An Introduction to Adaptive Markov Chain Monte Carlo

Candidate Number: HNKR2, Word Count: 13221

Abstract This report gives an introduction to the theory, implementation, and testing of adaptive Markov chain Monte Carlo (MCMC). We first give a motivation and theoretical framework for non-adaptive MCMC, and introduce the metrics by which we can measure its success. We then move to the adaptive case, outlining the pitfalls and the conditions under which they can be successfully evaded. This is done in the framework of the Robbins-Monro update, which is a stochastic approximation scheme. We then move to implement and test variations of adaptive schemes on toy models. Having done this we apply the adaptive schemes to a more complex, network based model.

Conceived under the supervision of Dr Sam Livingstone

Department of Statistical Science
University College London
14/08/2020

Contents

Chapter 1

Introduction

1.1 Introduction

1.1.1 Overview

Markov chain Monte Carlo (MCMC) algorithms are a class of algorithms which allow their user to sample from probability distributions. They permeate the work of much of academia, from artificial intelligence [1] to genomics [2], which is natural given that probability distributions are indispensable to the modern practice of scientific inquiry. The advent of big data, combined with Bayesian theory, leaves the practitioner with vastly complex posterior distributions they wish to infer from. Markov chain Monte Carlo is an ideal tool to do so.

1.1.2 Theoretical Underpinning

In its weak form, the Law of Large Numbers states that given a function f defined on the state space χ along with a sequence of independent, identically distributed (iid) variables $X_1, X_2, ..., X_n$, from a distribution π, $f(X_1), f(X_2), ..., f(X_n)$ are also iid, and their sample mean converges to $E_\pi(f)$ in probability, provided that this expectation is finite. What this means is that if we could take independent samples from a distribution π, such that we wished to evaluate some integral in the form $E_\pi(f)$, whether for inference purposes or otherwise, all we need do is take more and more samples, apply f to each one, and calculate the sample mean to achieve better and better approximations on average of that integral. We call π the target distribution. Markov chain Monte Carlo steps in when we cannot take independent samples from the distribution, but, we can formulate a Markov Chain (MC) whose limiting distribution is that which we wish to sample from. In this report we consider only discrete time Markov Chains. A discrete time MC is a sequence $\{X_k\}$ of random variables which we call 'states' such that the following holds:

$$P(X_{k+1} \in A | X_1 = x_1, ..., X_k = x_k) = P(X_{k+1} \in A | X_k = x_k)$$

for $A \subseteq \chi$. We call the function $P(X_{k+1} \in A | X_k = x) := P(x, A)$ the kernel of the Markov chain, and define the notation

$$P^n(x, A) := P(X_n \in A | X_1 = x)$$

Whenever you see $\pi(A)$ in the text that follows, we define it

$$\pi(A) := \int_{x \in A} \pi(x) dx$$

The Markov Chain Law of Large Numbers

When the samples $\{X_k\}$ form a MC the Law of Large Numbers in its form above no longer applies. This is due to the fact that consequent terms in a MC are not in general independent. To resolve this quandry we turn to the Markov Chain Strong Law of Large Numbers (MC SLLN): if $E_\pi(|f|) < \infty$ then

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(X_i) = E_\pi(f)$$

where $\{X_k\}$ has π as its unique equilibrium distribution. Armed with the notation that defines a MC, we can understand what an equilibrium distribution is. A chain $\{X_k\}$ with kernel P has π as its equilibrium distribution when

$$\lim_{n\to\infty} P(X_n \in A|X_1) = \pi(A)$$

where $X_1 \sim \mu$ for an arbitrary probability distribution μ.

Conditions for Convergence to a Unique Equilibrium Distribution π

As described above, the MC SLLN holds when π is the unique equilibrium distribution of the MC. The conditions under which this is guaranteed to happen are as follows

- π-irreducibility
  - π-irreducibility can be described as, given any starting point in the state space, the probability that the MC will visit a region A such that $\pi(A) > 0$ is non-zero.
- π-invariance
  - We have that the kernel P of a MC $\{X_k\}$ is π-invariant when

    $$X_i \sim \pi \implies X_{i+1} \sim \pi$$

    and, by induction,

    $$X_i \sim \pi \implies X_{i+n} \sim \pi$$

    for all $n > 0$.
- aperiodicity
  - Aperiodicity occurs if there is no disjoint collection of subsets of the state space $A_0, ..., A_{k-1}$ such that

    $$x \in A_i \implies P(x, A_{(i+1)\text{mod } k}) = 1$$

    i.e. when there is no cycle of events such that occurrence of one guarantees the occurrence of the next.

Often aperiodicity and irreducibility are easier to prove; invariance being the harder condition to satisfy, at least for the forms of MCMC covered in this report. There exist other forms in which irreducibility, say, is much more difficult. We will go on to describe a condition under which invariance is guaranteed.

The Markov Chain Central Limit Theorem

We also have a Central Limit Theorem (CLT) at our disposal: the Markov Chain CLT, which states that

$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} f(X_i) - E_\pi(f)\right) \to N(0,\sigma^2)$$

in distribution, where

$$\sigma^2 = \text{Var}(f(X_1)) + 2\sum_{k=1}^{\infty} \text{Cov}(f(X_1), f(X_k))$$

and $X_1 \sim \pi$. As we might expect of a stronger and more descriptive result, the Markov Chain CLT applies under stricter conditions than the MC SLLN.

Conditions for the Markov Chain Central Limit Theorem

There are many sets of conditions under which the MC CLT holds. The conditions here are so selected because they are most easily identified as satisfied by the chains in this report, others can be found in [3] along with proofs. Not only do we need that π is the chain's equilibrium distribution, but we require that our chains are geometrically ergodic to π, π-reversible, and that $E_\pi(f^2) < \infty$. Geometric ergodicity to π is where

$$\|P^n(x,.) - \pi(.)\| \leq M(x)\rho^n$$

for $n = 1, 2, 3, ...$, $\rho < 1$, and $M(x) < \infty$ for π-almost everywhere in the state space χ. The metric of distance ($\|.\|$) we use is total variational distance.

There are a few unfamiliar concepts in this set of conditions, so we explain them in turn:

- Total Variational Distance
  - This can be informally described as the worst disparity between the probability mass of two probability density functions, as they range over all the subsets in the state space. Formally we have
$$\|\nu_1(.) - \nu_2(.)\| = \sup_{A\subseteq\chi} |\nu_1(A) - \nu_2(A)|$$
    where each ν_i is a probability measure. Thus the total variational distance can be thought of as the 'closeness' of two probability distributions, and can be used to measure the disparity between the distribution of a state in our MC, and the target π
- π-almost everywhere
  - If a property holds π-almost everywhere, then there exists a set $A \subseteq \chi$ with $\pi(A) = 0$ such that all points $x \in \chi/A$ have that property.
- π-reversibility
  - This occurs whenever
$$\pi(dx)P(x, dy) = \pi(dy)P(y, dx)$$
    for all $x, y \in \chi$. This definition is somewhat opaque on first inspection, but we can interpret as follows: the left side of the equation is like the 'flow' of the chain from state x to state y, and the right hand of the equation is like the 'flow' from state y to state x. So under equality we would expect the forward-time chain to look like the reverse-time chain, hence reversibility. In an earlier section we mentioned a condition under which invariance is guaranteed. This is that condition. Proof of this fact is in the appendix.

It is important to give the MC CLT a theoretical grounding, because of the information it provides surplus to that of the MC SLLN. Not only will we know *that* the kernel of the MC will converge to π, we can also quantify over our uncertainty as to their disparity at a particular point n in the chain. So we would imagine that our ability to infer σ as defined above is key. It is not coincidental then that σ links many of the measures of a MC's health, as we will explain in section 1.1.5. This being said, geometric ergodicity is in practice a difficult and technical result to prove. In the vast majority of cases in which a new MCMC method is conceived, geometric ergodicity is not proved concurrently, although it may be the case that a practitioner can find a different set of conditions under which the MC CLT holds.

1.1.3 Motivation

Bayes' theorem codifies a method of updating our uncertainty. It is the foundation of huge tranches of modern science, and statistical methodology. It states that, with events A and B we should update our credence based on the following rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Hence if we let B be some data we observe, and A be some hypothesis, the above rule can be used to provide a codification for the empirical foundation of science, which it does in many cases. In terms of probability density functions the rule becomes

$$\pi(\theta|x) = \frac{P(x|\theta)\pi(\theta)}{P(x)}$$

The expression on the left hand side is known as the posterior density, $P(x|\theta)$ is the likelihood, and $\pi(\theta)$ is the prior density. Using the Law of Total Probability gives

$$\pi(\theta|x) = \frac{P(x|\theta)\pi(\theta)}{\int_{\theta\in\Theta} P_{X|\theta}(x)\pi(\theta)d\theta}$$

Oftentimes the integral in the denominator is intractable, in theory or in practice. Hence we need to make inferences about a distribution

$$\pi(\theta|x) \propto P(x|\theta)\pi(\theta)$$

which we only know up to multiplication of an unknown constant. This is the condition under which we can apply MCMC algorithms, and hence why they are so important to the modern practitioner.

1.1.4 The Metropolis-Hastings Algorithm

The Metropolis-Hastings Algorithm provides a framework within which we can guarantee π-irreducibility, π-reversibility, and aperiodicity. Conceived in 1953 by Metropolis et al. in [4] and generalised in 1970 by Hastings in [5], it takes the general form:

Algorithm 1: Pseudocode detailing the Generic Structure of a Metropolis-Hastings algorithm

```
Initialise $X_0$;
**for** *i in 1,...,n* **do**
    Given $X_i$;
    1. Propose the next state in the MC $Y_{i+1}|X_i \sim Q(X_i, .)$;
    2. Set $X_{i+1} = Y_{i+1}$ with probability $\alpha(X_i, Y_{i+1})$ and $X_{i+1} = X_i$ otherwise;
**end**
```

If we let $P(x, dy)$ be the kernel of the full MC, satisfying

$$\pi(dx)P(x, dy) = \pi(dy)P(y, dx)$$

gives us reversibility. Note that the algorithm involves sampling from a distribution $Q(X_i, .)$, which we call the *proposal distribution*, and accepting with a certain probability α. If we let $q(x, A)$ be the density of the proposal distribution, the requirement we wish to satisfy for reversibility then becomes

$$\pi(x)q(x, y)\alpha(x, y) = \pi(y)q(y, x)\alpha(y, x)$$

in terms of densities. The form of $\alpha(x, y)$ then dictates reversibility, given a general $q(x, y)$. The Metropolis-Hastings algorithm has that

$$\alpha(x, y) = \min\left(1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right)$$

This form conveniently allows us to only know π up to a constant, as described in section 1.1.3. The form also means that for a y which is accepted and not equal to x we have

$$\begin{aligned}
\pi(x)q(x, y)\alpha(x, y) &= \pi(x)q(x, y)\min\left(1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right) \\
&= \min(\pi(x)q(x, y), \pi(y)q(y, x)) \\
&= \pi(y)q(y, x)\min\left(1, \frac{\pi(x)q(x, y)}{\pi(y)q(y, x)}\right) \qquad (1.1) \\
&= \pi(y)q(y, x)\alpha(y, x)
\end{aligned}$$

□

which satisfies π-reversibility. Note also that

$$q(x,y)\alpha(x,y) = \min\left(1, \frac{\pi(y)q(y,x)}{\pi(x)}\right)$$

is greater than zero for $\pi(y) > 0$ so long as we construct the proposal correctly i.e. that we have $q(y,x) > 0$ for all y such that $\pi(y) > 0$ and for all $x \in \chi$. This then gives us

$$\pi(A) > 0 \implies P(x,A) > 0$$

for all $x \in \chi$ which satisfies π-irreducibility.

