Gaussian process regression and classification with elliptical slice sampling

Ricardo Marques U16077131

WST795 Research Report

Submitted in partial fulfillment of the degree BSc(Hons) Mathematical Statistics

Supervisor: A. de Waal

Department of Statistics, University of Pretoria

October 1, 2020

Abstract

The use of Gaussian processes to perform classification and regression is considered, however Bayesian inference can become difficult to perform as, Bayes theorem requires a closed form solution of an integral which is often intractable. This issue is addressed through the use of the probabilistic approach in sampling, particularly Markov chain Monte Carlo inference.

Within Markov chain Monte Carlo inference, an explanation as to why this is a suitable approach such that the difficulties arising from Bayes theorem and it's implementation are avoided. In addition to this, a comparison of the degree to which several Markov chain Monte Carlo techniques address this obstacle is provided.

Declaration

I, *Ricardo Daniel Marques Salgado,* declare that this essay, submitted in partial fulfillment of the degree *BSc(Hons) Mathematical Statistics,* at the University of Pretoria, is my own work and has not been previously submitted at this or any other tertiary institution.

_____

Ricardo Daniel Marques Salgado

_____

Alta de Waal

_____

Date

Acknowledgements

Contents

List of Figures

1 Introduction

There are several common questions which can be answered through the application of statistical techniques on data, either simulated or collected through experimental procedure. One such question is that of performing a hypothesis test, in which the belief that said data follows an assumed, null, distribution is tested. Another question is that of point estimation, where the exact distribution of the parameter of interest is unknown, yet it is believed to belong to a particular family of distributions. For each question above there are several techniques which may be applied. Such techniques include maximum likelihood estimation, minimisation of residuals and Monte Carlo methods.[1]

1.1 Research Objective

This research aims to investigate GPs in both a regression and classification context. In particular MCMC will be considered to calculate the GP posterior distribution. Various MCMC algorithms, will be implemented in Python[2]. The following algorithms will be considered: Metropolis - Hastings, Gibbs sampling and elliptical slice sampling. Once implemented, the aim is to draw conclusions regarding the shortfalls and appropriateness of each algorithm individually and comparatively. The evaluation metrics which will be used in order to draw conclusions regarding the algorithms include accuracy, computational speed, effective samples and the ability to handle large datasets as well as high dimensions. However, before the above mentioned Markov chain Monte Carlo (MCMC) inference can be introduced, Bayesian preliminaries [8] are required as these form part of the foundation of MCMC inference.

1.2 Bayesian Preliminaries

The fundamental theorem used throughout Bayesian inference is that of Bayes theorem [8]. Bayes theorem is the combination of several laws of probability, namely the laws of conditional probability, total probability and the law of multiplication of probabilities. This combination of the above laws produces Bayes theorem as:

$$\begin{aligned} p(X = x|Y = y) &= \frac{p(X = x, Y = y)}{p(Y = y)} \\ &= \frac{p(X = x)p(Y = y|X = x)}{\sum_{x'} p(X = x)p(Y = y|X = x')}. \end{aligned}$$

The importance of this theorem follows from the fact that it enables the calculation of the posterior distribution, namely $p(\theta|D)$, where θ is the parameter of interest, and D is the observed data. In

[1] Cliburn Chan and Janice McCarthy, "Computational problems in statistics", Duke University, retrieved on May 1, 2019 from `https://people.duke.edu/~ccc14/sta-663/ComputationalStatisticsMotivation.html`

[2] Python Software Foundation, "Python Language Reference", version 3.7, retrieved on May 3, 2019 from `http://www.python.org`

accordance with Bayes theorem, the posterior distribution is defined as [3]:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}.$$

The above result is useful, as the posterior distribution encapsulates all that is known regarding the unknown parameter θ [8]. This formula above can also be simplified into the following:

$$Posterior \propto Likelihood \times Prior.$$

The prior distribution represents the initial belief regarding the behaviour of the parameter of interest, before the observation of any data occurs. The likelihood function then updates this distribution, as data regarding the parameter of interest is observed. Thus, the posterior distribution is defined as the conditional distribution of the parameter of interest, θ, given the observed data as described mathematically above. However, this is not of particular use when the objective is to predict future values, but rather the predictive posterior is. The predictive posterior distribution enables the prediction of future values, and no longer depends upon θ, as θ is integrated out of the above equation to provide the following [8]:

$$p(x|D) = \frac{p(x, D)}{p(D)}.$$

One issue that arises when attempting to solve this equation is due to the nature of the denominator, $p(D)$, which contains the marginal density of all of the observations. In order to calculate $p(D)$, the closed form solution of the following needs to be determined: $p(D) = \int p(x, D)dx$.

However, this integral is often either intractable[3] or computationally intensive to evaluate [4]. An example of such is in the work of Poynor and Kottas [12], in which they model the mean residual life function of individuals[4]. In their work, the exponential Weibull distribution was used to generate data for simulation, but this predictive posterior cannot be evaluated in a closed form, and this is where sampling and Markov chain Monte Carlo inference in particular become of use.

In conclusion should one wish to perform Bayesian inference there are several requirements that need consideration. Firstly, the prior distribution is required. This distribution embodies the assumed information and hence behaviour of the parameter(s) of interest. Following this, observed data is then required. It is this data that will then be used in order to determine the likelihood distribution. The likelihood distribution is then updated as additional observations are obtained. Lastly, a decision as to which posterior distribution is required, either simply the posterior or the predictive posterior. Once this decision is made, the relevant equation from above may be applied.

[3]Intractable refers to the fact that the integral has no closed-form solution.
[4]The expected remaining lifetime of an individual, given he/she has survived up to a particular point in time.

1.3 Markov Chain Monte Carlo Inference

The motivation behind applying MCMC inference in order to obtain the predictive posterior is that MCMC algorithms enable the estimation of the predictive posterior. MCMC methods ensure the closed form solution of $p(D)$ is not required as the target density of interest will simply be approximated [8] rather than solved for explicitly. This ensures the circumvention of this problematic task. MCMC is preferred within this paper over more traditional Monte Carlo methods, such as rejection sampling [1]. This preference is due to the ability of MCMC to better handle high-dimensional data. That is, MCMC is recognised as the only general technique in which a solution can be obtained in reasonable time should the data be high-dimensional [1].

In order to execute an MCMC algorithm, both the prior distribution and the likelihood distribution are required. The latter is obtained directly from the observed data itself, which is a straight forward task but issues arise with the prior distribution, as it must be chosen based on the information available. For the context of this paper, it will be assumed that prior distribution is Gaussian and that the data of interest is continuous.

Now that the prior distribution is defined as Gaussian, the calculation or rather the approximation of the predictive posterior can occur. In order to approximate the predictive posterior, a decision must be made as to which MCMC algorithm is most appropriate. The process of MCMC can illustrated as follows:

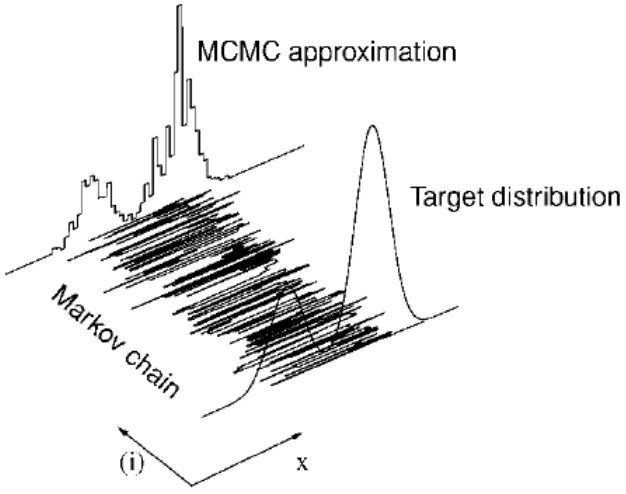

Figure 1: Visualisation of MCMC inference taken from [1].

Figure 1 illustrates how a particular target density of interest is approximated by MCMC inference. This approximation is determined through the application of a Markov chain onto the state space of the target density and is represented by the histogram. This histogram is built through the use of a sampling procedure, as described by any one of the MCMC algorithms that exist.

The following algorithms will form the focus of this paper, and hence will be illustrated and compared.

- **Metropolis - Hastings (MH) algorithm**. MH is considered to be the most widely used MCMC algorithm, while many other MCMC algorithms are also considered to be simply extensions, or special cases of MH [1]. The fundamental idea of MH is that at each iteration, a proposal is made. The proposal being to move from the current state, represented by X, to some new state X'. Each proposal can then be either rejected, or accepted, based on some criteria [8]. Lastly, attached to each proposal will be the probability of the transition, denoted by $q(X'|X)$, defined by q, the proposal distribution. One important variable within any MH algorithm is that of the proposal width, also referred to as the step-size. The importance of the size-size arises due to the fact should an inappropriate step-size be selected convergence of the algorithm to the target density may not only be slow, but may not occur should the number of iterations not be large enough [8, 9].

- **Gibbs sampling**. The fundamental idea behind Gibbs sampling follows closely to that of MH described above, as Gibbs sampling is considered to be one of the special cases of MH [1]. Gibbs sampling can be illustrated in the case for which there are 2 variables of interest, that is $D = 2$, as:

  - $x_1^{s+1} \sim p(x_1|x_2^s)$
  - $x_2^{s+1} \sim p(x_2|x_1^{s+1})$

  Using Gibbs, each variable is sampled sequentially, with each variable being conditioned upon the rest of the variables available within the distribution [8], at the time of the iteration. Where $\boldsymbol{x}^s$ represents a given joint sample of all variables, and $\boldsymbol{x}^{s+1}$ is the new sample generated. It is crucial to note that the choice of D here is arbitrary and thus the above can be extended to any D number of variables. However, as will be shown later, both Gibbs sampling and MH perform inadequately under certain circumstances [9].

- **Elliptical slice sampling**. Thus, due to the difficulties that can develop when implementing a MH algorithm with an inappropriate step-size, as well as due to the fact that Gibbs sampling is considered to perform inadequately when the prior is Gaussian [9], an alternative was required. Elliptical slice sampling was thus developed with the aim of addressing these issues when considering a Gaussian prior [9], but also as an alternative to MH. The approach of elliptical slice sampling (ESS) follows directly from slice sampling which is described as an auxiliary variable algorithm [1]. The implications of the introduction of this auxiliary variable into the sampling procedure will be discussed further in Section 3.5, with the focus now solely on ESS, rather than its' preliminary slice sampling.

  The fundamental idea of ESS is described as follows [9]; an ellipse described by $\nu \sim N(\mathbf{0}, \boldsymbol{\Sigma})$ is randomly generated. Secondly, the log-likelihood threshold is defined. Next, the algorithm proposes

an angle defined as $\theta \sim Uniform[0, 2\pi]$. Once θ has been obtained, the acceptance rejection criteria as per [9] is tested, incorporating the log-likelihood threshold. If the value satisfies the criteria, the new state is then returned. Otherwise the bracket described by θ is first narrowed, and then a new θ is drawn with a uniform distribution, defined over this new, smaller region. This process continues until such time as the required number of samples values has been acquired.

The above mentioned MCMC methods will be applied to Gaussian processes (GPs). GPs are a non-parametric modelling technique for both regression and classification. A GP describes the prior distribution over a set of functions [8]. As per the equation for $p(\theta|D)$ introduced earlier, this prior can be then be used to calculate the posterior distribution over the same set of functions, as data, describing the likelihood distribution, is observed.

1.4 Literature Review

The following is a description of the literature that was investigated for the purpose of this research. It is however, not an exhaustive list of either all existing literature on the relevant topics, or all of the literature that was reviewed for this research. However, this section does refer to the main references that form the backbone of this research.

1.4.1 Gaussian Processes

With the structure of this research described above in Section 1, a similar approach was followed to investigate the relevant topics. Firstly the work of Murphy [8], which this research heavily relies upon, was consulted. His work regarding Gaussian processes; their formal definition as well as the implementation of them when considering both noisy and noise-free data was considered. Next, having covered the theoretical definitions and notation of GPs, a more intuitive explanation was required and hence provided by the work of Bailey[5]. Here, she combines the explicit definitions as per [8] with her own examples and explanations. Thus, the two above sources provided all the necessary information regarding GPs to progress further.

Within [8], Murphy heavily discusses kernel, covariance, functions hand in hand with GPs. Hence, his work on kernel functions was next referenced. Having consulted his work and due to the focus of this research being on the implementation of Gaussian processes, the decision to only make use of the squared exponential kernel was made. The mathematical definition of this kernel function, as well as how it operates as a measure of similarity, as all kernel functions do, was also of interest.

Having obtained the foundation of GPs as a whole, further interest was given to GPs in specific contexts. Namely in a regression setting and a classification setting [8]. In order to gain further insight

[5]Katherine Bailey, "Gaussian Processes for Dummies", retrieved on March 25, 2019 from `https://katbailey.github.io/post/gaussian-processes-for-dummies/`

into the above areas, the work of Rasmussen in [13], was consulted. His derivation of the predictive posterior, combined with the derivation of the marginal and conditional distributions of multivariate Gaussian distributions [8] were examined, as they will both be made use of later.

1.4.2 Markov Chain Monte Carlo Inference

The focus of our research then turns to MCMC inference, from several sources. However, the theoretical overview of MCMC inference along with the motivation for it's use, and understanding as to why it works were supplied in [8]. In order to better understand MCMC and why preference was given to MCMC over perhaps more traditional methods in Monte Carlo inference, [1] was then reviewed. It was here, that due to the aim of this research being to address data that is high-dimensional the reasoning for this particular preference was founded. Lastly, when considering MCMC it was important to obtain a practical illustration of MCMC operating, particularly in Python. It was in the work of Wiecki[6] that such an example was found. Here Wiecki provides a "layman's" explanation to MCMC inference, while showing the process in action by considering the MH algorithm and a normal-normal model of which he aims to obtain the posterior mean for. It is also here that the issues of step-size choice are clearly demonstrated, but so too is a comparison of an algorithm coded from first principles versus an algorithm implemented through the use of the Python package PYMC3 [14].

