026	0.0009	**0.0007**
	3	0.2527	**0.1123**	0.1488	**0.0013**	0.0025	0.0035
	4	0.2697	**0.1148**	0.1998	0.0042	**0.0036**	0.0037
	5	0.2695	**0.1275**	0.1985	**0.0030**	0.0036	**0.0030**
	6	0.2599	**0.1396**	0.1769	**0.0025**	0.0052	0.0032
	7	0.2544	**0.1130**	0.1948	**0.0010**	0.0015	0.0017
	8	0.2708	**0.0811**	0.1283	0.0046	0.0019	**0.0017**
	9	0.2863	**0.0833**	0.1353	0.0049	**0.0020**	0.0027
	10	0.2727	**0.1245**	0.1670	0.0033	0.0023	**0.0021**

		NLPD			MSE		
	Training Set	REG	GPLC	GPLV	REG	GPLC	GPLV
Dataset M0:	1	**0.3097**	0.3167	0.3098	**0.0147**	0.0164	**0.0147**
	2	**0.2558**	0.2593	0.2565	0.0096	0.0100	**0.0093**
	3	**0.2725**	0.2816	0.2786	**0.0127**	0.0135	0.0131
	4	**0.3077**	0.3166	0.3149	**0.0142**	0.0157	0.0156
	5	0.3478	0.3539	**0.3439**	0.0347	0.0374	**0.0335**
	6	**0.3075**	0.3107	0.3108	**0.0226**	0.0228	0.0229
	7	**0.2897**	0.2945	0.2924	0.0137	0.0140	**0.0128**
	8	**0.2758**	0.2762	0.2823	**.0125**	0.0127	0.0133
	9	0.2874	**0.2871**	0.2901	**0.0154**	0.0166	0.0165
	10	**0.2700**	0.2858	0.2817	**0.0109**	0.0131	0.0113
Dataset M1:	1	0.4726	**0.3202**	0.3407	0.0186	**0.0120**	0.0201
	2	0.3852	0.3046	**0.2860**	0.0121	0.0113	**0.0098**
	3	0.4560	**0.3390**	0.3410	0.0305	**0.0247**	0.0277
	4	0.4300	0.3860	**0.3751**	0.0204	**0.0188**	0.0201
	5	0.4817	0.4321	**0.3906**	0.0426	0.0399	**0.0351**
	6	0.4668	0.3603	**0.3024**	0.0364	0.0247	**0.0163**
	7	0.4282	**0.3656**	0.3799	0.0195	0.0172	**0.0171**
	8	0.4184	**0.3198**	0.4022	0.0197	**0.0148**	0.0217
	9	0.4161	**0.3281**	0.3817	**0.0202**	0.0204	0.0297
	10	0.4253	0.3269	**0.3201**	0.0216	**0.0190**	0.0197
Dataset M2:	1	0.3736	**0.2413**	0.298	0.0099	**0.0092**	0.0093
	2	0.3934	**0.2458**	0.2965	0.0136	**0.0099**	0.0122
	3	0.4015	**0.2695**	0.2832	0.0167	**0.0105**	0.0128
	4	0.4819	**0.3636**	0.4202	0.0391	**0.0174**	0.0489
	5	0.4311	**0.3257**	0.3548	0.0269	0.0234	**0.0230**
	6	0.4348	**0.2678**	0.3140	0.0216	**0.0165**	0.0176
	7	0.3892	**0.2479**	0.2770	0.0102	**0.0065**	0.0083
	8	0.3709	0.3147	**0.2580**	0.0058	**0.0052**	0.0067
	9	0.4043	**0.2441**	0.2899	0.0156	**0.0124**	0.0146
	10	0.4718	**0.3355**	0.3923	0.0403	**0.0223**	0.0281