Aperiodicity is guaranteed whenever $q(x,.)$ and π are positive in a small enough region around x. Let $A_0, ..., A_{k-1}$ be any collection of disjoint subsets of the state space, and $\epsilon > 0$ be the radius of a ball $B(x_i, \epsilon)$ centred around x_i such that

$$0 < q(x_i, B(x_i,\epsilon)) < q(x_i, A_i)$$

and

$$\pi(B(x_i,\epsilon)) > 0$$

whenever $x_i \in A_i$ for any $i \in \{0, ..., k-1\}$. Then the existence of such an ϵ guarantees that the MC has a non-zero probability of moving *within* any of these subsets $A_0, ..., A_{k-1}$, and so is not guaranteed to move to the 'next' subset in the cycle, which would violate aperiodicity if it were. Existence of such an ϵ depends on the nature of the state space and the densities π and $q(x,.)$, but we will assume existence for the algorithms discussed in this report.

In many cases, MCMC runs are performed with a symmetric proposal. This means that $q(x,y) = q(y,x)$ and so the acceptance probability becomes $\min(1, \pi(y)/\pi(x))$, and we would expect to accept a move to a higher region of density in π. Using a proposal $Q(X_i,.) = N(X_i, h^2 I)$ where I is the identity matrix, gives the symmetric Random Walk Metropolis (RWM) algorithm.

1.1.5 Measuring Success

Say we've run an MCMC algorithm. How do we know whether we have enough samples? How are we to evaluate its quality? We need to establish some condition such that when it is reached, we can stop the algorithm, or, if our computing time is limited, we need a way of comparing different algorithms that run in that time.

Bias-Variance Decomposition

One way of estimating the quality of a statistical estimator $\bar{f}$ is to look at its mean squared error as compared to the true value $\hat{f}$. This quantity can be further decomposed into the bias and the variance, leading to the well known identity

$$E((\bar{f} - \hat{f})^2) = \text{Bias}(\bar{f})^2 + \text{Var}(\bar{f})$$

where

$$\text{Bias}(\bar{f}) = \hat{f} - E(\bar{f})$$

and

$$\text{Var}(\bar{f}) = E((\hat{f} - E(\hat{f}))^2)$$

As the earlier sections intimated, the statistical estimator we are interested in is $\bar{f} = 1/n \sum_{i=1}^{n} f(X_i)$ where the X_i are states in the MCMC run.

In the following section we will look at evaluating aspects of the MCMC run with respect to how they affect the bias and the variance of the estimator they produce.

Convergence

Below is a traceplot of a MCMC run in red, plotted with the mean of the target distribution in green. The algorithm used is a symmetric RWM algorithm, and so the acceptance probability takes the form $\min(1, \pi(y_{i+1})/\pi(x_i))$. Because we know the mean of the target, we can surmise the cause of the behaviour during the initial ≈ 125 iterations: that the chain begins in a region of low density, so points y closer to the mean are more readily accepted since they have a higher

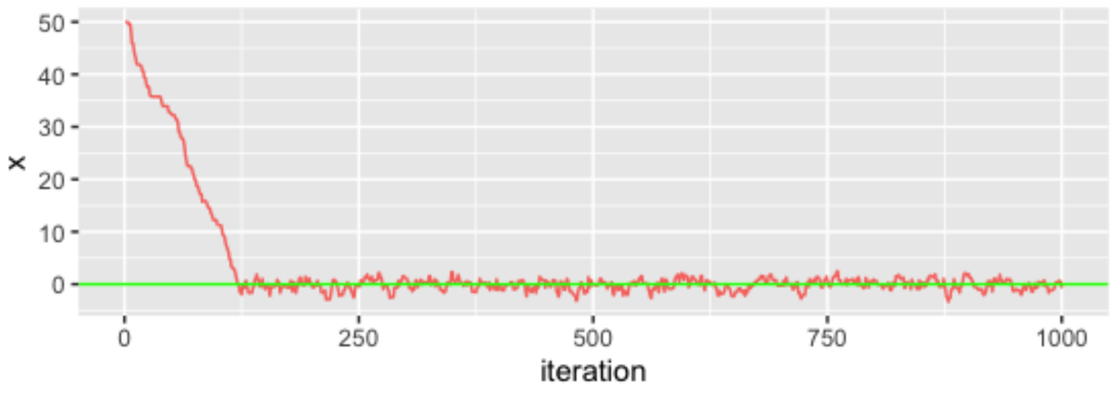

Figure 1.1: Simple example of a Symmetric RWM with the trace in red, and the target mean in green

density, and the acceptance probability is usually close to 1 for these points. Therefore the chain drifts until it oscillates or converges around the true mean. This initial 'drift' is known as the burn-in.

At least if our task is to estimate the target mean, its clear from the example that if we disregard the burn-in in our statistical estimate $\bar{f}$ we reduce the bias. This is especially so if the number of iterations in the chain is low. However in higher dimensional settings it becomes increasingly difficult to estimate how many terms to disregard just by the impression given by a multiple one dimensional traceplots such as the one in figure 1.1. Also if the target distribution is heavy tailed and the proposal is not, the ratio in the acceptance probability can tend to 1 as $|x_i| \to \infty$ regardless of the proposed state, so in higher dimensions we can be fooled into thinking we've converged when the chain isn't close to the target mode. So we want some measure of the time by which the chain converges.

We can use the Gelman-Rubin diagnostic [6] to evaluate convergence. This diagnostic works by starting many different chains in disparate regions of the state space. It is, roughly speaking, a measure of the variation within the chains divided by a measure between the chains. We would expect these variations to be similar as the chain achieves convergence since when the chains converge, jumping between the chains should be indistinguishable from the jumps between states within a particular chain. So we look for values of the diagnostic near 1. This then rectifies our problem of heavy tails since even if each individual chain looks like it has converged, the variance between the chains will be large, and so the diagnostic will be far from 1[1]. Below we see the Gelman-Rubin diagnostic plotted over time for four chains that use the same algorithm as the one depicted in figure 1.1 started from four different values.

Mathematically the diagnostic can be described as follows: imagine we run k chains for n iterations, and define $f_{ij} := f(X_{ij})$ where X_{ij} is the ith state in the jth chain, and f is the function we wish to take expectations of under the target. We then define the within-chain sample mean

$$\bar{f}_{.j} := \frac{1}{n}\sum_{i=1}^{n} f(X_{ij})$$

the full sample mean

$$\bar{f}_{..} := \frac{1}{nk}\sum_{i,j} f(X_{ij})$$

the between-chains variation

$$B := \frac{n}{k-1}\sum_{j=1}^{k}(\bar{f}_{.j} - \bar{f}_{..})^2$$

[1]This is either the case, or it may be that the chains have actually converged to different modes of a multimodal distribution. In this scenario we would hope a practitioner uses more than a single instance of a Gelman-Rubin diagnostic to assess the health of the MCMC, or that their domain-specific knowledge is such that they suspect multimodality.

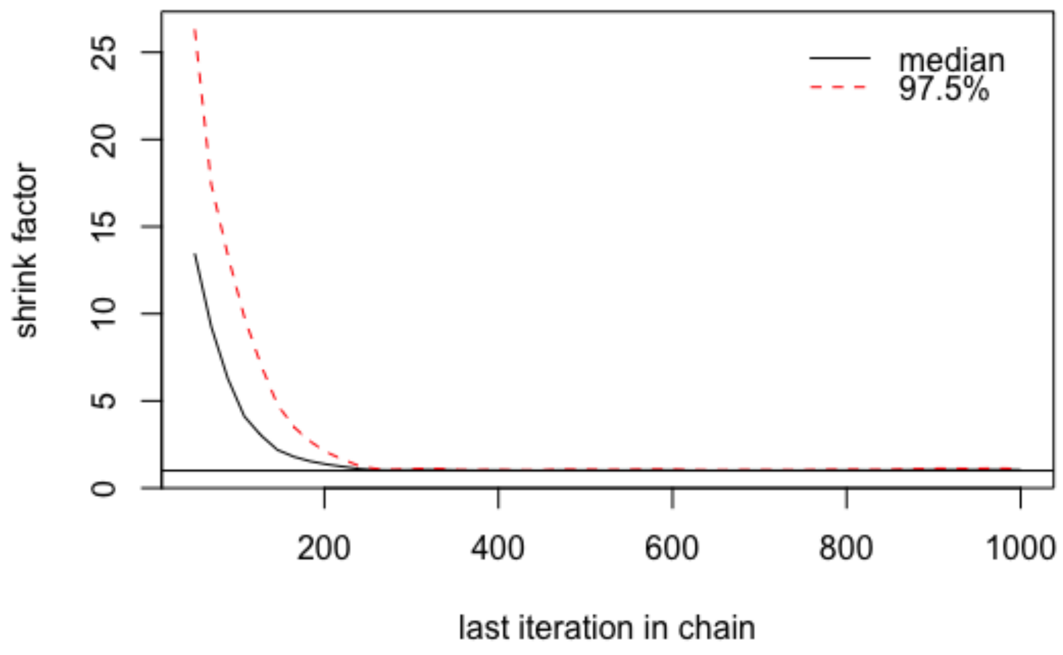

Figure 1.2: Gelman-Rubin diagnostic plotted over time for four chains started from the values $\{-50, -25, 25, 50\}$

the within-chain variation for the jth chain

$$W_j := \frac{1}{n-1}\sum_{i=1}^{n}(f_{ij} - \bar{f}_{.j})^2$$

and the estimator for the within-chains variation

$$W := \frac{1}{k}\sum_{j=1}^{k} W_j$$

These definitions should be familiar to anyone intimate with ANOVA (ANalysis Of VAriance). The Gelman-Rubin diagnostic is then defined

$$\widehat{R} := \sqrt{\frac{\frac{n-1}{n}W + \frac{1}{n}B}{W}}$$

Mixing

Once the chain has converged, we need some way of assessing how well it explores, and therefore represents, the target. As a quick way to diagnose problems, we can again look at the trace and autocorrelation plots of the MC. Clearly the autocorrelation of the ideal chain would drop to and remain at a small value for all lags, although this never happens in practice. Whilst these plots are useful for detecting obvious pathologies, we would like a way of quantifying how well the chain explores.

We can frame the mixing of the chain in terms of the variance term in the bias-variance decomposition. Let $\bar{f}_n$ be $\frac{1}{n}\sum_{k=1}^{n} f(X_k)$ where $\{X_1, ..., X_n\}$ is our MCMC run (as above), and $\hat{f}_n$ be $\frac{1}{n}\sum_{k=1}^{n} f(\widehat{X_k})$ where $\{\widehat{X_1}, ..., \widehat{X_n}\}$ are independent samples from π. We would ideally like the variance of $\bar{f}_n$ to be close to that of $\hat{f}_n$. In fact, you can calculate that

$$\text{Var}(\bar{f_n}) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\text{Var}(f(X_i)\right) + 2\sum_{i<j}\text{Cov}(f(X_i), f(X_j)))$$

If we set X_1 to be the first sample after the burn-in, and n to be the amount of samples after the burn-in (in other words we believe that $X_1 \sim \pi$) the expression becomes

$$\text{Var}(\bar{f_n}) = \frac{1}{n}\left(\text{Var}(f(X_1)) + 2\sum_{k=2}^{n}(\frac{n-k}{n})\text{Cov}(f(X_1), f(X_k))\right)$$

Since $\text{Var}(\hat{f_n}) = \text{Var}_\pi(f)/n$ we have that

$$\frac{\text{Var}(\hat{f_n})}{\text{Var}(\bar{f_n})} = \frac{1}{1 + 2\sum_{k=2}^{n}\text{Corr}(f(X_1), f(X_k))}$$

This gives us the next metric by which to judge our MCMC run: the Effective Sample Size (ESS):

$$\text{ESS} := \left(\lim_{s\to\infty}\frac{\text{Var}(\hat{f_s})}{\text{Var}(\bar{f_s})}\right)n = \frac{n}{1 + 2\sum_{k=2}^{\infty}\text{Corr}(f(X_1), f(X_k))}$$

(notice the index over s in the limit to avoid confusion). This is in theory how many samples from π you would need to take, to achieve an estimator with the same variance as $\hat{f_n}$. This is a nice definition, although it comes with some pitfalls. Firstly there is the issue of dealing with the infinite sum of correlations, although in practice this is unimportant since sensible methods of truncating it exist, and we can estimate correlations sufficiently well given a chain length above a certain size.