As MH is not the only algorithm of interest, the various other algorithms all of whom will be later implemented and compared were also investigated. Thus, next the definition of Gibbs sampling, its' intuition and limitations were all studied with the aid of [8]. Then, elliptical slice sampling became the focus of the research. However, upon beginning research into elliptical slice sampling it became apparent that an understanding of its' preliminary, slice sampling, was required. The work of Neal [10] was consulted for this. Here, the definition of slice sampling as an auxiliary variable algorithm, as well as to what an auxiliary variable algorithm is were provided. The purpose of introducing an auxiliary variable into the algorithm was also covered, that is it allows the issues that arise with MH and step-size choices to be avoided. [10], also explains that slice sampling is often easier to implement than Gibbs sampling under certain conditions which will be examined further.

Having considered its' preliminary, elliptical slice sampling was now considered. The work of Murray [9], in which he provides the motivation for the algorithm, pseudo-code of the algorithm, but most importantly the validation for the algorithm as a valid MCMC approach was of particular interest. He also illustrates the performance of the algorithm against several other algorithms, which will be a point of reference for the comparison made later in this research. Here, the first measures of comparison one may make use of to compare the above algorithms is also provided, namely computational complexity,

[6]Thomas Wiecki, "MCMC sampling for dummies", retrieved on March 29, 2019 from `https://twiecki.io/blog/2015/11/10/mcmc-sampling/`

effective samples and the likelihood evaluations required [9].

Lastly, having considered the above algorithms, attention was then given to any algorithms that have been developed more recently than ESS or that is able to directly address the issues which may arise when implementing ESS. The aim of including an alternative algorithm was to determine whether any of the shortcomings of ESS can be avoided appropriately while still addressing the problem areas of MH and Gibbs sampling as ESS does. The algorithm that was then chosen for this purpose was No-U-Turn-Sampling, (NUTS). NUTS was first introduced in 2011, while ESS was introduced in 2010. So although there is not a large difference between the time of introduction, NUTS differs greatly from the previous methods introduced [7]. These differences will be explored further later.

2 Gaussian Processes

2.1 Introduction

As mentioned before GPs are a non-parametric modelling technique for regression and classification, however in the context of this paper the use of GPs for classification will only be briefly discussed as, the focus for this paper lies on GPs and their uses within regression. A GP can be described in terms of random variables or stochastic processes. A GP as put by Rasmussen and Williams [13], is a collection of random variables $\{X_t : t \in S\}$, such that any finite subset, $\{X_{t_0}, X_{t_1}, ..., X_{t_N}\}$, has a joint Gaussian distribution. The use of t as an index here is intentional as the random variables are indexed according to the time of their occurrence, hence making the collection of random variables a stochastic process.

For illustration, if the subset of interest was defined as $\{X_{t_0}, X_{t_1}\}$, the joint distribution could be represented as a bivariate Gaussian distribution (Figure 2):

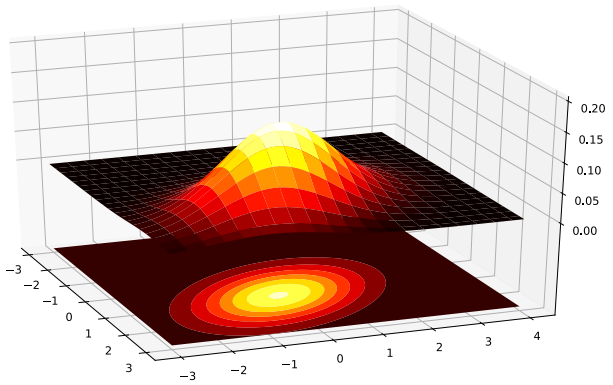

Figure 2: Bivariate Gaussian distribution

Upon inspection of the above figure, one may note that the above distribution has the appearance of a bivariate normal distribution. This is due to the fact that within distribution theory the terms Gaussian and normal are interchangeable. For the sake of uniformity throughout the remainder of this research, the term Gaussian will be preferred however.

The probability density function of the Gaussian distribution can then be represented mathematically, when considering D dimensions as, [8]:

$$N(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')'\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{x}')\Big).$$

Gaussian processes are of particular interest when working within a supervised learning context, within machine learning. Here, both a set of inputs, $\boldsymbol{x}_i$, and a set of outputs, $\boldsymbol{y}_i$, are observed. Murphy [8], noted that the relationship between the sets of outputs and inputs can be represented as $\boldsymbol{y}_i = f(\boldsymbol{x}_i)$. Should this be the case, Murphy states that, "The optimal approach is to infer a distribution over functions given the data, and then to use this to make predictions given new inputs". Hence, a Gaussian process provides the definition of the prior over functions f which is then transformed into a posterior distribution. The context of Gaussian processes, referred to as GPs from now, will be split into two sections, namely regression and classification as done in [13].

2.2 Gaussian Processes for Regression

GPs are fully specified by two parameters. Firstly $\boldsymbol{\mu}(\boldsymbol{x})$, which is the mean function of the GP and secondly $\boldsymbol{\Sigma}(\boldsymbol{x})$, referred to as the covariance function of the GP. Hence, if the prior on the regression function is a GP, it is denoted as follows:

$$f(\boldsymbol{x}) \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \tag{1}$$

where $m(\boldsymbol{x})$ is the mean function denoted as $E[f(\boldsymbol{x})]$, and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the kernel function (covariance function before). It is important to note that the kernel function is required to be positive definite, that is $k(\boldsymbol{x}, \boldsymbol{x}') > 0$. This is intuitive as a kernel function returns a measure of similarity between $\boldsymbol{x}$ and $\boldsymbol{x}'$ [8], for which negative values would be illogical.

It is important to note that a GP is expected to provide different output when considering data that is noise free, such as from a simulation, versus noisy-data, such as data arising from experimental procedure. Noise refers to unexplained variation within the data, thus if data is referred to as noisy, each observation possesses an element of randomness [8]. The opposite holds true for data which is noise-free. In the case of noise-free data, the GP is expected to interpolate values, which it has already seen without any uncertainty. While, in the case of noisy-data the GP is expected to simply be sufficiently close to the

observed data points. Both cases mentioned above will be discussed further in Sections 2.2.2 and 2.2.3 respectively.

2.2.1 Marginal and Conditional Distributions of Multivariate Gaussian Distributions

The Bayesian approach to modelling requires the following distributions: a prior, a likelihood, a posterior and lastly a marginal posterior. As GPs are based on the multivariate Gaussian distribution, the marginal posterior of a multivariate Gaussian distribution is required.

The result below follows directly from [8], where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ with a joint Gaussian distribution is being considered, whose parameters are:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}, \qquad \Sigma^{-1} = \Lambda = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix}.$$

The marginal distributions of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are:

$$p(\boldsymbol{x}_1) = N(\boldsymbol{x}_1|\boldsymbol{\mu}_1, \Sigma_{11}), \tag{2}$$

$$p(\boldsymbol{x}_2) = N(\boldsymbol{x}_2|\boldsymbol{\mu}_2, \Sigma_{22}). \tag{3}$$

The conditional posterior distribution then directly follows from (2) and (3) as:

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = N(\boldsymbol{x}_1|\boldsymbol{\mu}_{1|2}, \Sigma_{1|2}), \tag{4}$$

where:

$$\begin{aligned} \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \Lambda_{11}^{-1}\Lambda_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \\ &= \Sigma_{1|2}(\Lambda_{11}\mu_1 - \Lambda_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)), \end{aligned}$$

with

$$\begin{aligned} \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \\ &= \Lambda_{11}^{-1}. \end{aligned}$$

2.2.2 Noise-Free Data for Predictions

In the instance of noise-free observations, suppose $T = \{(X_i, f_i), i = 1, ..., N\}$ is defined as the training set, in which $f_i = f(x_i)$ is the observed value of the function at x_i, which is free of any noise. The aim as described in [8] by Murphy, is to then predict the outputs of said function f_*, given a training set T, a training input X, and lastly a test set denoted as X_*.

Thus, by the definition of a GP, the joint distribution of interest is as follows:

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim N\left[\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix} \quad , \quad \begin{pmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{pmatrix} \right].$$

Then by (4), it follows that the posterior distribution is represented as:

$$p(f_*|X_*, X, f) = N(f_*|\boldsymbol{\mu}_*, \Sigma_*), \tag{5}$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(X_*) + k(X_*, X)k(X, X)^{-1}(f - \boldsymbol{\mu}(X)), \tag{6}$$

$$\Sigma_* = k(X_*, X_*) - k(X_*, X)k(X, X)^{-1}k(X, X_*). \tag{7}$$

Lastly, should the GP be used to forecast a $f(x)$ value for a x value previously observed by the GP, $f(x)$ will be returned with 100% confidence as the value was previously observed and hence is known.

Figure 3 represents three samples from a Gaussian prior with the use of a Gaussian kernel and Figure 4 illustrates the effect of conditioning these prior samples on 5 noise-free observations.

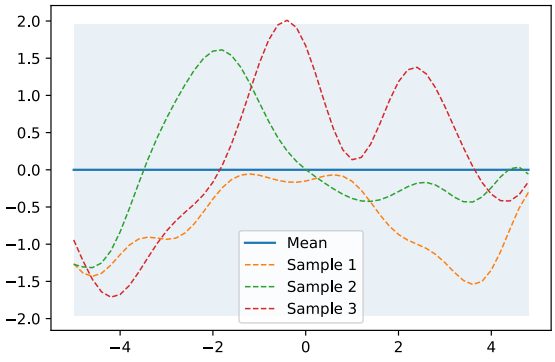

Figure 3: Three function samples from a Gaussian process prior

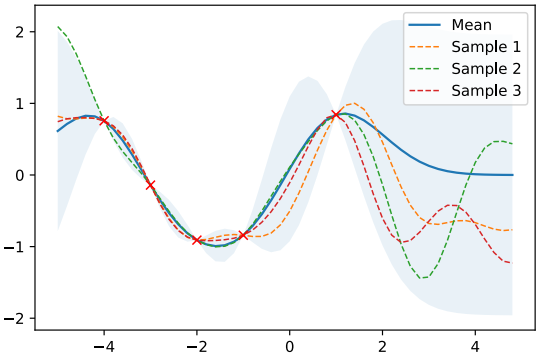

Figure 4: Three samples from a Gaussian posterior, after conditioning on 5 noise-free observations

It is clear from the two figures that conditioning the sample functions upon the data "pulls" the functions through the 5 observations. It is also visible that where the observed data points are concentrated, [-5,1] on the x-axis, so too becomes the 3 functions, with the opposite being true for when there are no data points, [1,5] on the x-axis. This follows as one has greater certainty as to where the functions should lie, if there is observed data within the range of interest. As the observed data diminishes over the range of interest, there is less certainty as to where the samples should lie.

2.2.3 Noisy Data for Predictions

However, it will not always be the case that the data of interest is noise-free. That is, often the data contains unexplained variability. Noisy-data is modelled by considering an underlying function $y = f(x) + \epsilon$, in which $\epsilon \sim N(0, \sigma_y^2)$ and represents the noise. Before consideration can be given to the joint density as before, the covariance of the observed responses needs to be considered as they now contain noise. Murphy[8], describes the covariance as $cov[y|X] = K + \sigma_y^2 I_N \triangleq K_y$.

It then follows that the joint density is:

$$\begin{pmatrix} y \\ f_* \end{pmatrix} \sim N \left[\begin{pmatrix} \mu \\ \mu_* \end{pmatrix} \quad , \quad \begin{pmatrix} K_y & k(X, X_*) \\ k(X_*, X) & k(X_*, X_*) \end{pmatrix} \right].$$

Again by (4), it follows that the posterior distribution is represented as:

$$p(f_*|X_*, X, y) = N(f_*|\mu_*, \Sigma_*), \tag{8}$$

$$\mu_* = \mu(X_*) + k(X_*, X)K_y^{-1}(y - \mu(X)), \tag{9}$$

$$\Sigma_* = k(X_*, X_*) - k(X_*, X)K_y^{-1}k(X, X_*). \tag{10}$$

Lastly, it is important to note that the GP is no longer expected to interpolate data as was the case when dealing with noise-free data. However, the GP is expected to predict values that are sufficiently near to the true observed values. Making use of the same three samples as in Figure 3, the implementation of a GP is illustrated through the use of conditioning upon 7 observations which all possess a component of noise.

In Figure 5, it is clear that once again by conditioning the 3 samples on the data, the functions have been altered. Here, the functions have been "pulled" towards the points. Unlike before where in Figure 4 the functions went exactly through the data points. The cause for this difference between Figure 5 and Figure 4 is due to the presence of noise within the observations conditioned upon in Figure 4. In conclusion, it is important to note that in most real-world scenarios, the results are expected to behave as those shown in Figure 5, as there will always be some degree of unexplained variation within the data

under consideration.

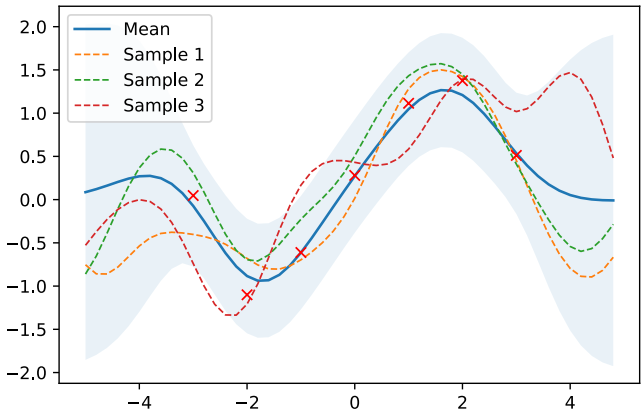

Figure 5: Three samples from a Gaussian posterior, conditioned on 7 noisy observations

2.2.4 Kernel Functions

The only kernel function under consideration within this paper is that of the Gaussian kernel, also known as the squared exponential kernel, as defined by Murphy [8], in the following manner:

$$k(x, x') = \exp\Big(-\frac{1}{2}(x - x')^T \Sigma^{-1}(x - x')\Big).$$

The choice of this relates to the interest of this research in GPs, and their uses. Hence the Gaussian kernel is the natural choice here.

2.3 Gaussian Processes for Classification

Classification is the process whereby data is grouped or classified according to some underlying characteristic(s) of said data [13]. An example of a classification problem would be that of an email spam filter. Through classification, the aim is to sort emails as either spam or not-spam with the use of machine learning. [7]

The aim of statistical classification as defined by Rasmussen and Williams [13], is to assign an input pattern x to a class C, where there exists classes $C_1, ..., C_C$. Any classification problem can be placed in one of two categories. Specifically binary classification, in which data can only be classified into two possible classes, the spam filter example, or multi-class classification in which more than two classes exist into which data can be placed.