More seriously there are clearly ways of violating the spirit of the above definition. Say we want to estimate $E_\pi(X)$, which would make f the identity, and our 'MCMC' run was the following: $X_1 \sim N(0,1)$, $X_2 = -X_1^5$, and $X_k = \text{constant}$ for $k > 2$. Then we have that

$$\begin{aligned}\text{Corr}(X_1, X_2) &= \frac{\text{Cov}(X_1, -X_1^5)}{\sqrt{\text{Var}(X_1)\text{Var}(X_1^5)}}\\ &= -\frac{E(X_1^6)}{\sqrt{E(X_1^{10})}}\\ &= -\frac{5!!}{\sqrt{9!!}}\\ &= -\frac{15}{\sqrt{945}}\\ \implies \text{ESS} &= \frac{n}{1-\frac{30}{\sqrt{945}}}\\ &\approx 41.5n\end{aligned} \tag{1.2}$$

where we use

$$n!! := \prod_{k=1}^{\frac{n+1}{2}}(2k-1)$$

for n odd, and

$$n!! := \prod_{k=1}^{\frac{n}{2}}(2k)$$

for n even, and the fact that

$$E(X^p) = \begin{cases}0, & \text{if } p \text{ is odd and}\\ (p-1)!!, & \text{if } p \text{ is even}\end{cases}$$

when $X \sim N(0,1)$. The fact that the ESS is larger than the amount of steps in the chain is completely nonsensical. This is hinting at the fact that the ESS is only reliable under certain conditions, and hence we should take a holistic approach in evaluating our MCMC run: using an ensemble of success metrics.

The form of the ESS prompts another metric: the acceptance rate. Recall from section 1.3 that each proposed next state in the chain is accepted with some probability. Say our chain has converged. If the proposals are not bold enough (i.e. we don't propose to move a great distance in the state space) the rate at which they are accepted will be high since we stay in a region of high density. This means we have high autocorrelation, and thus a smaller ESS. If the proposals are too bold, the proposed points will be outside the regions of high density, and will therefore be rejected. Hence we risk staying in the same position for many iterations, and the autocorrelation will still be high. So we want our acceptance rate to be within a desirable interval.

In figures 1.3 and 1.4 we show the traceplots, autocorrelation plots, ESS, and ESJD (introduced later this section) of two example MCMC runs: one with a high acceptance rate, one with a low acceptance rate. As we can see, the high acceptance chain meanders round without making any bold moves, and the low acceptance chain gets 'stuck' and then jumps a great distance suddenly.

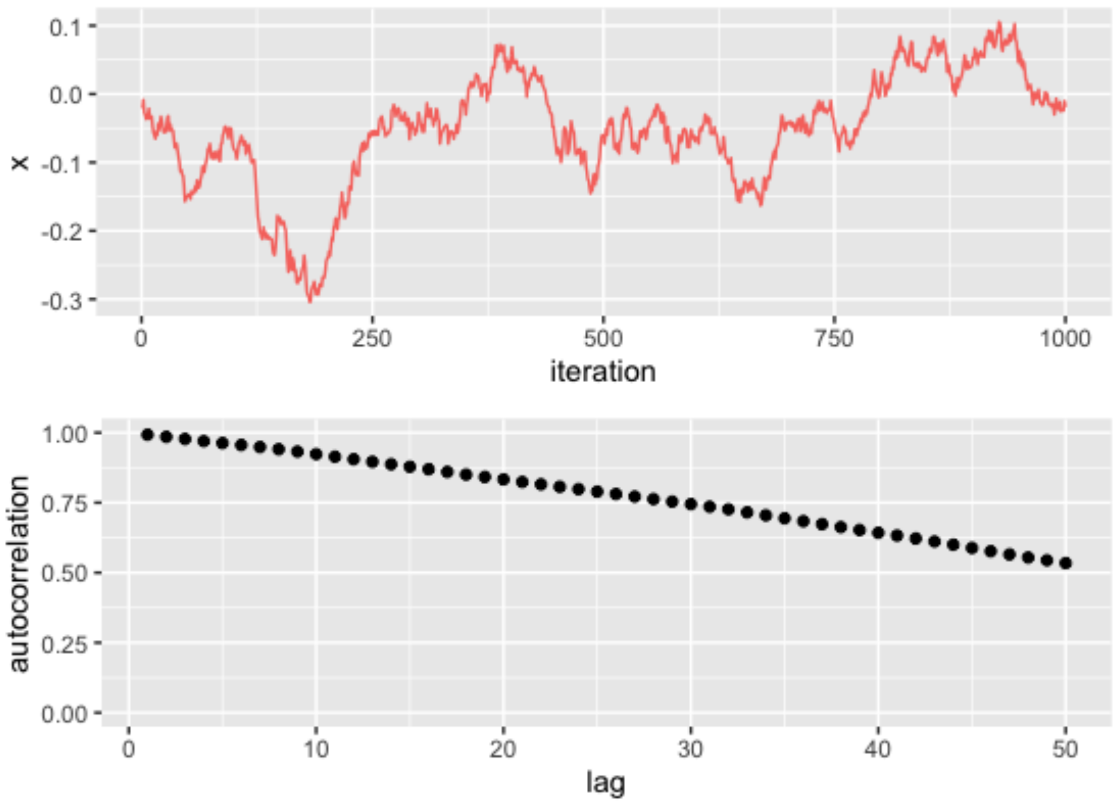

Figure 1.3: Traceplot and autocorrelation of an MCMC run with a high acceptance rate. The ESS here is 3.185, and the ESJD is 1.07×10^{-7}

Roberts, Gelman, and Gilks (1997)[7] showed that, when implementing RWM with $\Sigma = \sigma^2 I_d$ on target densities of the form

$$\pi(x_1, ..., x_d) = \prod_{i=1}^{d} f(x_i)$$

as $d \to \infty$ the optimal acceptance rate is 0.234, a limiting result which has been shown to apply to lower dimensional targets $d > 4$. In one dimension the optimal acceptance rate is 0.44. If the target could decompose as above, then you would just implement a separate MH algorithm on each component removing the need for MCMC. Hence Roberts and Rosenthal (2001) [8] extended the results to targets of the form

$$\pi(x_1, ..., x_d) = \prod_{i=1}^{d} k_i f(k_i x_i)$$

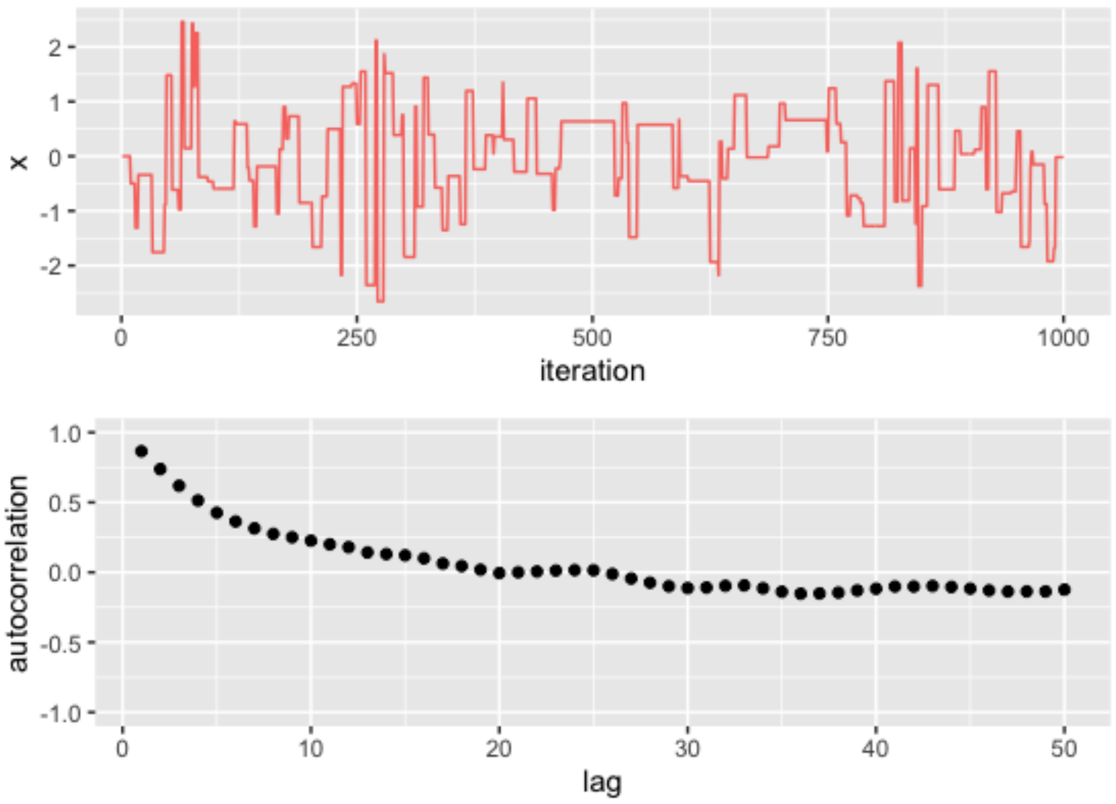

Figure 1.4: Traceplot and autocorrelation of an MCMC run with a low acceptance rate. The ESS here is 97.853, and the ESJD is 2.60×10^{-4}. Notice the scale of the y-axis of the autocorrelation

where $k_1, ..., k_d$ are iid. In the case that the target is of the form $N(0, \Sigma)$ and the proposal is $N(0, \Sigma_p)$ they go on to show that the ideal proposal covariance is proportional to the target covariance.

Another measure of good mixing is the expected square jump distance (ESJD) which is defined

$$\text{ESJD} = E((X_1 - X_2)^2)$$

where X_1 is presumed to be drawn from π and X_2 is generated using the algorithm in section 1.1.4. It is estimated using

$$\widehat{\text{ESJD}} = \frac{1}{1-n}\sum_{i=1}^{n-1}(X_i - X_{i+1})^2$$

Here the ESJD is presented in one dimension, but we can look at visualisations and summary statistics of the ESJD over all dimensions to get a fuller picture, or even redefine it as ESJD $= E(\|X_1 - X_2\|^2)$ for some distance metric $(\|.\|)$. We would like to maximise this quantity, as it describes how quickly the MC explores the space. Since the ESJD is defined in equilibrium $(X_1 \sim \pi)$, we can decompose it as follows:

$$\text{ESJD} = 2\text{Var}(X_1)(1 - \text{Corr}(X_1, X_2))$$

Hence we wish to minimise the correlation between consequent steps in the chain, which happens to be the first term of the sum in the denominator of the ESS.