[7] Karleigh Moore and Christopher Williams, "Classification Problems", Brilliant, retrieved on 19 March, 2019 from `https://brilliant.org/wiki/classification/`

Unlike in regression, the assumption that the likelihood of interest is Gaussian is no longer appropriate due to the fact the class labels are discrete in classification problems. While previously the assumption was made that our data of interest is continuous. Due to this difference between the nature of data, classification through the use of GPs is computationally more expensive than in regression where both the prior and likelihood were assumed to be Gaussian. This assumption resulted in a posterior which was Gaussian as well, due the the conjugate nature of Gaussian prior [8], and simplified the calculation. Although, the posterior is no longer Gaussian within classification, it may be approximated using a GP, as shown by Rasmussen and Williams [13]. However, due to the focus of this paper on the use of GPs in particular, the context of this paper moving forward will be strictly placed on regression.

3 Markov Chain Monte Carlo Inference

As mentioned before, the issue when attempting to make use of Bayes theorem arises from the evaluation of the denominator of $p(D)$. This complication can be addressed with various techniques. One could bid to obtain the closed-form solution of $p(D)$ through numerical integration, however the closed-form solution of this integral often does not exist, or is computationally intensive to compute. Another approach is the use the of conjugate priors, as the posterior distribution will have the same form as the prior, allowing for the derivation of the closed form solution of the integral. This was the case in the regression context described above. But, as was also shown above in the classification setting, this is not the case and there are many other such instances in which the prior of interest is not a conjugate prior. There are several other techniques which could be used to address this difficulty, one such technique is that of MCMC inference, which is the choice of this paper.

An explanation to Monte Carlo simulation is provided for context: In Monte Carlo simulation, the approach is to generate a set of N samples, $\{x_i : i = 1, ..., N\}$ which are independent and identically distributed. These samples are drawn from a target density, either the prior or posterior of interest. These samples values from $p(x)$, which should be defined on a high-dimensional space X, are then used to approximate the target density (Andrieu et al [1]). This idea can then be built upon to provide the foundation of MCMC inference.

So what is MCMC? Markov chain Monte Carlo inference, as the name suggests is the combination of two statistical branches, namely Monte Carlo simulation and Markov chains. Monte Carlo theory allows the estimation of the parameters of a distribution by considering only random samples of the distribution of interest [16]. An example in which Monte Carlo theory may be applied is when the mean of a gamma distribution is of interest. Rather than solving for the mean mathematically using the distributions' equations, random samples from the specific gamma distribution can be drawn. The mean of these samples can then be calculated. This sample mean then provides an estimate for the true mean.

The above approach is beneficial when to derive the mean of a distribution directly is computationally intensive and a large number of samples from the distribution can be readily drawn [16]. Next, the theory of Markov chains relates to the fact that samples are generated through a sequential process whereby each sample value is used to generate the next sample value, creating a chain [16]. This may appear to imply that the samples become dependent upon one another, which in sampling is problematic. However, in Markov chains the new sample/state is only dependent on the previous sample/state. This is described as the "Markov" property [16].

3.1 Why MCMC and Why it Works

It is also important to note the reasoning as to why MCMC inference is preferred here over Monte Carlo inference. Although, Monte Carlo inference may be applied to many of the same problems as MCMC, MCMC is believed to better handle datasets with high dimensions. As MCMC is believed to be the only approach in which results may be obtained, within a reasonable amount of time [1] when considering high dimensional data. With this performance advantage MCMC has over Monte Carlo inference, and the ever increasing size of datasets available, preference will be given to MCMC within this research.

There are several reasons as to why MCMC works,

1. The Markov chain will possess a unique and stationary distribution under certain criteria [8].
2. Secondly, any MCMC algorithm contains an acceptance - rejection criteria for the samples. In this way not every sample is selected, but only those that ensure the target distribution, and the area under it, are appropriately approximated [8].
3. Another reason as to why MCMC algorithms work, follows from their Markov chain structure. This structure ensures that after a certain amount of iterations the chain will have forgotten the initial values, by the Markov property. This ensures that the Markov chain will converge to a stationary distribution at some point in time [16].
4. Lastly, the posterior distribution of interest does not need to be known, only a distribution that is proportional to it needs to be known, i.e the target density [1].

In conclusion MCMC can be a powerful tool for performing Bayesian inference, however it is not without its' limitations. One such limitation arises due to the fact the posterior distribution does not need to be known, which was previously listed as an advantage. Not requiring the posterior distribution can simply the sampling procedure, however without knowing the posterior distribution it becomes difficult to determine the accuracy of the approximation and judge whether convergence has occurred or not. Despite this, we can proceed without concern for the above limitation as due to the conjugate nature of the the Gaussian prior, the posterior is known to be Gaussian.

3.2 Description and Implementation of MCMC

The fundamental idea of MCMC is to construct a Markov chain on the state space, X, of interest [8]. Hence, the state space is used to construct a stochastic process, as described before, such that the probability of any single event is dependent only upon the previous state. There are several properties which the Markov chain must possess, namely it should be ergodic[8], reversible and its stationary distribution should be the target density of interest.[9] Lastly the Markov chain should be constructed in such a manner that its transitions occur proportionally to the target density.

Making use of the code provided by Wiecki [18], MCMC inference is now illustrated practically. Within this code, the MH (whose detail will be discussed further in Section 3.3) is used. The objective being to determine the posterior distribution of the parameter μ, given the fact that both the prior and likelihood distributions are Gaussian. Consider Figures 6-8. Each figure represents an iteration of the MCMC algorithm. In the first graph (left) the prior distribution is drawn, as too is the initial μ value (blue) which is updated should the proposed μ from the previous iteration not be rejected. The proposed μ value is also overlaid here, whose colour represents whether the value is rejected (red) or not (green). The second and third graphs respectively are the likelihood distribution, given the proposed μ, and the posterior distribution as defined before. The last graph is a trace of how the value of μ develops throughout the iterations.

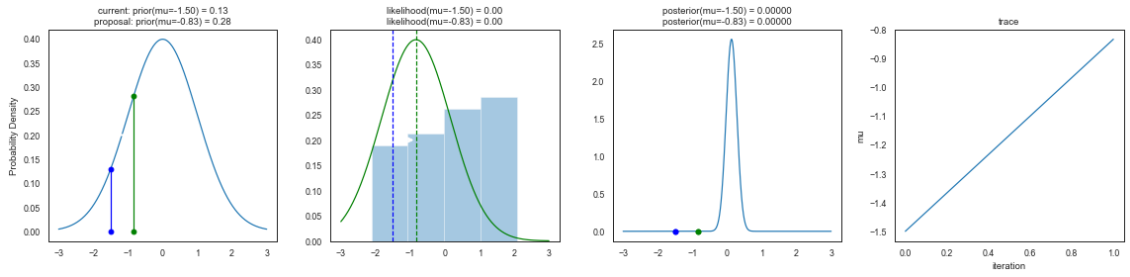

Figure 6: Iteration 1 of a Metropolis - Hastings algorithm, in which the proposed μ is not rejected

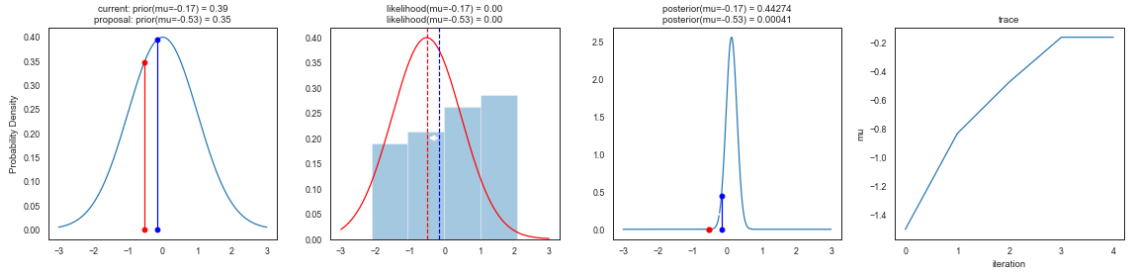

Figure 7: Iteration 4 of a Metropolis - Hastings algorithm, in which the proposed μ is rejected

[8] An ergodic Markov chain is one in which every state of the Markov chain is aperiodic, it is possible to take a single transition from state i to then return to state i, and positive recurrent, the expected transitions between leaving state i and returning to state i is finite.

[9] Thomas Wiecki, "MCMC sampling for dummies", retrieved on 3 March, 2019 from `https://twiecki.io/blog/2015/11/10/mcmc-sampling/`

Figure 6 illustrates an instance of the algorithm in which the proposed μ was not rejected, and thus the value of μ was updated, as shown by the trace. As μ was updated, so too was the likelihood distribution, which in affect updates the posterior due to the reliance of the posterior distribution on the likelihood. In Figure 7 however, the proposed μ value is rejected and hence the value of μ is not updated, again seen in the trace. Due to the rejection of the proposed μ, the likelihood distribution is not updated, as no additional accepted data was observed, and neither was the posterior.

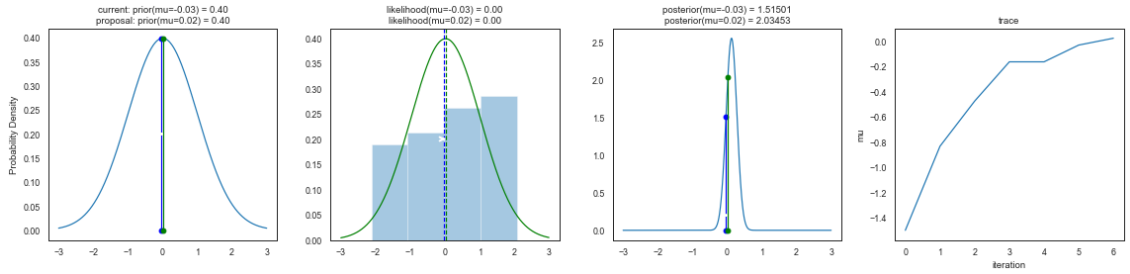

Figure 8: Iteration 6 of a Metropolis - Hastings algorithm, in which the proposed μ is not rejected

While in Figure 8, the proposed μ value is again accepted. By iteration 6, Figure 8, it appears convergence of μ to its' true value has already occurred. However, it will often be the case that for convergence to occur many more iterations will normally be required. Not only will more iterations be required for convergence but more iterations will be required to ensure the independence of the samples. In Section 4 we will show that in order to obtain independent samples a varying amount of iterations based on the implementation of the algorithm will be required.

Thus, if the above illustration is then extended to also determine the posterior distribution of the parameter σ, given that both the prior and likelihood distributions are still Gaussian. Now, we no longer consider only 8 iterations of the algorithm, but rather 15000. With such a large number of iterations, it would no longer be practical to view each iteration individually as above but rather a single trace and histogram of the iterations can be obtained.

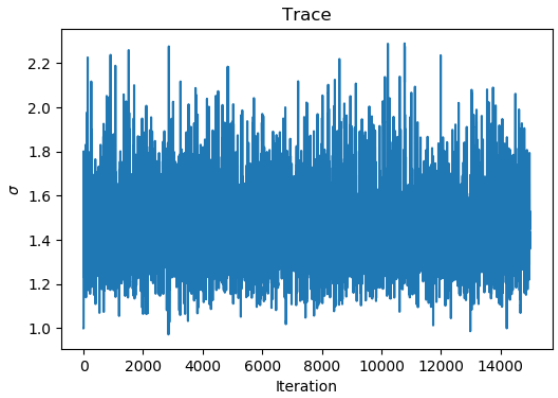

Figure 9: Trace of 15000 iterations of a Metropolis - Hastings algorithm, to approximate σ

Figure 9 illustrates how the value of σ is updated throughout the 15000 iterations performed while Figure 10 is a histogram of the accepted values of σ obtained in the same 15000 iterations. Both of the above figures thus show how the value of σ is approximated through the use of MH. It is this approach that will be made use of later to examine the results provided by any of the MCMC algorithms that will be implemented.

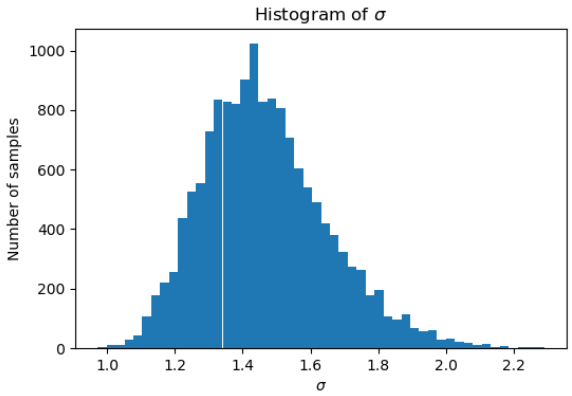

Figure 10: Histogram of 15000 iterations a Metropolis - Hastings algorithm, to approximate σ

Finally, it is important to note that both of the implementations above were coded from first principles. Packages do exist to execute MCMC algorithms directly, which enables easier as well as normally more optimised code. One such package that is readily available for Python is PyMC3, which is used and optimised for Bayesian statistical modelling as well as probabilistic machine learning with particular focus on advanced Markov chain Monte Carlo algorithms [14]. Proceeding into any further application of MCMC algorithms in this paper, PyMC3 will be made use of. The aim of making use of the functions provided by PyMC3, will be to provide easily reproducible and robust code going forward.

3.3 Metropolis - Hastings Algorithm

MH is widely considered the most popular MCMC algorithm [1]. This is largely due to the fact that the algorithm is simple [1] as well as flexible [8]. The algorithm itself has already been illustrated in Figures 6-10, but now the theory as to how the algorithm operates, why it works and any issues that can appear when implementing MH will be addressed.

The fundamental idea of MH is that the algorithm proposes a transition from the current state x to some new state x' at each iteration [8]. The probability of these transitions proposed is characterised by the proposal distribution q and thus the probability of each proposal is given by $q(x'|x)$. Each proposal is either rejected or accepted in accordance with certain criteria [1]. If the transition from state x to state x' is accepted, then the current state becomes x', otherwise the algorithm remains in state x. The acceptance/rejection criteria of the proposals need be such that the time spent within each state by the

Markov chain is proportional to the target density being approximated [1]. In accordance with Figure 6 and Figure 7 before, we saw that should the proposed state x' be more probable than the current state x, the transition is certain. This follows directly due to the fact $\frac{p(x')}{p(x)} > 1$ [8]. In the alternative case in which the proposed state x' is less probable than the current state x, a transition is still possible. These transitions to less probable states ensure that the target density is appropriately approximated [8]. Once the required number of accepted samples has been obtained, a histogram of these samples can be drawn and it is this histogram which approximates the target density [1].

The above described procedure can be summarised into the following pseudocode [8]:

Algorithm 1: Metropolis-Hasting algorithm

Initialise x^0;

for $s = 0, 1, 2, ...$ **do**

  Define $x = x^s$;

  Sample $x' \sim q(x'|x)$;

  Compute acceptance probability

  $$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x')q(x'|x)'}$$

  Compute $r = \min(1, \alpha)$;

  Sample $u \sim U(0, 1)$;

  Set new sample to

  $$x^{s+1} = \begin{cases} x' & \text{if } u < r \\ x^s & \text{if } u \geq r \end{cases}$$

3.3.1 Limitations

Despite the flexibility, and simplicity of the algorithm described above in Algorithm 1 there are several issues which one may face when attempting to implement MH. Any MH algorithm relies heavily on the proposal distribution, as indeed do most MCMC algorithms [16]. This makes the choice of the proposal distribution vital to the success of the algorithm as a whole [1]. One can validate the choice of proposal distribution by ensuring that the density of the proposal distribution results in a non-zero probability of the chain moving to any state within the target density which too has a non-zero probability of occurring [8]. In the implementation in Section 3.2, the choice for the proposal distribution was that of a Gaussian distribution. This is a valid choice for any problem which involves a continuous state space of interest, as any Gaussian random walk proposal has a non-zero probability density on the entire state space [8].

One other issue which greatly affects the speed of any MH algorithm is that of the step-size choice, also known as the proposal width. To illustrate the concept of the proposal width, the implementation within Section 3.2 will be referenced. Within that example, μ was initialised as 0.5. Then, as the proposal

distribution was specified as Gaussian a random variate is drawn from a Gaussian distribution centred around 0.5, the μ. The standard deviation of this distribution is then referred to as the proposal width. The proposal width within Section 3.2 was chosen as 0.5, which worked rather well. However, it is extremely rare that one will be able to select the ideal proposal width without attempting various values and proceeding with a trial and error approach. The effects of poor step size choice can be illustrated in the following figures:

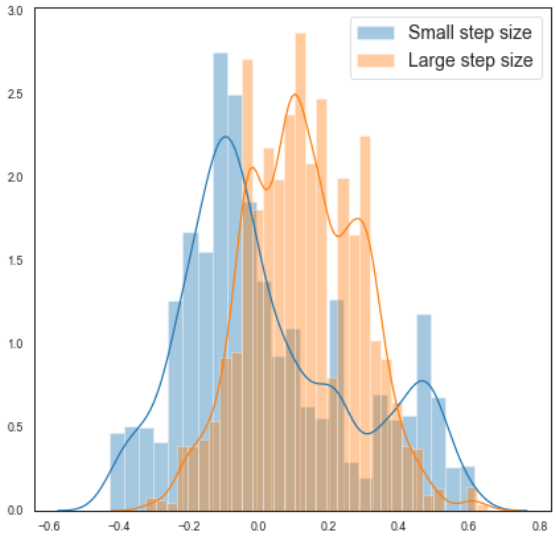

Figure 11: Histogram and density approximating μ

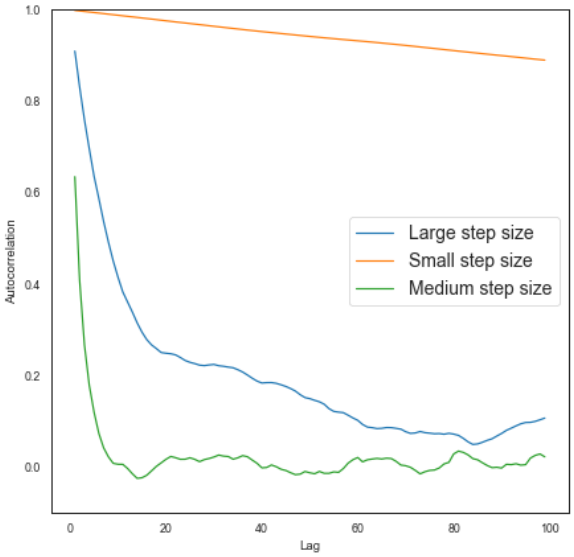

Figure 12: Autocorrelation of the iterations

Hence in Figure 11, the effect of varying step-sizes on the approximation for the parameter of choice is illustrated. In both the cases of the small step-size, 0.01, and the large step-size, 3, convergence to the true distribution of μ has not yet occurred, despite the algorithm undergoing 5000 iterations. If one were to increase the number of iterations, both instances would indeed converge eventually, as the algorithm itself is still a valid MCMC algorithm. Thus, if one were to select a poor choice of step-size this error could be addressed by opting to perform a large number of iterations. However, Figure 12 shows the extent to which the number of iterations would have to be increased by. The reasoning here is that, the samples are required to be independent [1]. In the case of both the small step-size, and the large step-size samples have greater autocorrelation between them than that of the medium step-size, 0.5. This again shows the care that needs to be taken when selecting the proposal width.

3.4 Gibbs Sampling

Gibbs sampling is a special case of MH [8]. That is, when executing a Gibbs sampler each subsequent variable sampled is done so conditionally on all previous variables sampled. Should $D = 3$, there are 3

variables of interest, Gibbs sampling can be represented as [8]:

- $x_1^{s+1} \sim p(x_1|x_2^s, x_3^s)$,
- $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_3^s)$,
- $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1})$,

where x^s represents a given joint sample of all variables, and x^{s+1} is the new sample generated. The above choice of D is done so without the loss of generality, and hence the result can be extended to any D number of variables.

An important consideration when implementing a Gibbs sampler is that of the burn-in period required [8]. This requirement arises due to the fact that the initial state of the Markov chain is arbitrary. Due to the conditional nature of Gibbs samplers this initial state can result in samples which do not approximate the target density of interest appropriately [8], and thus affecting the accuracy of the approximation obtained. In order to address this, the first samples are required to be ignored. This set of iterations for which the samples will be ignored is referred to as the burn-in period [8]. The length of this period required varies, as there is no general rule of thumb that may be applied [1]. However, the use of statistical as well as visual tests may be performed with the aim being to determine if the Markov chain still "recalls" the initial state of the chain [1].

Taking into account the need of a burn-in period, the above procedure can be summarised into the following pseudocode [1]:

Algorithm 2: Gibbs Sampler

Initialise $x_{0,1:n}$;

for $i = 0$ *to* $N - 1$ **do**

Sample $x_1^{(i+1)} \sim p(x_1|x_2^{(i)}, x_3^{(i)}, ..., x_n^{(i)})$.

Sample $x_2^{(i+1)} \sim p(x_2|x_1^{(i+1)}, x_3^{(i)}, ..., x_n^{(i)})$.

$\vdots$

Sample $x_j^{(i+1)} \sim p(x_j|x_1^{(i+1)}, ..., x_{j-1}^{i+1}, x_n^{(i)})$.

$\vdots$

Sample $x_n^{(i+1)} \sim p(x_n|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{n-1}^{(i+1)})$.

Making use of the above algorithm, we will now attempt to implement it with the aid of a tutorial[10]. The goal being to estimate whether or not two coins are biased, and if so to what degree. For each coin $i = 1, 2$, the following information will be available: (z_i, n_i). Where z_i is a count of the number of heads obtained within the n_i tosses undertaken. This problem is a well documented example with an analytical

[10]Duke University, "Metropolis and Gibbs Sampling", retrieved on 23 June, 2019 from `http://people.duke.edu/~ccc14/sta-663-2016/16A_MCMC.html`

solution readily accessible. It is this analytical solution with which comparisons will drawn to determine the accuracy of the approximation obtained. The solution is then the following:

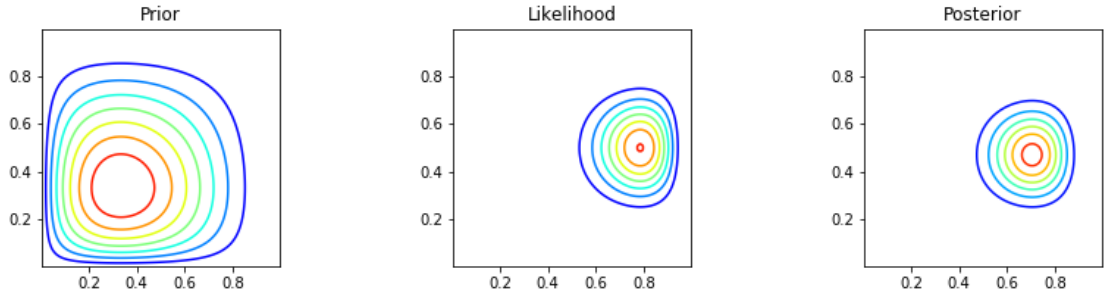

Figure 13: 2-Dimensional representations of the bivariate joint density functions

So in the above figure the analytical joint densities are shown. These densities consist of the following distributions; both the prior and the posterior are beta distributions while the likelihood is a Bernoulli distribution. Then, as each coin is independent of one another, the joint density of the two coins is simply the product of the individual densities [8]. Gibbs sampling is then performed to approximate the densities obtained in Figure 13. The results obtained then appear as follows:

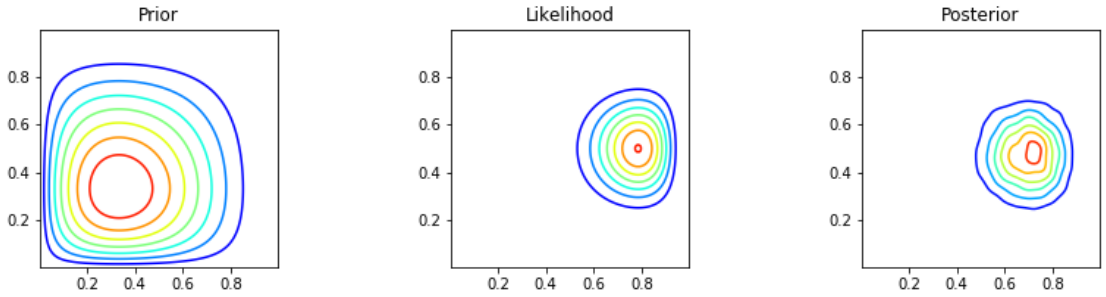

Figure 14: 2-Dimensional approximation of the bivariate joint density functions

Having performed 10000 iterations of the Gibbs sampler and obtained the approximation in Figure 14. Comparisons between the results in Figure 13 and Figure 14 can now be drawn. It is important to note that the priors and the likelihoods are identical in both figures, as neither the prior or the likelihood are approximated here, only the posterior. It is also clear that the approximation obtained is near identical to the analytical solution, with the only difference being in the "roughness" present in the approximation. This "roughness" could also be further addressed by increasing the number of iterations performed. Hence, illustrating the effectiveness with which Gibbs samplers can be executed.

3.4.1 Limitations

Despite the performance and ease with which Gibbs sampling was implemented in the above example, there are several instances in which Gibbs sampling can not only perform poorly, but can be extremely

difficult to implement. As a special case of MH [8], Gibbs suffers many of the same drawbacks as MH [8]. However, there are a few considerations that require attention that are unique to any Gibbs sampler. Firstly, due to the reliance of the algorithm on conditional distributions as shown in Algorithm 2, the effectiveness of the sampler depends greatly upon the ability to draw samples from these conditional distributions [3]. Secondly, due to the sequential nature of Gibbs, only one variable is considered at any given time. This induces strong dependencies between successive samples [3], which as in the case of MH was shown to result in a poor approximation. In order to produce independent samples, one could sample directly from the joint distribution of interest [3]. However, part of the motivation for undertaking MCMC was due to the fact that the joint distribution is often intractable and hence one would be unable to draw samples from this distribution. Having taken into account the above limitations as well as those discussed in Section 3.3.1, the focus of this research now turns to auxiliary variable approaches.

3.5 Slice Sampling

Despite the simplicity of both MH and Gibbs sampling [8], there are several instances in which both algorithms are no longer ideal. Gibbs samplers are restricted as to what model they may be applied to. One such example is that of a logistic regression model [8], where Gibbs is no longer appropriate as, the corresponding visual model has no Markov structure which is of use [8]. However, should Gibbs be applied to an appropriate model, it is often a slow algorithm to obtain approximations [8]. While in the case of MH, the form of the proposal distribution is vitally important [8]. One must be able to select an appropriate proposal distribution, such that the resulting sampling will be efficient [10]. It has also been shown in Section 3.3.1 that the selection of the proposal width within any MH algorithm is crucial to the performance of the algorithm itself as the rate of decorrelation of the samples depends greatly on this parameter. Due to these difficulties, the selection of the step-size is often done so through a trial and error approach [8]. However, this can be extremely cumbersome to perform. Thus, slice sampling was developed by Neal [10] in order to address this issue directly.

The idea of both slice sampling and later elliptical slice sampling is to remove the need for a tuning parameter altogether, the step-size in MH, by introducing an auxiliary variable [1]. The form of the auxiliary variable differs between slice sampling and elliptical slice sampling. Within slice sampling the auxiliary variable is of the form $u \in R$ [3]. The introduction of this variable allows the Markov chain of interest to adapt to the attributes of the target density [10]. Making use of this variable, the target distribution previously defined as $p(x)$ is then extended to $\hat{p}(x, u)$ [8]. The procedure to sample from $p(x)$ then involves sampling from $\hat{p}(x, u)$ and simply ignoring the variable u [10]. Hence, the sampling procedure operates in 2 directions, namely the vertical direction in which samples are drawn from the interval described by the target density and the horizontal direction that is the compromised of the

union of intervals, the slice, within the density at this vertical location [10]. The directions in which the sampling algorithm operates within alternates between the two aforementioned until the required number of samples is obtained [10]. While it is also important to note that there are no samples which are rejected here, unlike in MH. However, 'tuning' of the horizontal region can occur through either a stepping out procedure or a doubling approach such that values returned always lie under the joint density of interest, for the respective vertical position [10]. The procedure can then be illustrated as the following:

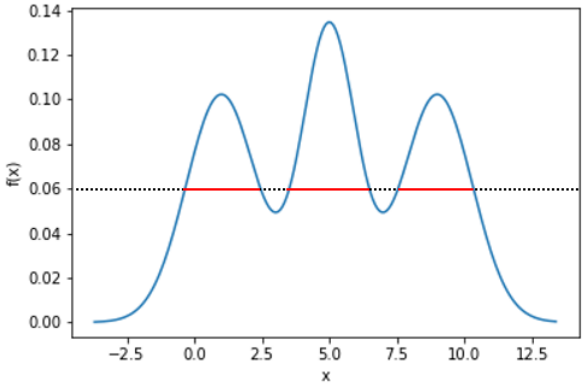

Figure 15: Illustration of slice sampling algorithm on a multimodal distribution

In Figure 15, a multimodal distribution is considered. Consider the vertical position of 0.06, sampled uniformly from under the target density. The line at this position is split into two portions. Namely, the red solid line which compromises of all the points which can be sampled for this vertical position, and the black dotted line which compromises of all the points that cannot be sampled for this particular position.