Chapter 2

Adaptation

2.1 Introduction

In non-adaptive MCMC both the proposal and decision step are agnostic to the prior values of the MC, and they are carried out in the same way regardless of the iteration. This can be fine although, as has been discussed, the more the covariance structure of the proposed states differs from the covariance structure of π, the less efficient the chain will be. The more our proposal distribution looks like the target, the better. We can use the previous values in the chain, along with results like the MC SLLN and MC CLT to continually update the proposal, and so for each new state in the chain we can update our conception of the target.

The trouble with adaptive MCMC is that there are numerous pitfalls to tip-toe around. For one we risk losing the Markov property of the chain, and even if we retain it we may lose ergodicity [9]. Assuming we are guaranteed these properties, we still need to choose the right adaptive strategy for the given algorithm so that we balance the heavy computation cost per adaptation (at least $O(d^2)$ to update the covariance matrix, where d is the dimension of the state space), against the overall gains in efficiency.

2.2 Motivation

In the absence of adaptive MCMC, practitioners working with complex distributions they wish to sample from, will still need to tailor the proposal to achieve an efficient algorithm. For instance, it may take too much computational investment to be satisfied with a single long run, which is inefficient due to the ill-suitedness of the proposal. Therefore the practice is to make a series of short initial runs to drill down the inefficiencies. For example, if you make a short run, all of whose proposals are rejected, you would reduce the scale at which you propose and try again. The idea in adaptive MCMC is to absorb this process into the final MCMC run from which you infer the expectation under the target.

Consider as a simple motivating example applying MCMC to a distribution in two dimensions, that you know is normally distributed, centred at the origin, and heterogeneous in these dimensions. Mathematically

$$\pi = N\big(0, \left(\begin{smallmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{smallmatrix}\right)\big)$$

is the state of our knowledge, with $\lambda^1 \neq \lambda^2$ and $\lambda^1, \lambda^2 \in \mathbf{R}$, superscripts acting here as indices not powers. If we had hyperpriors for the scales, we could run a series of short MCMCs starting with the proposal scales at their hyperprior means, and updating based on the output. Adaptive MCMC provides a more elegant solution. Here we implement the following algorithm:

Algorithm 2: Pseudocode for simple scale-updating MCMC

```
Initialise $X_0$, $\lambda_0^1$, and $\lambda_0^2$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\lambda_i^1$, and $\lambda_i^2$;
    for $k$ in $\{1, 2\}$ do
        1. Sample $Y_{i+1} \sim X_i + e_k N(0, \lambda_i^k)$;
        2. Accept with probability $\alpha = \min(1, \pi(Y_{i+1})/\pi(X_i))$ otherwise stay put.;
        3. Update $log(\lambda_{i+1}^k) = log(\lambda_i^k) + 0.01(\alpha - \alpha_*)$;
    end
end
```

where e_k is the unit vector along the kth dimension, α_* is the optimal acceptance rate, which, as discussed in section 1.1.5 of the first chapter, is 0.234. Here λ_i^k is our best guess of the scale in the kth direction at iteration i. We also ran a chain with no scale update to see how it would compare.

We started the chain at the origin, and initialised the scales as $(\lambda_0^1, \lambda_0^2) = (0.001, 0.001)$, and ran the chain for 10^4 iterations. In the figures 2.1-3 we compare the trace plot of the adaptive algorithm, the non-adaptive algorithm, and if we were to sample from the actual target, whose scales are $(\lambda_*^1, \lambda_*^2) = (10, 1)$. The ESS of the adaptive run is $(1575, 1556)$ (an ESS per dimension) compared to $(232, 4.36)$ in the non-adaptive case.

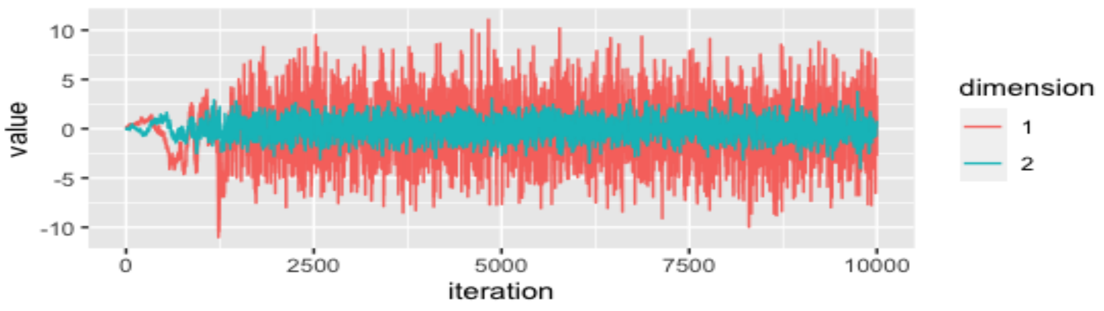

Figure 2.1: Trace of the adaptive algorithm

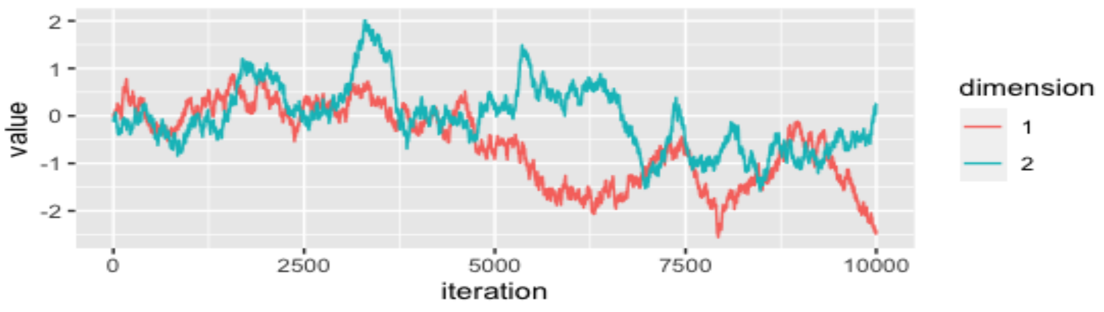

Figure 2.2: Trace of the non-adaptive algorithm

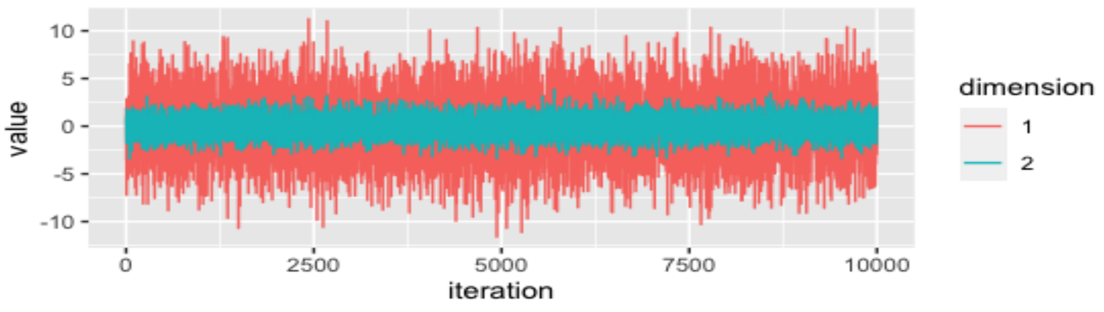

Figure 2.3: Trace of the target

The figures show that the adaptive algorithm is robust to incorrect scaling, where the non-adaptive algorithm performs poorly, with an acceptance rate that is too high, yielding a high-autocorrelation and therefore poor ESS. Figures 2.4 and 2.5 show the proposal scales λ_1 and λ_2. Note that their values are approximately 6 times what they should be by the end of the run, but that their ratio is close to the ratio of the scales of the target distribution. This is what we need for the proposal covariance to be proportional to the target covariance, and so is optimal in the sense discussed at the end of section 1.1.5, where the optimal acceptance rate is introduced.

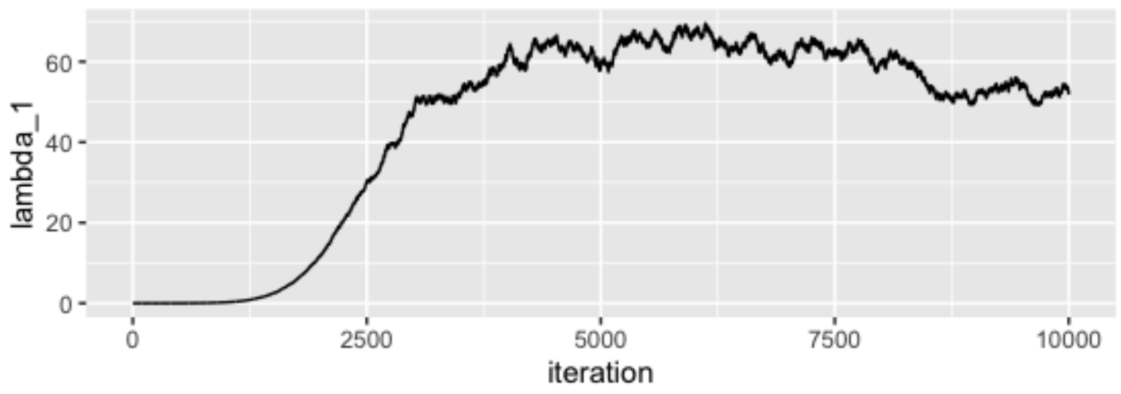

Figure 2.4: Plot of λ^1 over time

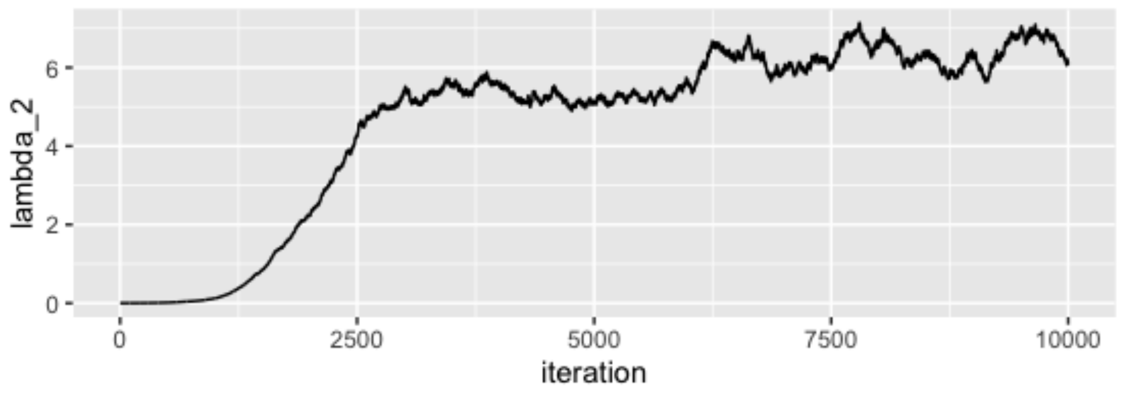

Figure 2.5: Plot of λ^2 over time

2.3 Generic Adaptive Structure

The generic structure of an adaptive MCMC is as follows:

Algorithm 3: Pseudocode detailing the Generic Structure of an adaptive Markov Chain Monte Carlo algorithm

```
Initialise $X_0$, $\theta_0$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\theta_i = \theta_i(\theta_0, X_0, ..., X_i)$;
    1. Propose the next state in the MC $Z_{i+1}|(\theta_0, X_0, ..., X_i) \sim Q_{\theta_i}(X_i, .)$;
    2. Use some decision procedure such that either $X_{i+1} = Z_{i+1}$ or $X_{i+1} = X_i$;
    3. Update $\theta_{i+1} = \theta_{i+1}(\theta_0, X_0, ..., X_{i+1})$;
end
```

where $Q_{\theta_i}(X_i, .)$ is the proposal kernel.