Considering the above illustration, it is clear to see how the target density is approximated by the sampling algorithm. While it is also particularly clear how the regions of higher density will be visited more frequently by the procedure than those of lower densities. This behaviour ensures valid approximations are returned by the algorithm. As with all preceding algorithms covered within this research, the above approach can be summarised through the use of pseudo-code. However, in order to fully specify the above, further notation is required. Namely, consider the extended target density $\hat{p}(x, u)$, as defined in [1]. It is such that:

$$\hat{p}(x|u) = \begin{cases} 1 & \text{if } 0 \leq u \leq p(x) \\ 0 & \text{otherwise.} \end{cases}$$

This marginal distribution can then be examined further, with the aid of [3]:

$$\int \hat{p}(x|u)du = \int_0^{p(x)} du = p(x)$$

The above integral shows the validity of the ideology of sampling from the extended distribution and

ignoring u in order to sample from $p(x)$. Next, the conditional distributions for this extended density are:

$$\begin{aligned} p(u|x) &= U_{[0,p(x)]}(u) \\ p(x|u) &= U_A(x) \end{aligned}$$

From the above conditionals, it follows that the easy with which A can be specified and identified affects the ease with which slice sampling can be implemented, where $A = \{x; p(x) \geq u\}$ [1]. The algorithm can then be summarised as, [1]:

Algorithm 3: Slice Sampler

for $l = 1$ *to* L **do**

- Sample $u_l^{(i)} \sim U_{[0,f_l(x^{(i-1)})]}(u_l)$

Sample $x^{(i)} \sim U_{A^{(i)}}(x)$ where $A^{(i)} = \{x; f_l(x) \geq u_l^{(i)}, l = 1, ..., L\}$

Slice sampling up until this point has only been considered in the univariate case, however the above description of slice sampling extends to multiple dimensions by sampling each variable successively [10], as in the case of Gibbs, but also introducing a additional auxiliary variable for every dimension considered [8]. Slice sampling however, regularly operates with greater efficiency than simple MH algorithms but is also easier to implement than Gibbs samplers [10]. The reasoning for this follows from the adaptive nature of any slice sampler to select the size of proposals [10], where this decision is based upon the local attributes of the target density. It is this adaptive nature of the algorithm, that makes it attractive as a 'black-box' algorithm that can be applied to a vast array of problems with no tuning required [10].

Having described the algorithm above in Algorithm 3, as well as the relevant notation. Slice sampling is now implemented.

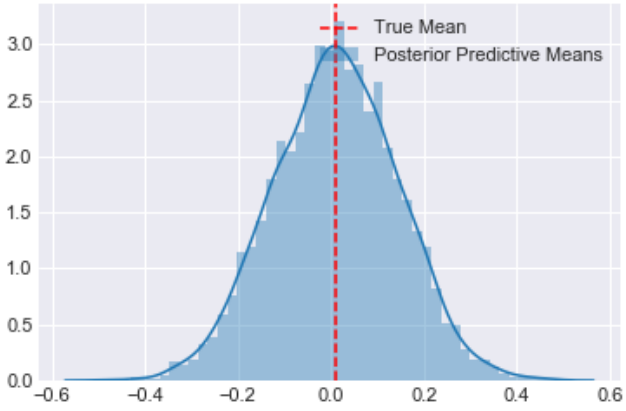

Figure 16: Implementation of slice sampling

Here, the mean of the posterior predictive of a normal distribution is approximated with the use of [14]. It is important to note that the above approximation was obtained with only 5000 iterations, and no tuning of any parameters as is the case in any slice sampler. The approach followed to obtain the result in Figure 16 is an introductory representation of what will be performed later to compare the various techniques discussed within this research.

3.5.1 Elliptical Slice Sampling

The focus of this paper lies on the application of MCMC methods onto GPs in particular. However, when making use of multivariate Gaussian priors or Gaussian processes in particular most 'traditional' probabilistic models induce substantial dependencies between the variables [9]. This implies many of the models previously discussed within this paper may perform inefficiently should the prior be Gaussian. This behaviour forms the motivation for Murray's development of ESS [9], a variant of slice sampling. However the algorithm also possesses several other advantageous properties besides simply it's effectiveness for Gaussian processes. Namely, it possesses no free parameters [9]. Thus, as in the case of slice sampling there is no need to derive and tune parameters directly, but rather the algorithm itself does any tuning required. Secondly, the code to apply the algorithm is both straightforward and comprehensive to a wide range of models [9]. Lastly, despite the above mentioned areas in which elliptical slice sampling is more effective than traditional MCMC methods, it is also considered to be faster than many of these other models mentioned [9]. These aforementioned attributes are why elliptical slice sampling is considered a state-of-the-art MCMC method [6].

When considering elliptical slice sampling, it is important to consider the objective of the algorithm as well as the necessary notation in order to fully specify the procedure. The objective is to sample from a posterior distribution, where *posterior* $\propto$ *prior* $\times$ *likelihood* as before [8]. But the prior of interest is a multivariate Gaussian [9]. This is inline with the objectives maintained throughout this paper when applying any MCMC algorithm, with the only difference being the specification of prior. In order to fulfil this objective, we introduce $\mathbf{f}$, a vector of variables that we hope to sample [9]. Making use of this vector, and the knowledge that the prior is Gaussian, the following distribution can be specified:

$$N(\mathbf{f}\,; 0, \Sigma) \equiv |2\pi\Sigma|^{-1/2} \exp\Big(-\frac{1}{2}\mathbf{f}\,'\Sigma^{-1}\,\mathbf{f}\Big)$$

The distribution above is a Gaussian distribution with covariance Σ and a zero-mean [9]. The consideration of a zero-mean is for simplicity, and due to the fact that any non-zero Gaussian distribution can be simply transformed into a zero-mean Gaussian [9]. The likelihood function is then denoted by $L(\mathbf{f}) = p(data|\mathbf{f})$ [9]. Combining the above likelihood and prior, the posterior distribution can be represented as $p^*(\mathbf{f}) \propto N(\mathbf{f}\,; 0, \Sigma)L(\mathbf{f})$.

In order to sample from the above distribution, Murray began with a Metropolis-Hastings algorithm [9]. Thus, for a given starting point, the vector $\mathbf{f}$ in this case, the proposed state for the chain to move to is $\mathbf{f}' = \sqrt{1-\epsilon^2}\ \mathbf{f}\ + \epsilon\boldsymbol{\nu}$, where $\boldsymbol{\nu} \sim N(0, \Sigma)$ and $\epsilon \in [-1, 1]$ [9]. The probability that the Markov chain will indeed undergo this transition is then given by $p(\text{accept}) = min(1, L(\mathbf{f}\,')/L(\mathbf{f}))$ [9]. However, as this is a MH algorithm there is a step-size parameter present within the algorithm, ϵ in this case. As with slice sampling, ideally the algorithm itself should be able adaptively select and adjust ϵ. Thus, Murray reparameterizes the above such that $\mathbf{f}\,' = \boldsymbol{\nu} sin(\theta) + \mathbf{f}\, cos(\theta)$. This follows as for the any random draw of the auxiliary variable ν, the set of points defining the possible proposals, with varying ϵ, forms half an ellipse [9]. Then, $\mathbf{f}'$ defines an ellipse with $\mathbf{f}$ as a point on the ellipse. The choice of an ellipse as the set of points for the possible proposals is nature even in scenarios with high-dimensions as, within any single iteration of the algorithm the states considered will be within a 2-dimensional plane [9].

The implementation of ESS as described in Section 1.3 as well as above can then be summarised by the following pseudocode [9]:

Algorithm 4: Elliptical Slice Sampler

Input: current state $\mathbf{f}$, a routine that samples from $N(0, \Sigma)$, log-likelihood function $logL$.

Output: a new state $\mathbf{f}'$. When $\mathbf{f}$ is drawn from $p^*(\boldsymbol{f}) \propto N(\boldsymbol{f}; 0, \Sigma)L(\boldsymbol{f})$, the marginal distribution of $\mathbf{f}'$ is also p^*.

1 Choose ellipse: $\nu \sim N(0, \Sigma)$

2 Log-likelihood threshold:

$u \sim Uniform[0, 1]$

$log(y) \leftarrow logL(\boldsymbol{f}) + log(u)$

3 Draw an initial proposal, also defining a bracket:

$\theta \sim Uniform[0, 2\pi]$

$[\theta_{min}, \theta_{max}] \leftarrow [\theta - 2\pi, \theta]$

4 $\mathbf{f}' \leftarrow \mathbf{f}\cos(\theta) + \nu sin(\theta)$

5 **if** $logL(\mathbf{f}') > log(y)$ **then**:

6 Accept: **return** $\mathbf{f}'$

7 **else:**

Shrink the bracket and try a new point:

8 **if** $\theta < 0$ **then**: $\theta_{min} \leftarrow \theta$ **else**: $\theta_{max} \leftarrow \theta$

9 $\theta \sim Uniform[\theta_{min}, \theta_{max}]$

10 **Go to 4.**

In Algorithm 4, it is clear that no explicit rejection of proposals occurs. However, not every proposal is undertaken by the Markov chain. When a proposal is not undertaken, the region from which a proposal may be selected on the ellipse is shrunk, until a suitable state is returned [9]. This procedure continues

until either the required number of samples is obtained, or there only exists a single point on the ellipse remaining which may be selected, characterised by $\mathbf{f}$ [9].

In Figure 17, the process of an ESS algorithm is shown. The initial input is represented by the black cross, while then an ellipse is drawn using $\nu \sim N(0, \Sigma)$ and the definition of $\mathbf{f}'$. This ellipse is characterised by the auxiliary variable ν, shown as the red cross. In order to determine whether a sampled value is accepted or not, the likelihood threshold is required and represented as the green proportions of the ellipses. The blue dot, represents the sample value drawn at each iteration. Figure 17 a) - c) illustrates the process when the sampled value does not lie within the threshold, and is thus the sampled value is not accepted. As the sampled value was not accepted, a new one is drawn, however the available area of ellipse's circumference is shrunk first each time. This is shown through the use of the dotted lines on the circumference of the ellipse. In Figure 17 d) - e) two sampled values which lie within the threshold region are observed and hence returned as accepted sampled values. This process continues until the number of required samples is returned by the algorithm.

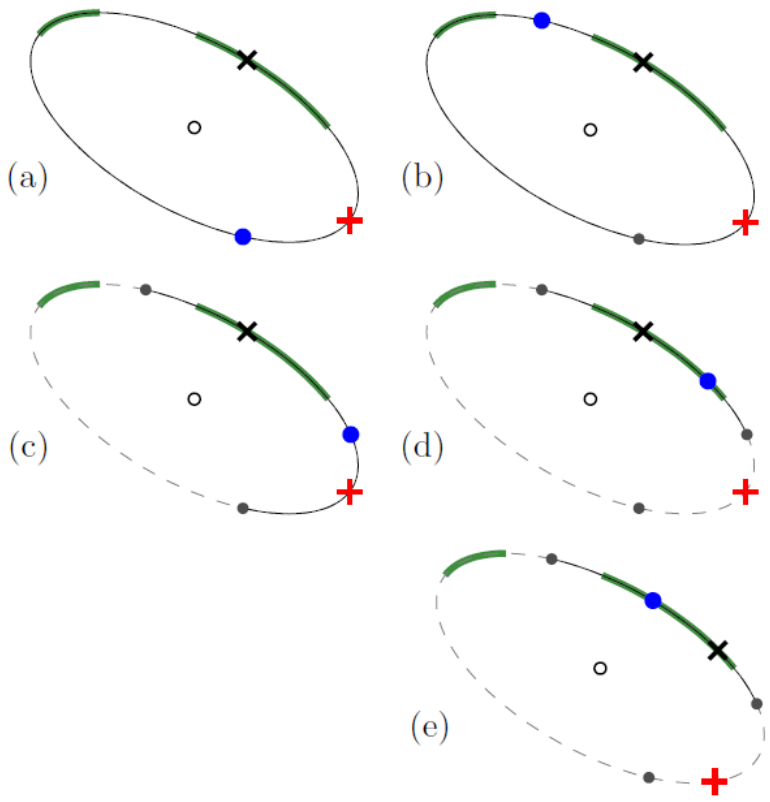

Figure 17: Illustration of an elliptical slice sampling algorithm

Having shown the theoretical framework of elliptical slice sampling, and how it operates it is important to consider it's validity as a MCMC algorithm. In Section 3.2, the properties that any algorithm needs to satisfy in order to be a valid MCMC algorithm were discussed. Those properties include the fact the

Markov chain produced must be irreducible and aperiodic [15]. In the case of elliptical slice sampling, the distribution for the first proposal given an initial angle is $N(\mathbf{f}cos(\theta), \Sigma sin^2(\theta))$ [9]. This then implies that the chain has a non-zero probability of undertaking any transition to a region of non-zero probability under the posterior distribution [9]. As put by Murray, this is then enough to formally prove the chain is both irreducible and aperiodic as required. One can then conclude that the Markov chain possesses a unique stationary distribution, which was mentioned earlier in Section 3.1 as one of the reasons as to why MCMC algorithms work. We can then conclude that applying elliptical slice sampling, with sufficient iterations, will produce samples from the target density or interest.