Here θ_i represents those parameters the practitioner feels it is salient to update to ensure the efficiency of the MCMC run, and they serve as parameters in the proposal distribution. The hope is that, by learning better values of θ, we have a better idea of the target.

2.4 Conditions For Ergodicity

As mentioned in section 1.1.5 ergodicity may be lost in the adaptive setting. Take the following as a simple example: we have the target $\pi = N(0,1)$ and we run the algorithm in the pseudocode in algorithm 3, with the MH acceptance and the proposal $Z_{i+1}|X_i \sim N(X_i, 1/|X_i|^2)$. So we wish to propose moves based on the current state of the chain. In practice the MC gets 'stuck' for long periods of time close to zero, as the variance of the proposal is so high that many of the proposed states are rejected. We show a comparison between the histogram of the MC, and an $N(0,1)$ histogram in the figure 2.4 along with a traceplot in 2.5.

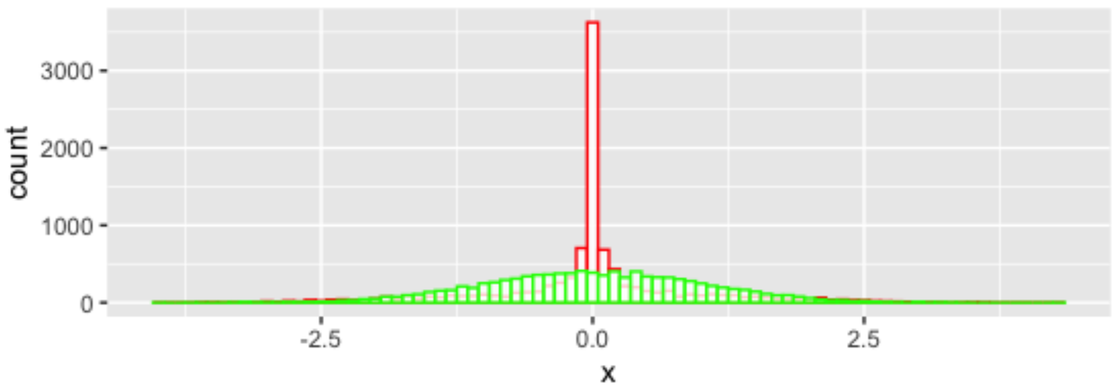

Figure 2.6: Histogram of the maladaptive MCMC (red) compared with the target $N(0,1)$ (green)

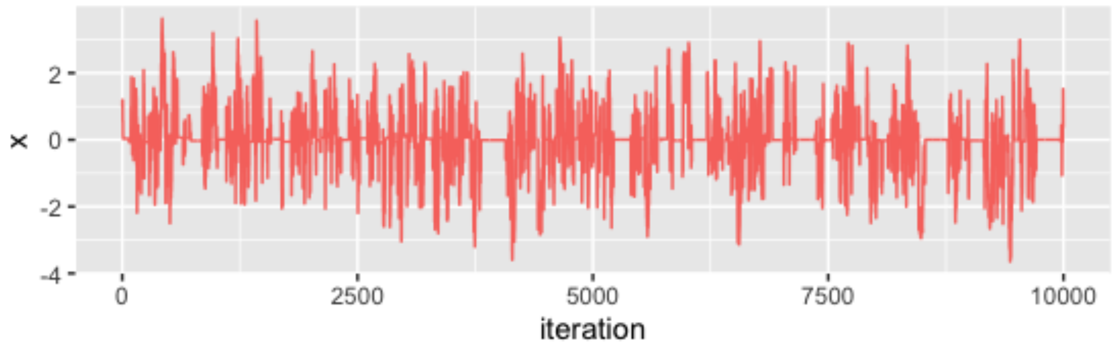

Figure 2.7: Trace of the maladaptive MCMC

So we would like some conditions under which our chain remains ergodic to π. Luckily Roberts and Rosenthal (2005) [9] show that $\lim_{n\to\infty} \sup_{A\subset\chi} \|P(X_n \in A) - \pi(A)\| = 0$ (i.e. the total variational distance between the kernel and the target density converges to 0) and, for bounded $f : \chi \to \mathbf{R}$, $\lim_{n\to\infty} 1/n \sum_{i=1}^{n} f(X_i) = E_\pi(f)$ under the following conditions:

2.4.1 Diminishing Adaptation

Diminishing Adaptation is defined as the condition that

$$\lim_{n\to\infty} \sup_{x\in\chi} \|P_{\theta_{i+1}}(x,.) - P_{\theta_i}(x,.)\| = 0$$

in probability, where $P_\theta(x,.)$ is the kernel of the *full* MC (that is, of the proposal step together with the decision step) given the current state is x.

2.4.2 Bounded Convergence

Bounded Convergence is when the sequence

$$\{M_\epsilon(X_n, \theta_n)\}_{n=0}^{\infty}$$

is bounded in probability for $\epsilon > 0$ where

$$M_\epsilon(x, \theta) := \inf\{n \geq 1 : \|P_\theta^n(x, .) - \pi\| \leq \epsilon\}$$

can be described as the first time the kernel comes within a certain variational distance of the target. In most cases, certainly in the ones implemented in this report, Bounded Convergence is not violated. Diminishing Adaptation is a more interesting condition. It states that the amount by which the algorithm adapts converges to 0.

To understand its importance we follow Andrieu and Thoms' (2008) 'A Tutorial on Adaptive MCMC' [10] in an analysis of the asymptotic behaviour of

$$|\hat{E}_*(f(X_i)) - E_\pi(f(X))|$$

Where $\hat{E}_*(f(X_i)$ is the expectation of the process started at an arbitrary (θ, X). As is outlined in the tutorial, and explicitly shown in the appendix to this report, if the sequence of adaptive parameter values $\{\theta_k\}$ is such that there is a deterministic sequence $\{\gamma_k\}$ for which

$$|\theta_{i+1} - \theta_i| \leq \gamma_i$$

with $\gamma_i \to 0$ as $i \to \infty$, we can provide a reasonable upper bound for the above distance between expections. If we then view the condition $|\theta_{i+1} - \theta_i| \leq \gamma_i$ as part of Diminishing Adaptation, we see that it is integral to understanding ergodicity to π.

Take for a simple example, wanting to use the sample mean of all prior samples as the proposal mean μ. Then, after each new state X_{i+1} is achieved, we would update as follows:

$$\mu_{i+1} = \mu_i + \frac{1}{i+1}(X_{i+1} - \mu_i)$$

to get the new sample mean μ_{i+1}. Hence the amount by which we adapt converges to 0, but interestingly the sum over all adaptations might not. This is an example of the Robbins-Monro update which the tutorial exploits to great explanatory effect. Equally we can perform the following operation to update to the new empirical covariance:

$$\Sigma_{i+1} = \frac{1}{i+1}(i\Sigma_i + (i+1)\mu_i\mu_i^T - (i+2)\mu_{i+1}\mu_{i+1}^T + X_{i+1}X_{i+1}^T)$$

where the update is $O(\frac{1}{i})$ again. More generally we can reframe the update recursion in the proper Robbins-Monro form:

$$\theta_{i+1} = \theta_i + \gamma_{i+1}H_{i+1}(\theta_0, X_0, ..., Z_i, X_i, Z_{i+1}, X_{i+1})$$

as is done in the tutorial, where $\{Z_k\}$ is the sequence of proposed states. $\{\gamma_k\}$ is chosen to satisfy Diminishing Adaptation, and $\{H_k\}$ is chosen such that as it converges to 0, we achieve optimality of some kind, whether it be that some measure of the chain (like the acceptance rate) reach an optimal value, or that some feature of the chain (like the covariance) resembles a feature of the target. Clearly if we knew the exact form of the feature of the target, we wouldn't have to perform the adaptation, or the MCMC at all, so we use results like the MC SLLN to construct $\{H_k\}$ such that we achieve optimality in the limit, as in the above examples of the sample mean and covariance. We could also conceive of some measure such as the Kullback-Leibler divergence dependent on θ_{i+1}, where we only need know π up to a constant, that is optimised for $H_k \to 0$.

The tutorial illustrates a useful decomposition of H. Take, for example, our update step for the sample mean. We can decompose it as follows:

$$\mu_{i+1} = \mu_i + \frac{1}{i+1}(X_{i+1} - \hat{\mu_i}) + \frac{1}{i+1}(\hat{\mu_i} - \mu_i)$$

Here $\hat{\mu_i}$ is the sample mean where the samples are iid from the target distribution; inkeeping with the notation in the formulation of the ESS. The first component of the decomposition represents the ideal update: the improvement on our estimate of the sample mean, had we perfect knowledge of the target. The second term in the decomposition represents the noise due to the difference between the ideal sampling strategy and the MCMC sampling strategy. In the figure 2.6 we plot both of these quantities for the first adaptive algorithm introduced in the next chapter, where, instead of having $\gamma_i = O(i^{-1})$ it maintains a small but constant value:

$$\mu_{i+1} = \mu_i + 0.001(X_{i+1} - \hat{\mu_i}) + 0.001(\hat{\mu_i} - \mu_i)$$

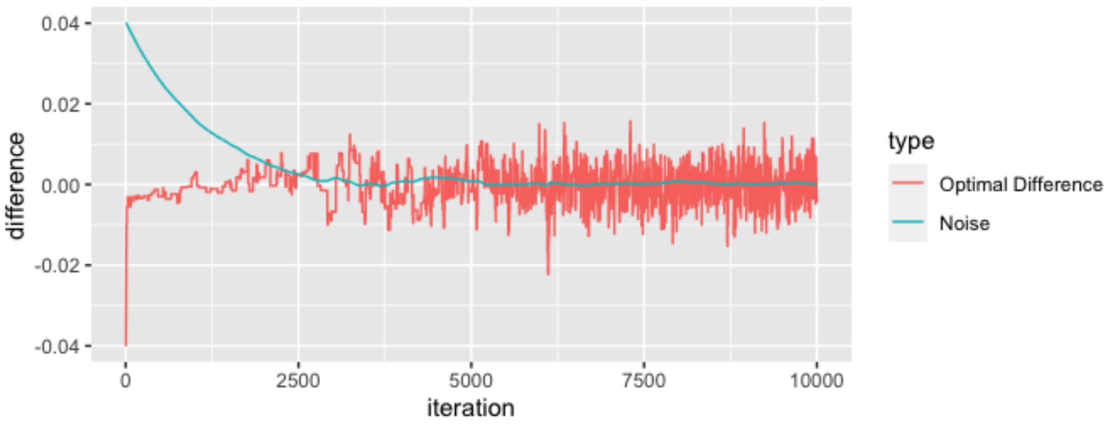

Figure 2.8: Demonstration of the theoretical quantities the Robbins-Monro update decomposes into

If we had let $\{\gamma_k\}$ be non-increasing, then since $|\mu_{i+1} - \mu_i| \to 0$ as $i \to \infty$ and, provided that the quantity $\hat{\mu_i} - \mu_i$ is sufficiently smooth in θ, we expect the sequence $\{\hat{\mu_i} - \mu_i\}$ of noise to average out to zero. We would then expect the trajectory described by $\mu_{i+1} = \mu_i + (i+1)^{-1}(X_{i+1} - \mu_i)$ to oscillate about the ideal trajectory $\mu_{i+1} = \mu_i + (i+1)^{-1}(X_{i+1} - \hat{\mu_i})$, where the magnitude of oscillations tend to 0 as $i \to \infty$.

More generally, the tutorial describes the following recursion, which is a noisy form of the Robbins-Monro algorithm:

$$\theta_{i+1} = \theta_i + \gamma_{i+1} h(\theta_i) + \gamma_{i+1}\xi_{i+1}$$

where

$$h(\theta) := E(H(\theta, X_0, ..., Y_i, X_i, Y_{i+1}, x_{i+1}))$$

and $\xi_{i+1} := H(\theta, X_0, ..., Y_i, X_i, Y_{i+1}, x_{i+1}) - h(\theta_i)$ is the "noisy" part of the update. The recursion can also be seen a noisy gradient algorithm. Rearranging,

$$\frac{\theta_{i+1} - \theta_i}{\gamma_{i+1}} = h(\theta_i) + \xi_{i+1}$$

so, provided that the contribution of the noise ξ_i cancels out on average, then if we interpolate between each value of the sequence $\{\theta_k\}$ and look at the trajectory over time, we would expect to see a solution to the ODE

$$\dot{\theta(t)} = h(\theta(t))$$

subject to a rescaling, with stationary points at $h(\theta(t)) = 0$.

Chapter 3

An Exploration of Adaptive schemes

3.1 Introduction

In order to better understand the state of the art, we implemented the following five adaptive schemes

- Adaptive Metropolis (AM) from Haario et al. 2001 [11]
- Componentwise AM with Componentwise adaptive scaling (CAM-CAS)
- Global AM with Componentwise adaptive scaling (GAM-CAS)
- AM with Principal Components Update (AM-PC)
- Localised Normal-Symmetric Random Walk Metropolis (Local RWM)

all of which are adaptive variants on Random Walk Metropolis. We tested them on a variety of target distributions; highlighting the strengths and weaknesses of each relative to the problem at hand. The structure of the schemes is taken from Andrieu and Thoms (2008)[10]

3.2 Pseudocode Descriptions

The schemes are described by the following pseudocode:

3.2.1 Adaptive Metropolis

Algorithm 4: Pseudocode detailing the Adaptive Metropolis Algorithm

```
Initialise $X_0$, $\mu_0$, and $\Sigma_0$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\mu_i$, and $\Sigma_i$;
    1. Sample $X_{i+1} \sim P^{SRWM}_{\mu_i,\Sigma_i}(X_i, )$;
    2. Update $\mu_{i+1} = \mu_i + \gamma_{i+1}(X_{i+1} - \mu_i)$ and
     $\Sigma_{i+1} = \Sigma_i + \gamma_{i+1}((X_{i+1} - \mu_i)(X_{i+1} - \mu_i)^T - \Sigma_i)$;
end
```

where $P^{SRWM}_{\mu_i,\Sigma_i}(X_i,)$ is the transition kernel of a Normal-Symmetric RWM Markov Chain, $\{\gamma_i\}$ is a non-increasing sequence of positive real numbers, to ensure the conditions discussed in the previous chapter.

3.2.2 Componentwise Adaptive Metropolis with Componentwise Adaptive Scaling

This algorithm chooses a direction at random out of the default basis of the state space, samples along that direction, and, based on the result of the sample, updates the scale of the distribution used to sample in that direction.

Algorithm 5: Pseudocode detailing the Componentwise Adaptive Metropolis with Componentwise Adaptive Scaling

```
Initialise $X_0$, $\mu_0$, $\Sigma_0$, and $\lambda_0^1$,...,$\lambda_0^n$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\mu_i$, $\Sigma_i$, and $\lambda_i^1$,...,$\lambda_i^n$;
    1. Choose a component $k \sim U\{1,...,n\}$;
    2. Sample $Y_{i+1} \sim X_i + e_k N(0, \lambda_i^k (\Sigma_i)_{k,k})$;
    3. Accept with probability $\alpha = \min(1, \frac{\pi(Y_{i+1})}{\pi(X_i)})$ otherwise stay put.;
    4. Update;
            i) $\mu_{i+1} = \mu_i + \gamma_{i+1}(X_{i+1} - \mu_i)$;
            ii) $\Sigma_{i+1} = \Sigma_i + \gamma_{i+1}((X_{i+1} - \mu_i)(X_{i+1} - \mu_i)^T - \Sigma_i)$;
            iii) $log(\lambda_{i+1}^k) = log(\lambda_i^k) + \gamma_{i+1}(\alpha - \alpha_*)$;
            iv) $\lambda_{i+1}^j = \lambda_i^j$ for $j \neq k$
end
```

where n is the dimension of the state, e_k is a vector of dimension n with a 1 in the kth position and zeros everywhere else. $(M)_{i,j}$ is the (i,j) - th element of a matrix M, and α_* is the optimal acceptance rate as discussed in the introduction.

The algorithm's sample scheme is exactly the same as random scan. As we go on to find out, this approach is poorly suited to problems where the vectors in the default basis are not aligned with those we are interested in (e.g. eigenvectors of the target covariance matrix).

3.2.3 Global Adaptive Metropolis with Componentwise Adaptive Scaling

Here the sample step can move in any direction, and it does so informed by what it has learned about the scales in all the default directions.

Algorithm 6: Pseudocode detailing the Global Adaptive Metropolis with Componentwise Adaptive Scaling algorithm

```
Initialise $X_0$, $\mu_0$, $\Sigma_0$, and $\lambda_0^1$,...,$\lambda_0^n$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\mu_i$, $\Sigma_i$, and $\lambda_i^1$,...,$\lambda_i^n$;
    1. Sample $Z_{i+1} \sim N(0, \Lambda_i^{\frac{1}{2}} \Sigma_i \Lambda_i^{\frac{1}{2}})$ where $\Lambda_i = diag(\lambda_i^1, ..., \lambda_i^n)$;
    2. Set $X_{i+1} = X_i + Z_{i+1}$ with probability $\min(1, \frac{\pi(X_{i+1})}{\pi(X_i)})$ otherwise stay put.;
    3. for $k$ in 1,...,$n$ do
            $\mu_{i+1}^k = \mu_i^k + \gamma_{i+1}(X_{i+1} - \mu_i^k)$;
            $\Sigma_{i+1}^k = \Sigma_i^k + \gamma_{i+1}((X_{i+1} - \mu_i^k)(X_{i+1} - \mu_i^k)^T - \Sigma_i^k)$;
            $log(\lambda_{i+1}^k) = log(\lambda_i^k) + \gamma_{i+1}(min\{1, \frac{\pi(X_i + Z_{i+1}^T e_k)}{\pi(X_i)}\} - \alpha_*)$;
    end
end
```

Again, misalignment of the default directions with those of interest causes problems, although not as much as CAM-CAS since GAM-CAS always has a probability of sampling along directions of interest. Note that the sample step is equivalent to sampling a $Y_{i+1} \sim N(0, \Sigma_i)$ and transforming $Z_{i+1} = \Lambda_i Y_{i+1}$.

3.2.4 Adaptive Metropolis with Principal Components update

This algorithm uses the same sample and update scheme as CAS-CAM, but also adds a Principal Components update. In the following W is a matrix whose columns ($w(l)$ denotes the lth column) are what we estimate to be the first m eigenvectors of Σ_π the covariance matrix of the target distribution. These columns have corresponding eigenvalues $\rho(l)$, and l_i is a vector of scale factors.

Algorithm 7: Pseudocode detailing the Adaptive Metropolis with Principal Components update algorithm

```
Initialise $X_0$, $(l_0, \rho_0, W_0)$;
for $i$ in 1,...,$n$ do
    Given $X_i$, $(l_i, \rho_i, W_i)$;
    1. Sample an update direction $l \sim (d(1), ..., d(m))$;
    2. Sample $Z_{i+1} \sim N(0, l_i(l)\rho_i(l))$ and set $Y_{i+1} = X_{i+1} + Z_{i+1}w(l)$;
    3. Set $X_{i+1} = Y_{i+1}$ with probability $min\{1, \frac{\pi(Y_{i+1})}{\pi(X_i)}\}$, stay put otherwise;
    4. Update $(l_i, \rho_i, W_i)$ to $(l_{i+1}, \rho_{i+1}, W_{i+1})$ in light of $X_{i+1}$;
end
```

In effect, this is the same as the CAM-CAS/Random Scan algorithm, expect for the fact that the directions sampled in are updated in each iteration.

3.2.5 Localised Random Walk Metropolis

The purpose of this algorithm is to best replicate the target $\pi(x)$ using $\hat{q}_{\theta_i}(x)$, which is a mixture of normal distributions.

Algorithm 8: Pseudocode detailing the Localised Random Walk Metropolis algorithm

```
Initialise $X_0$, $\mu_0^{1:s}$, $\Sigma_0^{1:s}$, $\lambda_0^{1:s}$, and $w_0^{1:s}$.;
for $i$ in 1,...,$n$ do
    Given $X_i$, $\mu_i^{1:s}$, $\Sigma_i^{1:s}$, and $\lambda_i^{1:s}$;
    1. Sample $Z_{i+1} \sim \hat{q}_{\theta_i}(Z = k|X_i)$ and $Y_{i+1} \sim N(X_i, \lambda_i^{Z_{i+1}}\Sigma_i^{Z_{i+1}})$;
    2. Set $X_{i+1} = Y_{i+1}$ with probability $\alpha_k = min\{1, \frac{\pi(Y_{i+1})\hat{q}_{\theta_i}(k|Y_{i+1})}{\pi(X_i)\hat{q}_{\theta_i}(k|X_i)}\}$ otherwise stay
     put;
    3. for $k$ in 1,...,$s$ do
            $\mu_{i+1}^k = \mu_i^k + \gamma_{i+1}\hat{q}_{\theta_i}(Z_{i+1} = k|X_i)(X_{i+1} - \mu_i^k)$;
            $\Sigma_{i+1}^k = \Sigma_i^k + \gamma_{i+1}\hat{q}_{\theta_i}(Z_{i+1} = k|X_i)((X_{i+1} - \mu_i^k)(X_{i+1} - \mu_i^k)^T - \Sigma_i^k)$;
            $w_{i+1}^k = w_i^k + \gamma_{i+1}(\hat{q}_{\theta_i}(Z_{i+1} = k|X_i) - w_i^k)$;
            $log(\lambda_{i+1}^k) = log(\lambda_i^k) + \gamma_{i+1}\mathbb{I}\{Z_{i+1} = k\}(\alpha_k - \alpha_*)$;
    end
end
```

where

$$\hat{q}_{\theta_i}(Z_{i+1} = k|x) := \frac{w_i^k N(x; \mu_i^k, \Sigma_i^k)}{\hat{q}_{\theta_i}(x)}$$

and

$$\hat{q}_{\theta_i}(x) := \sum_{k=1}^{s} w_i^k N(x; \mu_i^k, \Sigma_i^k)$$

with $N(x; \mu, \Sigma)$ being the density of a normal distribution at x.

The algorithm decides which component to sample from using a discrete distribution conditional on the current position, whose weights are proportional those in the mixture $\hat{q}_{\theta_i}(x)$. Given this one might think that each component corresponds asymptotically to a (not necessarily connected) region in the state space. Given that these regions are selected only in probability we would expect their boundaries to be 'fuzzy'.

After sampling, the algorithm updates the mean and covariance for each component of the mixture, proportional to the corresponding probability mass used to select the component:

$$\hat{q}_{\theta_i}(Z_{i+1} = k|x) := \frac{w_i^k N(x; \mu_i^k, \Sigma_i^k)}{\hat{q}_{\theta_i}(x)}$$

Again, this is conditional on the current state, and hence the covariance and mean associated with the region in which the chain sits in that iteration will be updated by a greater amount.