3.5.2 Limitations

Despite the advantages that both slice and elliptical slice samplers have over MH and Gibbs algorithms, they are not without their own limitations. Slice sampling, particularly in the univariate setting will not work for most if not all complex models [17]. In the multivariate case, the chance of the algorithm functioning optimally is however higher [17]. In the context of elliptical slice sampling, performance suffers greatly when the likelihood is not weak or when the prior no longer aligns with the posterior [6]. The reasoning as to why ESS performance languishes when the likelihood is strong, is due to the fact that once the sampler enters a region with this high likelihood, the probability of the algorithm moving to points outside of this region is low [6]. This results in the algorithm "getting stuck" in this region and thus not approximating the true target density appropriately.

3.6 Hamiltonian Monte Carlo

Up until this point within this research, only Markov chain Monte Carlo methods have been considered to perform inference. We have seen that each of these previously mentioned approaches has limitations but also has models in which they perform best when compared to other MCMC algorithms. However, there exist several other approaches besides MCMC one might consider to perform inference such as expectation propagation [6], variational inference [8] and even hybrid Monte Carlo inference [7]. The attention of this research now turns to hybrid Monte Carlo, also known as Hamiltonian Monte Carlo (HMC) [8]. The reason as to why HMC is considered for this paper, over and above the other techniques mentioned is due to the likeness of HMC and the auxiliary variable MCMC methods discussed within this paper. This likeness will allow direct comparisons to be drawn between the various algorithms.

The theoretical foundation of HMC relies heavily on principles of both physics and calculus [2]. Thus, the theoretical complexity of the algorithm lies far beyond the scope of this research, however we will aim to provide an intuitive explanation of the algorithm. That is, the fundamental idea of the algorithm is to consider any parameter as an atom in space [8]. Auxiliary variables can then be introduced to capture

and represent the 'energy' of these atoms [8]. The reasoning for considering the 'energy' of the parameters is to include information regarding the target density's gradient, with the aim being to improve mixing when considering a high number of dimensions [1]. As with any auxiliary variable introduced so far, the auxiliary variables are updated throughout the procedure in accordance with certain criteria [8]. Despite the presence of auxiliary variables, there are two parameters which the user is still required to specify. Firstly, the user is required to specify the number of 'leapfrog' steps that are taken when updating the auxiliary variable(s) [8], and secondly how large each step taken is. If the number of 'leapfrog' steps is L, at every iteration L deterministic steps are taken, making use of this 'energy' information [1]. Thus, any HMC algorithm is sensitive to these two tuning parameters [8]. There do however, exist methods to automatically select these parameters, but this adds further complexity to an already complex algorithm [7].

The algorithm to perform general HMC is as follows [1]:

Algorithm 5: Hamiltonian Monte Carlo Algorithm

1 Initialise $x^{(0)}$.

2 **for** $i = 0$ *to* $N - 1$ **do**

- Sample $\nu \sim U_{[0,1]}$ and $u^* \sim N(0, I_{n_x})$.
- Let $x_0 = x^{(i)}$ and $u_0 = u^* + p\Delta\frac{(x_0)}{2}$
- For $l = 1, ..., L$, take steps:
  - $x_l = x_{l-1} + pu_{l-1}$
  - $u_l = u_{l-1} + p_l\Delta x_l$

  where $p_l = p$ for $l < L$ and $p_L = \frac{p}{2}$
- If $\nu < A = min\{1, \frac{p(x_L)}{p(x^{(i)})} exp\Big(-\frac{1}{2}\Big(u_L^{'}u_l - u^{*'}u^*\Big)\Big)\}$
  - $\big(x^{(i+1)}, u^{(i+1)}\big) = (x_L, u_l)$

  else $\big(x^{(i+1)}, u^{(i+1)}\big) = (x^{(i)}, u^*)$

It is thus clear from the algorithm above, that the target density is required to be differentiable in its' entirety in order to execute HMC [1]. By ensuring the differentiability of the target distribution, we ensure the gradient at any location can be determined and then incorporated. This inclusion of the gradient, improves mixing of the distributions, as mentioned before, but also allows the suppression of random walks by the algorithm [7]. This greatly aids in improving the efficiency of the approach as, unlike in the case of MH, the region of the target density is no longer explored in a truly random walk, which is considered to be inefficient [7]. It would thus be ideal, if a similar approach could be adapted to the MCMC algorithms mentioned beforehand in order to improve their respective efficiencies. Such, an approach exists in No-U-Turn sampling, a MCMC algorithm that incorporates the favourable properties of HMC with those of slice sampling [7].

3.6.1 No-U-Turn Sampling

No-U-Turn sampling, NUTS, is an MCMC algorithm, which is used when the objective is to sample from some target density in order to perform inference, as before [8]. NUTS differs from the algorithms previously introduced before within this paper in several ways. Firstly it has the ability to 'silence' transitions within the random walk the Markov chain undertakes [7]. This improves the efficiency of the algorithm, while also addressing the limitation of elliptical slice sampling to become 'stuck' within regions of high likelihood [7]. The silencing, or rather the suppression of random walks is performed through the inclusion of a HMC characteristic to account for the 'energy' of the parameters [7]. As before, the 'energy' of the parameters is captured through the gradient of the target density, however NUTS does not require the specification of L, the number of 'leapfrog' steps taken at each iteration [7], unlike the general HMC described above. Due to the sensitivity of the algorithm to L, as well as the issues that were shown in the case of MH when a tuning parameter is poorly selected in Section 3.3.1, this is ideal. The reason as to why L is no longer required to be specified, is due to the addition of a criteria within the algorithm that determines whether sampling further steps within each iteration will increase the distance between the initial value, and the proposed value [7]. It follows intuitively then that L can be dynamically adjusted throughout the iterations so that at each iteration the maximum possible change between the initial value and the proposed occurs. The criteria that ensures this, is a dot product of r, the current 'energy', and $\theta^* - \theta$, the difference between the proposed state θ^* and the initial state θ [7]. The algorithm can then iterate through the 'leapfrog' steps until this criteria becomes negative, and this is where the algorithm draws its' name from as iterations are hence undertaken until the a state is proposed in the opposite direction [7].

We could now provide pseudo-code for NUTS, however the algorithm is extremely complex [2] and thus the mathematical explanation of the algorithm as well as the pseudo-code will not be considered further within this research. Readers can however consult [7] in which both a naive as well as more efficient formulation of NUTS are provided. Despite not delving further into the theoretical framework of NUTS, we can still make use of PYMC3's built in NUTS algorithm [14]. The aim within the implementation of NUTS to follow is to make use of Gaussian processes in order to perform regression. In order to perform the above experiment, 100 bivariate Gaussian data points are generated randomly, while a zero-mean for the Gaussian process is chosen. In order to put the above experiment inline with data from a true experimental procedure, Student T noise will be added to the data. We then make use of the theory in Section 2, and more specifically Section 2.2.3 to condition the functions appropriately. Lastly, the range of possible x values considered when generating the data is $[0, 10]$. We will aim to make predictions regarding x in the region $(10, 15]$, through the use of the conditional distributions derived in Section 2.2.3.

Hence, we obtain the following results:

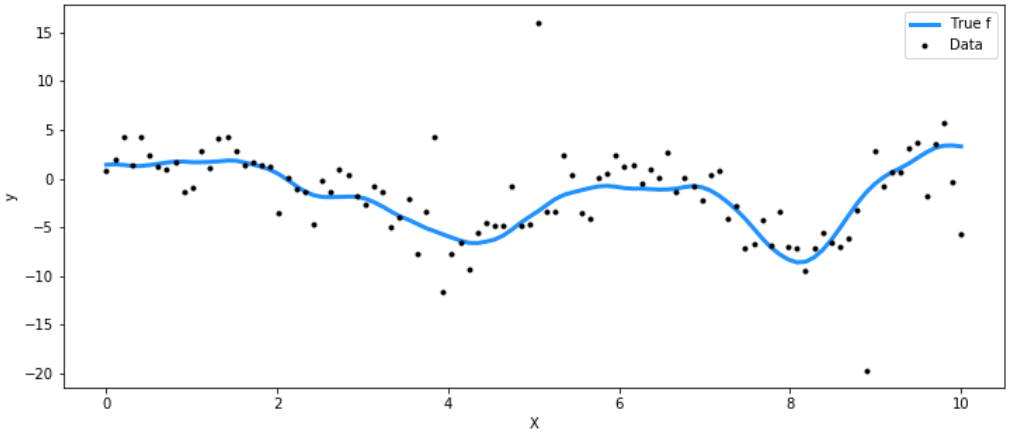

Figure 18: Approximation of the posterior through the use of NUTS

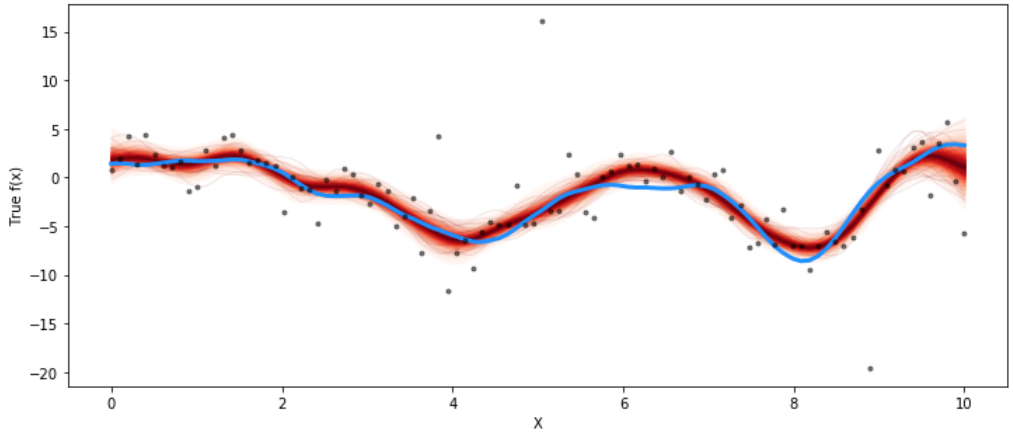

Figure 19: Approximation of the posterior through the use of NUTS

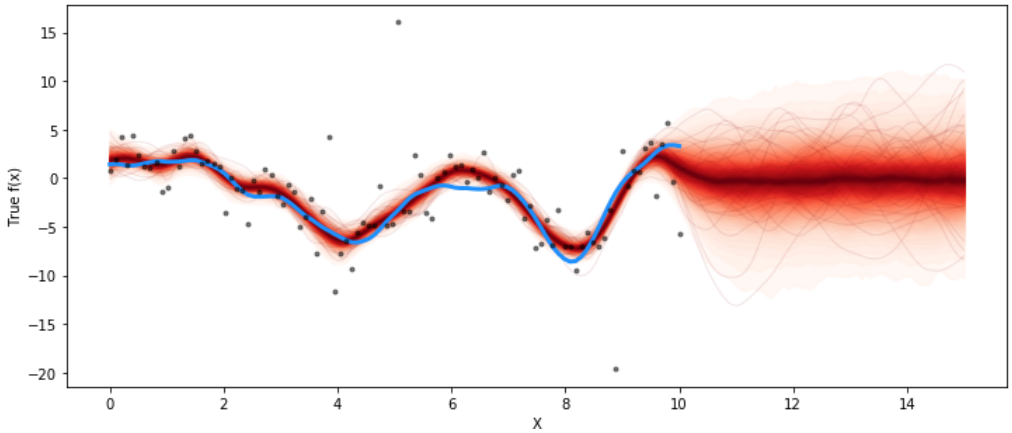

Figure 20: Approximation of the posterior predictive through the use of NUTS

Figure 18 contains the 100 generated data points, as well as the true unknown f function underlying the data. In Figure 19, 1000 GP priors are placed upon the unknown f. As was shown before in Section

2.2.3, the priors are heavily concentrated when the data points themselves are concentrated, while there is greater spread between the priors when there either fewer data points, $x \in [9, 10]$, or the data points themselves contain greater variability. It is clear that the GP priors approximate the true underlying f rather well, except around $x = 6$ where the GP priors model the data very well, but not f. Lastly, in Figure 20, the range of the GPs is extended making use conditional GP formulations to make predictions, where there are no data points. There is a strong concentration of the GPs in the range $(10, 15]$ roughly around zero. This agrees with the formulation of the original problem as a zero mean GP was specified. Thus, one can conclude that NUTS has performed rather satisfactory here based on visual inspection of the above figures. Finally, it is important to note that a similar approach could be followed to perform Gaussian process classification.

4 Application

Having discussed the theory of several MCMC algorithms available, which one could make use of. A choice as to which method will be made use of to solve a particular problem is required. In practice not all of the above methods can be implemented on any single problem. This is due to time constraints, if one were to attempt to implement them one after another or simultaneously, there is a great computational cost involved in doing so. It is also considered poor statistical practice to select the algorithm based on the ease of implementation as, although the chosen algorithm is easy to implement, it may not be suited to the characteristics of the problem at hand and hence produce incorrect results. Thus, the ability to select between these algorithms is crucial, however this is far from a straightforward task. There are certain criteria one can make use of to aid in the decision making process however, namely the availability of data and it's dimensions, the correlation between variables, the knowledge of the user and the purpose of the analysis. These aspects can then be broken down, and weighted according to importance. If one is considering a data-set with high-dimensionality, methods such as NUTS and elliptical slice sampling which are better designed to handle such data should be prioritised [7, 9]. Secondly, if the user has little to no knowledge of MCMC and statistical techniques, methods which require the user to specify tuning parameters should be avoided, such as MH and HMC [8, 2]. The purpose of the inference should be greatly considered and whether efficiency, computation time, or accuracy of the estimates should be prioritised. Such a decision is required as there often exists a trade-off between the efficiency of the algorithm and the accuracy of the results obtained. Thus, the requirements should be given immense consideration when deciding upon the number of iterations, chains[11] and the choice of algorithm to implement.

In order to provide further clarity on which algorithm is to be preferred under a range of conditions, we aim to build on various experiments already performed. In [9], Murray provides a comparison of

[11] The number of MCMC runs undertaken.

elliptical slice sampling, line slice sampling, control point MH and Neal's MH by considering data-sets of size 200, with the dimension varying from 1 through to 10. The measures of interest used to compare these 4 methods were computation time, the number of likelihood evaluations and the number of effective samples [9]. One issue with this comparison however, is that these metrics only consider the efficiency of the algorithms and not the accuracy of any of the results obtained. Secondly, a comparison of how the correlation between variables affects algorithm performance will be considered. This comparison is inspired by Nishihara [11], in which elliptical slice sampling's performance is evaluated for varying levels of covariance. Next, a comparison of slice sampling is required, and found within [10]. Here, Neal compares various different MH specifications to slice sampling making use of a 10 dimensional Gaussian distribution with zero-mean as the sampled distribution. Lastly, [7] provides a comprehensive example of the performance of NUTS against Metropolis - Hastings and Gibbs sampling, but it is important to note that within this example, all 3 approaches undergo $1,000,000$ iterations. This will not always be possible due to the computation challenges that arise with such a large number of iterations.