The mixture weights are adjusted based on the new weights in the discrete distribution, and the scale of the selected component is updated to ensure an optimal acceptance rate is converged on.

3.3 Testing

Having implemented the algorithms, we tested them on various target distributions. The targets all took the form of $N(\mu_\pi, \Sigma_\pi)$ distributions in 2 dimensions; simple enough such that the basic characteristics of each adaptive scheme can be measured up, and visualised in their totality. Of course, the testing in this chapter is deficient in that it renders us ignorant as to the algorithms' performance in greater dimensions. However it suffices as a simple illustrative account, and we move to a higher dimensional setting in the following chapter.

3.3.1 The Setup

In each case we attempted to set up the mean and covariance of both the target and the initial proposal distribution to make the learning task sufficiently difficult for the adaptive scheme. The intention was to identify the specific weaknesses and strengths of each scheme, so that they could be better characterised and differentiated from the other schemes.

Specifically, for AM, CAM-CAS, GAM-CAS, and Local-RWM we used the following parameters in the target distribution:

$$\mu_\pi = (20, -20)^T \quad \Sigma_\pi = \frac{1}{4}\left(\begin{smallmatrix} 101 & 99 \\ 99 & 101 \end{smallmatrix}\right)$$

and for reasons that will become clear, we used the following parameters in the target for AM-PC

$$\mu_\pi = (20, -20)^T \quad \Sigma_\pi = \frac{1}{2}\left(\begin{smallmatrix} 100 & 0 \\ 0 & 1 \end{smallmatrix}\right)$$

In every, case apart from the Local-RWM, the parameters of the initial proposal are

$$\mu_0 = (-20, 20)^T \quad \Sigma_0 = \frac{1}{4}\left(\begin{smallmatrix} 101 & -99 \\ -99 & 101 \end{smallmatrix}\right)$$

In Local-RWM these are not initialised in general, but a set of parameters is initialised per component of the mixture. Contour plots of the distributions whose parameters are described above can be seen in figures 1.1-3

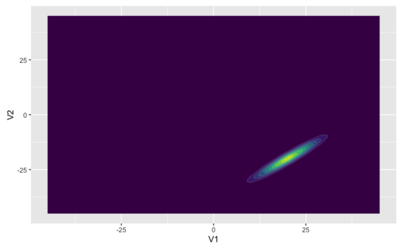

Figure 3.1: Target Distribution

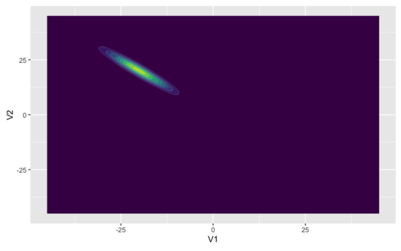

Figure 3.2: Initial Proposal Distribution

You'll notice that the semi-major axes of the ellipses which form the contours of the target are rotated $\frac{\pi}{2}$ away from those of the proposal. Hence the algorithms will need to adapt both the location and the orientation of the proposal, since it is beneficial for the proposal covariance to be proportional to the target.

Each chain was run for 10,000 iterations with $\gamma_i = 0.001$ for every iteration.

3.3.2 Adaptive Metropolis

The AM algorithm is the least specialised in that its basic function is copied and elaborated upon in all the other algorithms. Because of this, it is reasonable to assume that it's most sensible to use AM when relatively little is known about the target distribution, since if you did have any knowledge, albeit incomplete, you would use a more specialised adaptive scheme.

The trace plot (figure 3.4) indicates that the chain converges around iteration 5000, which is corroborated by the Gelman-Rubin diagnostic (figure 3.5). The evolution plot (figure 3.6) confirms the fact that the state moves quickly to the target, as shown in the trace plot. The green dot depicts the state, the blue depicts the proposal mean. The red contours are those of the proposal

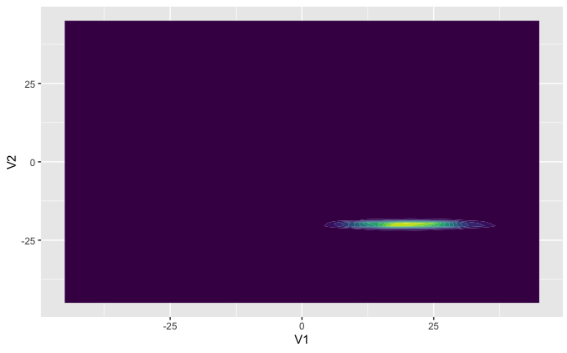

Figure 3.3: AM-PC Target Distribution

distribution, the blue are those of the target. We can now see that the proposal covariance and mean lags behind substantially. This is partly due to the specific values of $\{\gamma_i\}$ along with the fact the the proposal covariance and mean have to 'forget' all those states recorded on the chain's journey from the top left to the bottom right.

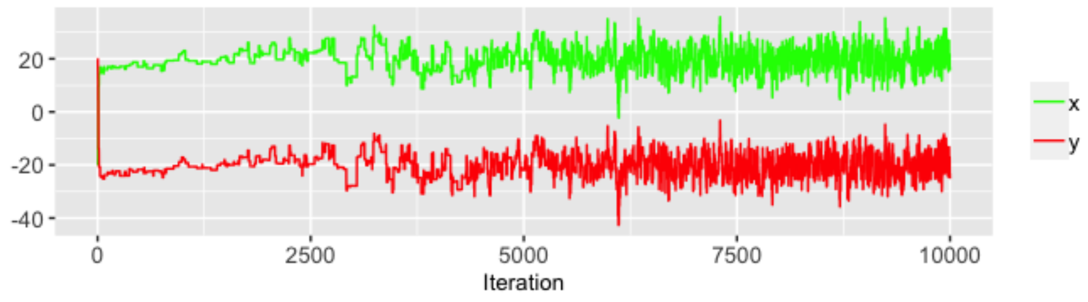

Figure 3.4: Traceplot for the Adaptive Metropolis

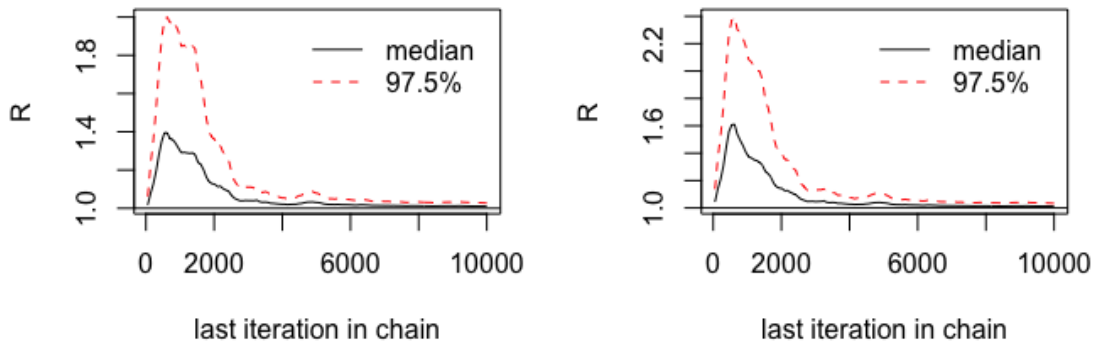

Figure 3.5: Gelman-Rubin diagnostic plots for Adaptive Metropolis run

With effective sample sizes (ESSs) of (233, 361) (one for each dimension), given convergence, the AM algorithm provides one of the better performances of the five. This is probably due to the fact that it is undifferentiated, or rather, that the other adaptive schemes are differentiated, but not to the target provided, since the target is Gaussian.

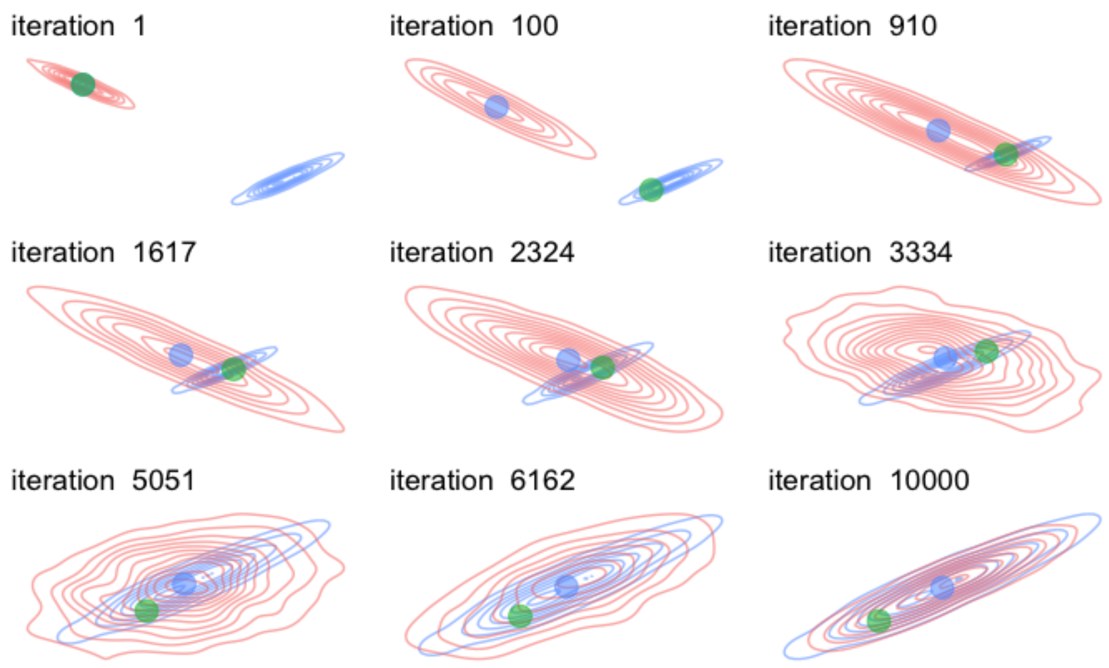

Figure 3.6: AM Evolution. The green dot depicts the state, the blue depicts the proposal mean. The red contours are those of the proposal distribution, the blue are those of the target.

3.3.3 Componentwise Adaptive Metropolis with Componentwise adaptive scaling

This adaptive scheme has the user provide a list of scales which are updated having explored along the default directions. The directions themselves are not updated, hence if the default directions are not aligned with the eigenvectors of the target covariance, the algorithm performs poorly. In our case, the default directions are $\{(0,1)^T, (1,0)^T\}$ and we set the eigenvectors of the target to be $\{(1,1)^T, (-1,1)^T\}$ hence, regardless of which direction we choose, they will always be at least $\pi/4$ away from the optimal. This is reflected in the algorithm's performance.

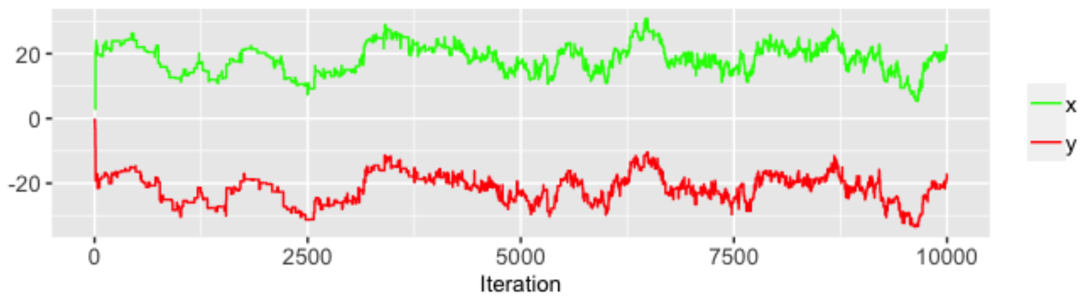

Figure 3.7: CAM-CAS Trace

Even though the scales are tuned on the fly to adjust to the optimal acceptance rate, the chain's mixing seems poor. We might understand heuristically this as resulting from the fact that, along the default directions, the chain needn't travel far until it reaches the 'edge' of the distribution at which point the proposals are mostly rejected, defining the 'edge' to be the contour outside of which only a small mass lies. Hence the Expected Squared Jump Distance (ESJD) is low ($O(10^{-5})$ where all other schemes produced ESJDs of $O(10^{-4})$), since the scales along the default directions are small and the proposals are often rejected. The resulting ESS is also poor: (20, 19).