Taking into account the above mentioned experiments, there are several questions this research aims to address, specifically:

1. *How does the number of dimensions affect the performance, with respect to accuracy and efficiency, of Metropolis - Hastings algorithms, slice samplers, elliptical slice sampling and no-u-turn samplers?*

2. *How does the correlation between variables affect the performance, with respect to accuracy and efficiency, of Metropolis - Hastings algorithms, slice samplers, elliptical slice sampling and no-u-turn samplers?*

3. *Based on the above two questions, are there clear circumstances in which a specific algorithm performs significantly worse/better than all other algorithms investigated within this paper?*

In the above questions Gibbs sampling has been excluded, the reason being that Gibbs sampling is only specified within PyMC3 [14], with respect to variables which are either categorical or discrete. For the purposes of this paper, an assumption was made earlier to only consider data which is continuous. Secondly, as the focus of this research lies on drawing conclusions regarding the performance of various MCMC algorithms, PyMC3 [14] was selected to ensure that fairness is maintained when comparing algorithms. That is, if one were to code a Gibbs sampler for continuous data, comparing the performance of a sampler coded from first principles against those which are optimised and available within PyMC3, would not provide meaningful results. This is due to the fact that any comparative measure would be measuring the coding ability of the individual and would hence be on an 'apples to oranges' basis. Before continuing further it is important to reiterate that all application will thus make use of PyMC3, version 3.7 and will be implemented in Google Colab[12].

[12]Google Colab, accessible at `https://colab.research.google.com/`

4.1 Dataset and Implementation

The datasets on which the various MCMC algorithms will be applied within this research will be randomly generated within Python with several varying properties. Each dataset will consist of 200 observations, with the dimensions ranging from 2 through to 10, as done in [9]. The distribution of the data itself will be multivariate normal, with a randomly generated $\boldsymbol{\mu}$, and alternating covariance structures. These covariance structures will consist of three structures, particularly two unstructured covariances, one with low covariances between variables, $(0, 10]$, one with higher covariances between variables, (10.40], and lastly a variance component covariance structure. The structure of the datasets has been chosen in this way, so that the relationship between the number of dimensions under consideration, the correlation between variables and the performance of the algorithms can be observed.

The choice of the above data structure is also done so as to ensure it matches the work in both [8] and [10], hence providing us with a means to compare the results obtained within this paper, with similar investigations. This point of reference will provide assurances that the algorithms are operating as intended and that no errors are made in specifying the models within PyMC3's framework. Finally, in order to implement elliptical slice sampling in PyMC3, the Cholesky decomposition of the covariance matrix is required to be specified [14]. The use of the Cholesky decomposition is appropriate as it allows the computationally intensive task of inverting large covariance matrices to be avoided, which in turn improves the numerical stability of the computation [8]. The application of the decomposition can be further justified as instinctive as, it frequently appears in both GP regression and GP classification procedures [8].

4.2 Results

The results of the above described experiment will now be provided. However, due to the nature of the experiment being that of a simulation study, the direct results of the simulation such as trace plots are often very difficult to interpret and compare between the various simulations. Considering, Figure 21 in which only a 2 component $\boldsymbol{\mu}$ is considered, it is clear that in large dimensions these are uninformative to the reader. This issue is further exacerbated by the sampling of multiple chains. Another visual technique which could be employed to evaluate MCMC algorithms, is that of credible regions such as the one shown in Figure 22. Once again though, making use of this visual measure to investigate the performance of any MCMC algorithm is extremely difficult to scale to dimensions larger than 2. Thus, going forward for this paper, numerical measures will be preferred due to their scalability to high dimensions. When evaluating the performance of any MCMC algorithm, it is important to note that any single measure can fail to detect the convergence or lack thereof that it is designed to test for [5].

Due to such circumstances, it is considered good statistical practice to consider a combination of

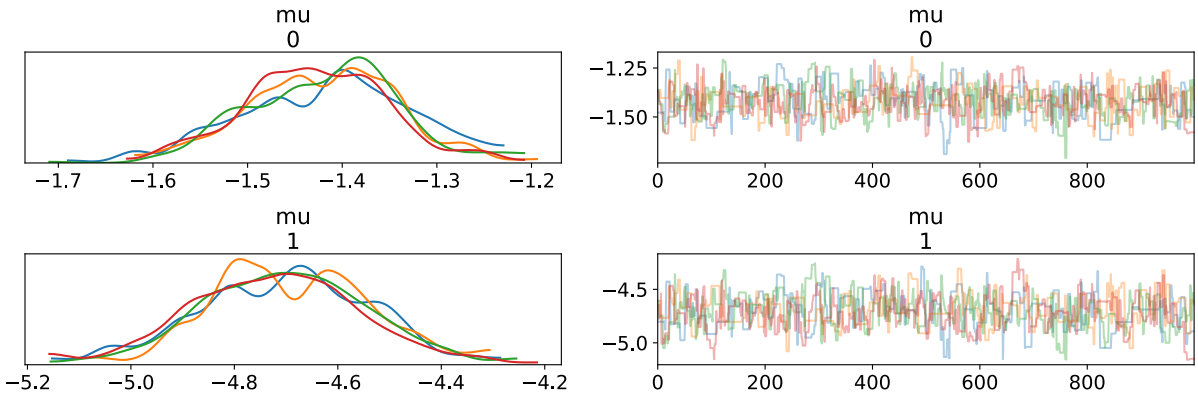

Figure 21: Trace plots of 4 chains of a 2-dimensional $\boldsymbol{\mu}$

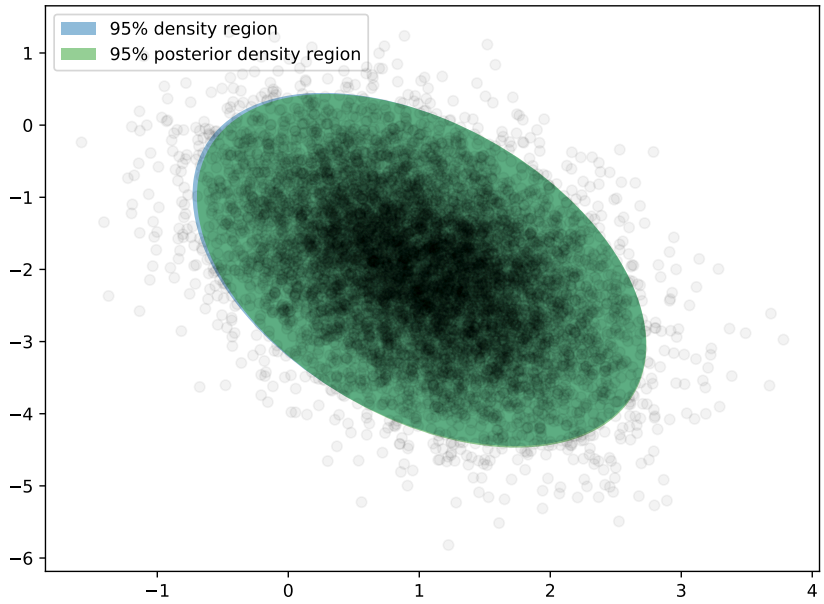

Figure 22: 95% Credible region of a posterior density approximation

measures to evaluate any MCMC algorithm [5]. The chosen combination for this paper is the computation time, the effective sample size, the Gelman - Rubin test statistic and the standard errors of the estimated parameter values obtained. This combination allows the speed with which any estimate is obtained, the accuracy of the estimates obtained, and the reliability of the estimates obtained to all be evaluated and compared.

The computation time, despite not being a statistical measure, is important to consider because often in practice there are strict time constraints in which results are required to be obtained within. Secondly, by providing the computation time and measures of accuracy, any reader can decide based on their needs where the balance lies between obtaining timely results and requiring more accurate results. Thus, the

respective computation times were obtained as follows:

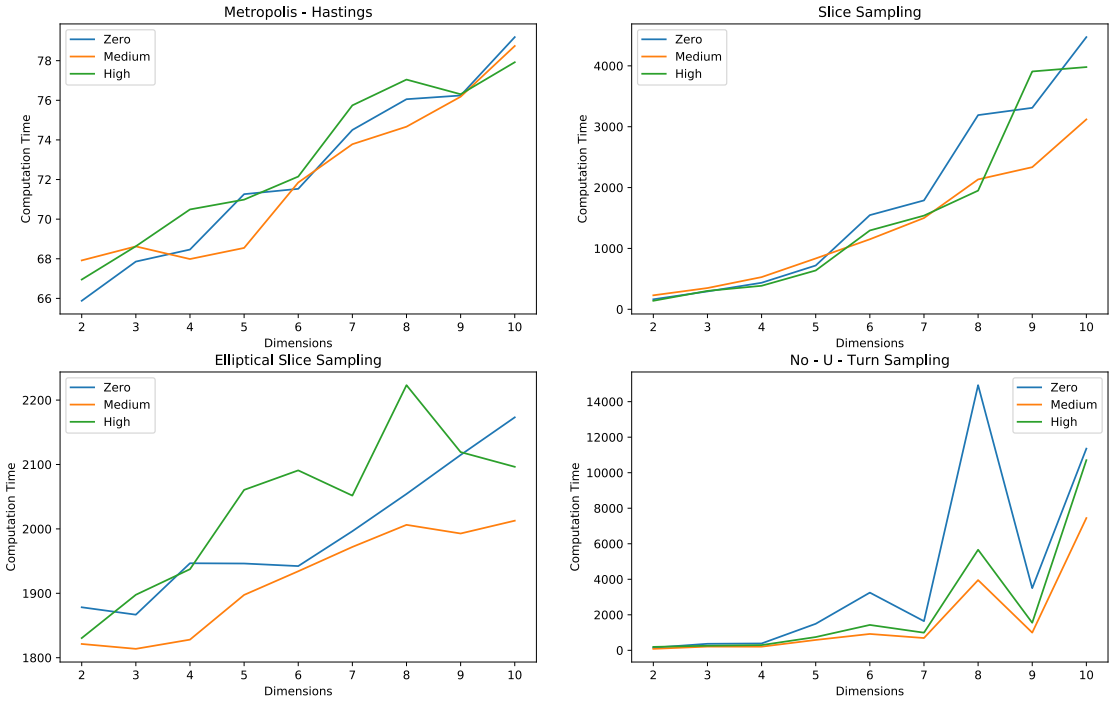

Figure 23: Computation time per algorithm with varying covariance structures

From the above figure, it is clear that as the dimensions under consideration increase, so too does the run-time required of every algorithm. However, it is important to note that the y - axis scale differs greatly between the various algorithms considered. This is common throughout the results provided within this section, due to the manner in which the results are presented on a per algorithm basis. From the results, it can also be seen that the varying covariance structures and subsequent correlations between the variables had a minimal impact on the run-time.

The second measure considered is that of the effective sample size. This is a measure of the number of samples obtained within the run of the algorithm which have sufficiently low autocorrelation between one another, such that they can be considered independent. Hence, when the samples under consideration are independent, the number of effective samples equals the sample size itself. Therefore, the effective sample size provides a measure of reliability. Due to the nature of MCMC algorithms previously discussed throughout this paper, the independence of samples is a desired property and the degree to which the MCMC algorithm satisfies this can be evaluated. As the experiment undertaken here aims to approximate the value of $\boldsymbol{\mu}$, the effective sample size per a component of this $\boldsymbol{\mu}$ are obtained. This is difficult to compare between the various dimensions under consideration, thus the effective samples will be averaged across

the components to obtain a comparable measure across dimensions. Hence, the effective samples obtained are as follows:

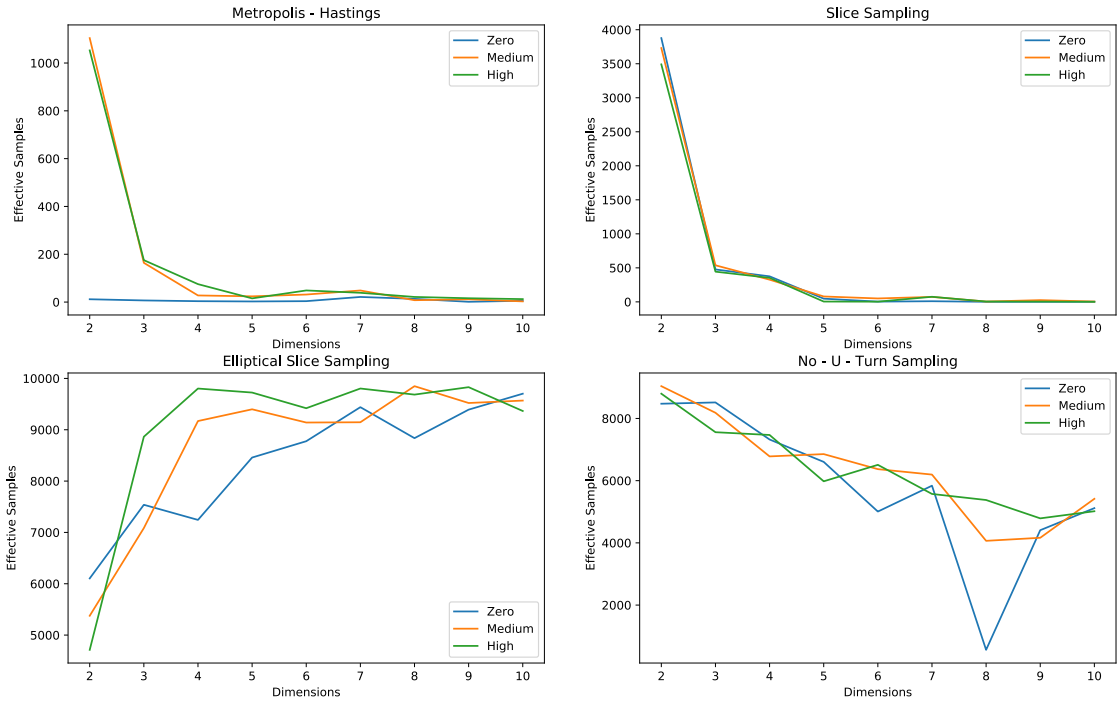

Figure 24: Effective sample size per algorithm with varying covariance structures

Figure 24 shows the trend of the effective samples as the dimensions increase. For all of the algorithms, the effective sample size decreases as the dimensions increase, except for elliptical slice sampling. For MH and slice sampling, the decrease could be classified as exponential, while for NUTS the decrease can be considered linear. Again, as the y - axis scale differs between the algorithms shown, it is important to note that the performance of Metropolis - Hastings and slice sampling are both well below that of elliptical slice sampling and no - u - turn sampling. In lower dimensions NUTS performs the best with respect to the effective sample size, however elliptical slice sampling outperforms NUTS from dimension 4 and onward. Finally, the varying covariances appear to not have a significant impact on the effective sample size as was the case with computation time.

The third measure examined is the Gelman - Rubin test statistic. The test statistic allows for the convergence of the algorithm to be measured, where convergence is considered to have occurred, the test statistic is 1 [5]. Any divergence or lack of convergence by the algorithm is thus represented by a divergence of the test statistic from 1. The Gelman - Rubin test statistics obtained are the following:

To evaluate these Gelman - Rubin test statistics, a decision boundary is required to decide when

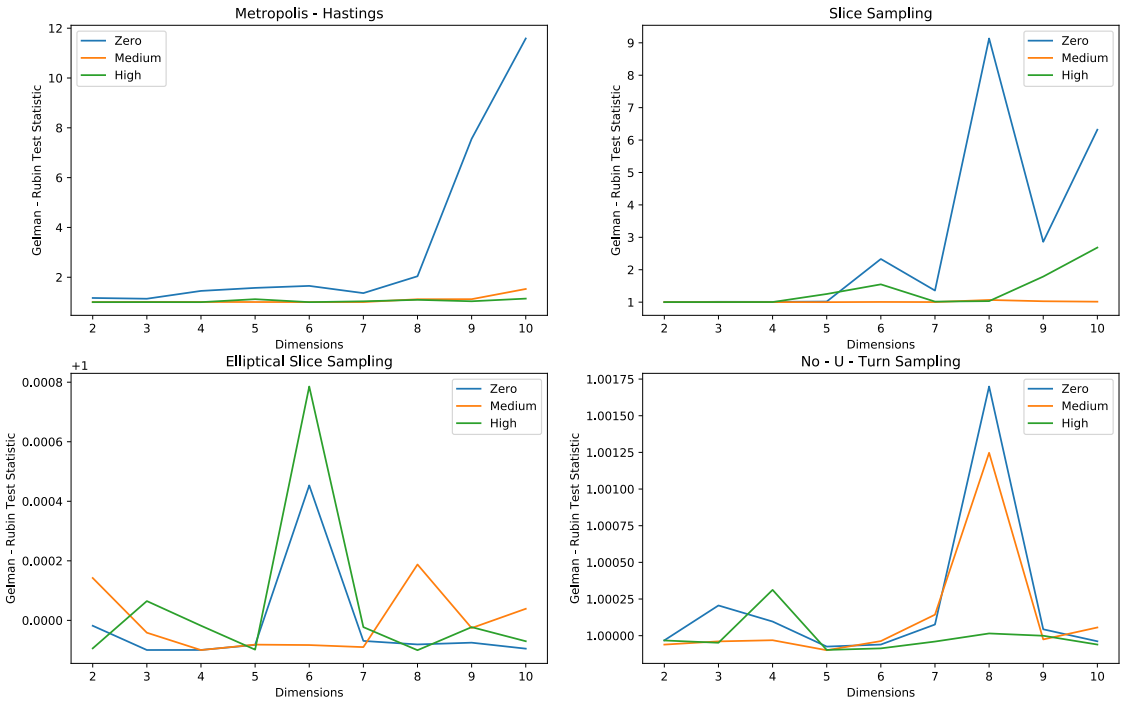

Figure 25: Gelman - Rubin test statistic per algorithm with varying covariance structures

convergence is deemed to have not occurred. The choice for this is subjective, thus for the purposes of this paper, a Gelman - Rubin value of above 1.4 will be considered to indicate divergence, as this is the choice of PyMC3 [14]. While, a Gelman - Rubin value of 1 does not indicate the algorithm has converged to the true underlying distribution, but rather the algorithm has converged to a stationary distribution. It can then be see that for all of the runs of ESS and NUTS the algorithm was considered to converge. Whereas, in the case of MH and slice sampling, in high dimensions, roughly dimension 7 and onward, the algorithms did not achieve convergence. In fact the algorithms experienced significant divergence. This is illustrated by the significantly large test statistic values observed. In all of the algorithms excluding ESS, when the variables where independent, the algorithms obtained the highest Gelman - Rubin test statistic. This is a rather interesting result as often in statistics, independence is highly desirable characteristic of the observations. That is, often independence is assumed, despite the observations not being independent. From the results, it can be concluded that no such assumption is required in the context of MCMC inference.

The last measure considered is that of the standard error of the estimates obtained, which provides a measure of accuracy. The choice of standard error over something such as mean square error is due to the behaviour of the mean squared error as dimensions increase. That is, by the definition of the mean

squared error, as the dimensions increase so too will the mean squared error. The standard errors can then be summarised as follows:

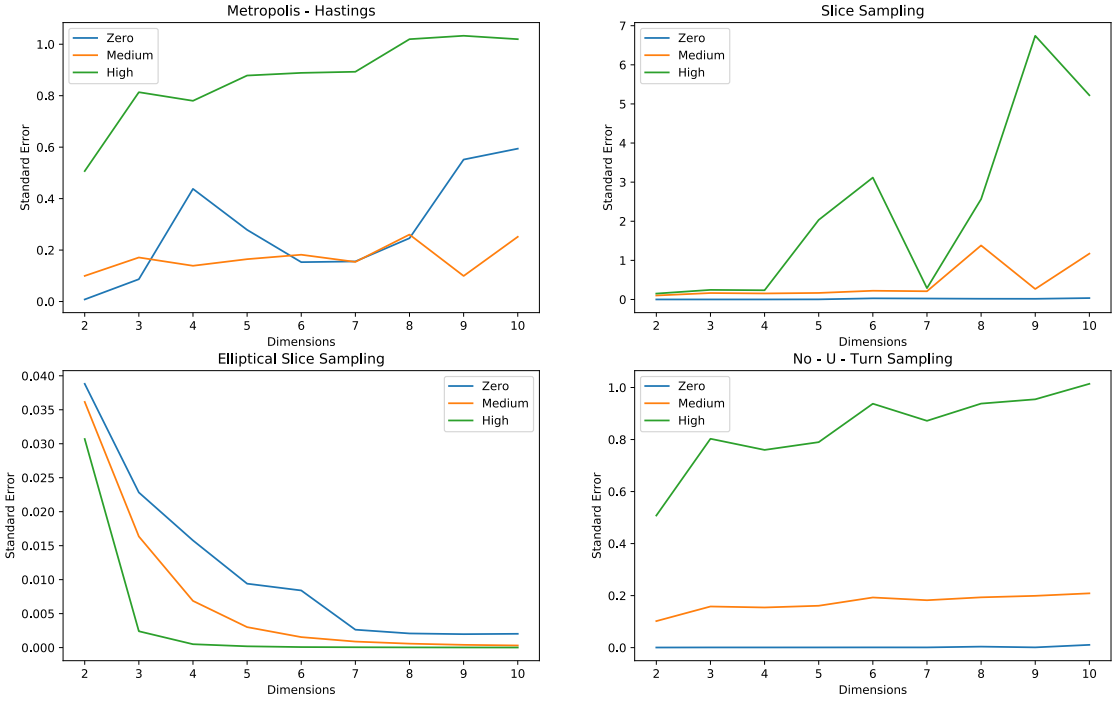

Figure 26: Standard error per algorithm with varying covariance structures

The standard errors above are arguably the most important measure considered within this study as they provide an indication of the accuracy of the estimates obtained by the various algorithms considered. Within the standard error values, two general trends can be observed in all of the algorithms considered, barring ESS. The first trend is that of an increasing standard error as the dimensions under consideration increase. The second trend observable is that when the variables observed contained higher correlations with one another, the standard error calculated was also higher.

4.3 Discussion and Future Works

From the results obtained in Section 4.2, there are several conclusions which can now drawn. Firstly, that is if there are strict time constraints with which results are required to obtained within, Metropolis - Hastings is the algorithm to utilise. In this investigation, MH provided results far faster than any of the other algorithms considered, this was expected as a result of the simplistic nature of MH previously discussed. There is however, a consequence of the speed and simplicity of MH. MH performed the worst of the 4 algorithms considered with respect to the other 3 measures. In this particular investigation,

the standard errors for MH are still not particularly large, so the estimates obtained are still relatively accurate. However, if the low effective samples as well as the lack of convergence obtained by MH is considered, MH should only be preferred in two circumstances. Namely, when under strict time constraints as discussed before, or to obtain a rough idea of the possible values of the parameters of interest. This rough idea should then be used to either improve the MH algorithm, or appropriately select the algorithm which will be utilised.

Secondly, the experiments performed within this paper show that slice sampling performed rather poorly across the board. One possible reason for this relates to the formulation of slice sampling, as shown in Figure 15. Within the experiment, the Gaussian distribution, a uni-modal distribution, is only considered. Hence, any slice of the distribution is simply the vertical subsection of the distribution, while slice sampling is known to excel particularly when the distribution under consideration is multi-modal [10]. Should any reader of this paper require the use of MCMC inference when considering a multi-modal distribution, the performance of slice sampling can be expected to greatly improve.

Elliptical slice sampling and it's performance is now considered. ESS, obtained the highest effective sample size, showed arguably the best convergence, as well as the most accurate estimates of any of the algorithms considered. This was all done in reasonable time, even in high dimensions. The gap in performance between ESS, and the other algorithms considered, is due to the formulation of ESS to handle Gaussian priors specifically, whilst the other algorithms considered are known to incur difficulties [9]. Although, ESS and it's performance is far better than any of the algorithms with which it is compared to within this paper, there are difficulties which arise to make use of the algorithm itself. That is, in order to implement ESS with the use of PyMC3, the specification of either the prior covariance matrix, or as is the case in our implementation, the prior precision matrix is required [14]. This additional specification implies that as a 'black-box' approach for individuals without a statistical background, ESS can be difficult to implement. However, should this difficulty be overcome, ESS provides the best statistical results. One last caution to recognise that is unique to ESS and implementing it through the use of PyMC3 is that ESS is flagged as experimental by the developers of PyMC3. This implies, that in it's current state ESS in PyMC3 may behave unexpectedly under certain conditions.

NUTS performed rather well with respect to all measures, except computation time, as the algorithm was by far the slowest of those considered. The additional computation time appeared to provide no additional benefits with respect to accuracy or reliability. One such reason as to why the computation time was significantly higher here than for the other algorithms considered, is due to the fact that NUTS evaluates the gradient of the density at every iteration. However, again due to the consideration of the Gaussian distribution, whose gradient does not differ significantly, these additional evaluations do not provide any significant improvements to the algorithm. If the reader requires an approximation of a

distribution whose gradient has significant changes with little change in the x value, NUTS should be able to perform better across the board.

In conclusion, it is important to note none of the algorithms discussed within this paper were implemented to solve either regression or classification type problems. Hence, an avenue for future work to be done is to compare their ability to learn and predict unseen results within these respective contexts. Another shortfall of the research performed here is the assumption explicitly of Gaussian distributions for both the prior and likelihood. This could be extended to several other distributions, including mixtures of distributions, but also discrete distributions. Lastly, another future consideration could be the comparison of ESS, or all of the other algorithms to various other approaches such as Monte Carlo simulation, expectation propagation or any other alternative to MCMC.

References

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(5), 2003.

[2] Michael Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[5] Mary Kathryn Cowles and Bradley P. Carlin. Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[6] Francois Fagan, Jalaj Bhandari, and John Cunningham. Elliptical Slice Sampling with Expectation Propagation. *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 2016.

[7] Matthew D. Hoffman and Andrew Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[8] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[9] Iain Murray, Ryan Prescott Adams, and David J. C. MacKay. Elliptical slice sampling. *AISTATS 13*, 9:541–548, 2010.

[10] Radford M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003.

[11] Robert Nishihara, Iain Murray, and Ryan P. Adams. Parallel MCMC with Generalized Elliptical Slice Sampling. *The Journal of Machine Learning Research*, 15(1):2087–2112, 2014.

[12] Valerie Poynor and Athanasios Kottas. Nonparametric Bayesian Inference for Mean Residual Life Functions in Survival Analysis. *Biostatistics*, 20(2):240–255, 2018.

[13] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[14] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016.

[15] Luke Tierney. Markov Chains for Exploring Posterior Distributions. *Annals of Statistics*, 22(4):1701–1728, 12 1994.

[16] Don Van Ravenzwaaij, Pete Cassey, and Scott D. Brown. A simple introduction to Markov Chain Monte-Carlo sampling. *Psychonomic Bulletin & Review*, 25(1):143–154, 2018.

[17] Stephen G. Walker. Slice Sampling: Discussion. *The Annals of Statistics*, 31(3):755–758, 2003.

[18] Thomas Wiecki. MCMC sampling for dummies. `https://twiecki.io/blog/2015/11/10/mcmc-sampling/`. Accessed: 2019-03-03.

Appendix

The programming language of choice for this paper is the programming language Python, version 3.7.1, developed by the Python Software Foundation and available at:
`https://www.python.org/downloads/release/python-371/`
While the versions of the packages utilised within this research are:

- numpy - version 1.16.4
- pandas - version 0.24.2
- matplotlib - version 3.1.0
- pymc3 - version 3.7
- seaborn - version 0.9.0
- scipy - version 1.2.1
- theano - version 1.0.4

All code used throughout this research paper is available online at: `https://github.com/RicSalgado/Research-Report` and was executed on a personal computer, barring the application section, containing the following specifications:

- Operating System: Windows 10 Home Single Language 64-bit
- System Model: HP Pavilion x360 Convertible 14-ba1xx
- Processor: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (8 CPUs), 1.8GHz
- Memory: 8192MB RAM

The code made use of to perform the experiments shown within the Application Section was executed with the use of Google Colab, accessible at `https://colab.research.google.com/`.