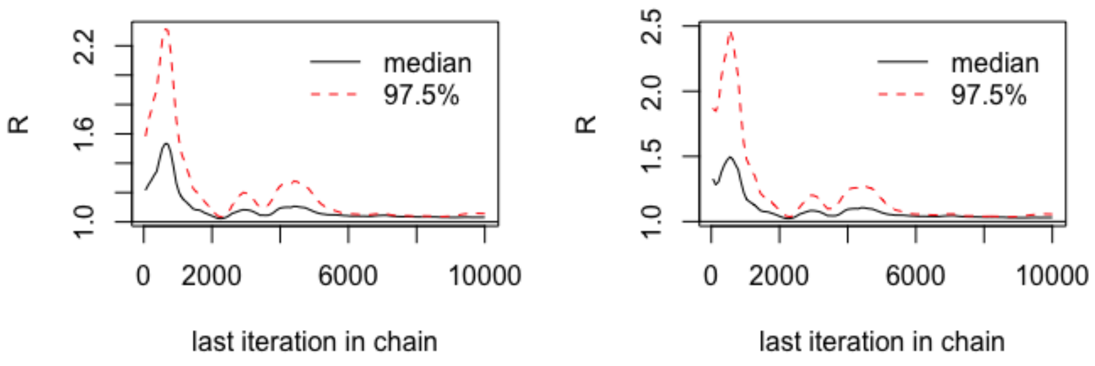

Figure 3.8: CAM-CAS Gelman Rubin

3.3.4 Global Adaptive Metropolis with Componentwise Adaptive Scaling

Instead of moving along a preset direction per iteration, here we use the information gained about the scales in all directions to move, and update accordingly. Interestingly the scales are updated *as though* the update is made in the direction along which the scaling applies. This sets the GAM-CAS update apart from the other algorithms in that it needs to generate a new set of acceptance probabilities, as opposed to just recycling those generated in the sample step, which comes with the associated computational cost.

As one might imagine, the scheme performs considerably better than CAM-CAS and about equally as well as AM in these circumstances (see figures 3.9-10) with an ESS of $(224, 228)$

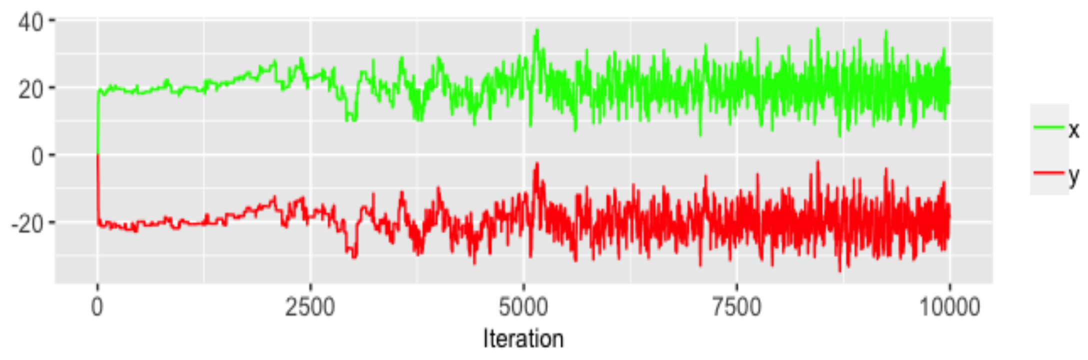

Figure 3.9: GAM-CAS Trace

3.3.5 Adaptive Metropolis with Principal components update

In this scenario, we composed our adaptive scheme by taking the sample and update scheme from CAS-CAM, and added on a way of updating the directions in which we sample (step 4. in the pseudocode), so that they are aligned with the eigenvectors of our best estimate of the covariance matrix. Seeing as though, had our initial proposal distribution been like the others, the eigenvectors would already be perfectly aligned, we decided to rotate the target by $\pi/4$. Here the green and blue dots, red and blue contours represent what they did before, with the addition that the eigenvectors are now represented by the lilac arrows. The arrows are rescaled so that they are clearly visible.

As we can tell from figure 3.11 the eigenvectors spend the first half of the chain out of alignment. This is due to the fact that the state has to travel from the top left corner of the state space to the bottom right, leaving a smear as it goes. This smear is interpreted by the algorithm as being the true distribution and hence the eigenvectors point in the direction they do. After the state spends

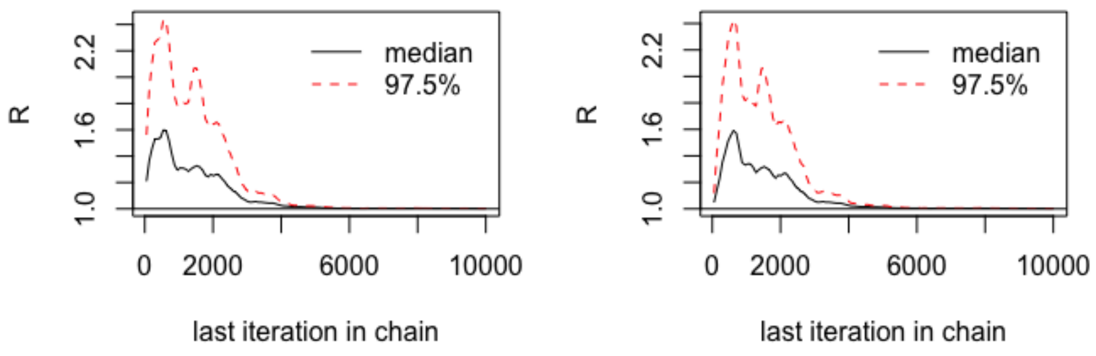

Figure 3.10: GAM-CAS Gelman Rubin

long enough in the locale of the true distribution, the adapted eigenvectors soon become aligned with the true eigenvectors.

The effective sample size was $(82, 445)$, which can be seen in the traceplot, as the y component converges much quicker than the x, taking less time to explore the extent of the distribution in that direction.

It's interesting to note which proposed eigenvector aligns with which target eigenvector. At iteration 100 it seems natural to assume that the eigenvector pointing sideways would be the one to align with the $(1, 0)^T$ target eigenvector and the one pointing down would be the one to align with $(0, 1)^T$, but further ahead in the run they switch. A further avenue of research would then be to examine which proposed eigenvector settles on which target eigenvector, depending on their initial position and alignment relative to the target.

Its easy to calculate the eigenvectors/values of a 2×2 matrix, so the algorithm is given a leg up due to having to learn a 2 dimensional target.

3.3.6 Local Random Walk Metropolis

In our implementation of Local RWM, we initialised $\hat{q}_{\theta_0}$ with four components of equal weight, distributed evenly in the locale of the target distribution. As the figures show, we found that the 'closest' component i.e. the one whose mean was closest to the target's quickly accumulated most of the weight, which makes sense as it would be the most sensible initial proposal distribution if one were to use any of the other adaptive schemes. By the end of the run, it had a weight > 0.999.

The evolution plot in figure 3.13 shows the 95% contour of the target in yellow along with the contour of each component in the mixture. The size of the points in each contour denotes the weight of that component.

We also tried a run in which all components were initialised with the same mean and covariance. In this case all the weights stayed close to 0.25, and the components moved very sluggishly, in unison towards the target, as though the algorithm couldn't decide which to use. In this case we were essentially running the AM algorithm at a quarter of the speed, and so the resulting run had a quarter of the efficiency.

This adaptive scheme is specialised to a family of target distributions which are very different from the one used here. Consider the scenario in which the scheme is fit to a heterogeneous distribution i.e. one in which the density in a given region of the state space has a character which does not resemble the density in the other regions. Given the fact that the scheme chooses which component of the mixture to sample from conditional on the current state, we might infer that in this scenario each component is associated with some region of the state space. The target distribution we actually fit the scheme to obeys basically the same regime in all regions since it is a simple 2D Gaussian.

This case illustrates a cautionary scenario in the evaluation of MCMC runs. As you can see from the evolution plot, the 'special' component of the mixture does converge to *something* but not the thing we would like it to converge to. This means that we achieve a good Gelman-Rubin, and ESS, and so had we not known the form of the target (which would be the case in a real setting)

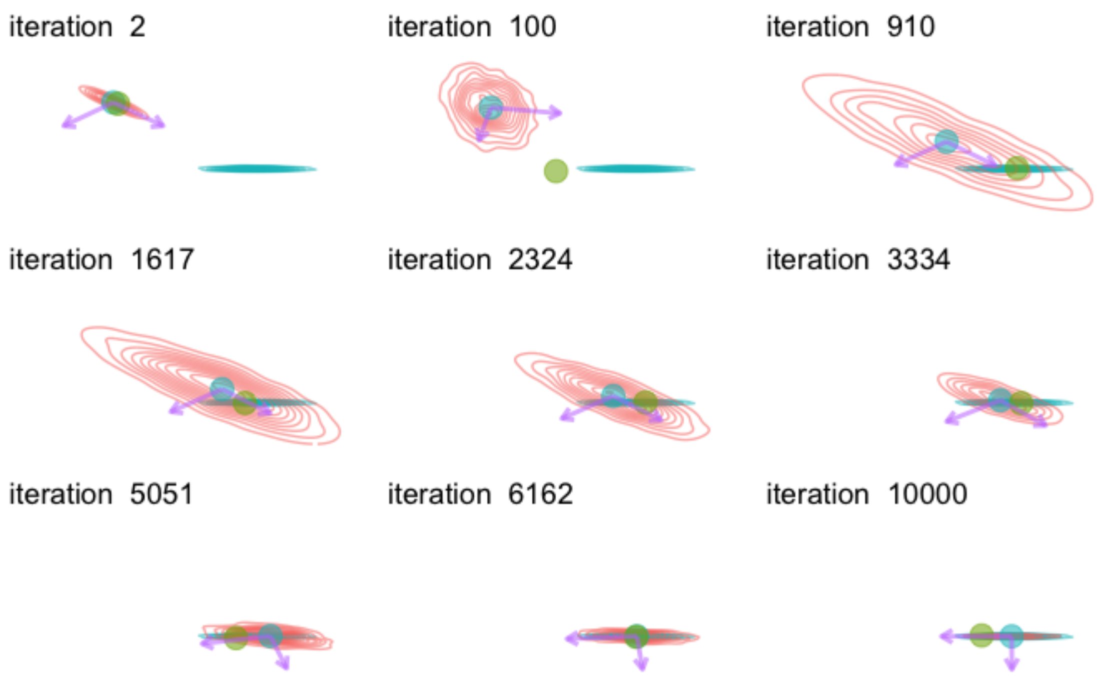

Figure 3.11: PC Evolution. The green dot depicts the state, the blue depicts the proposal mean. The red contours are those of the proposal distribution, the blue are those of the target. The eigenvectors are represented by the lilac arrows.

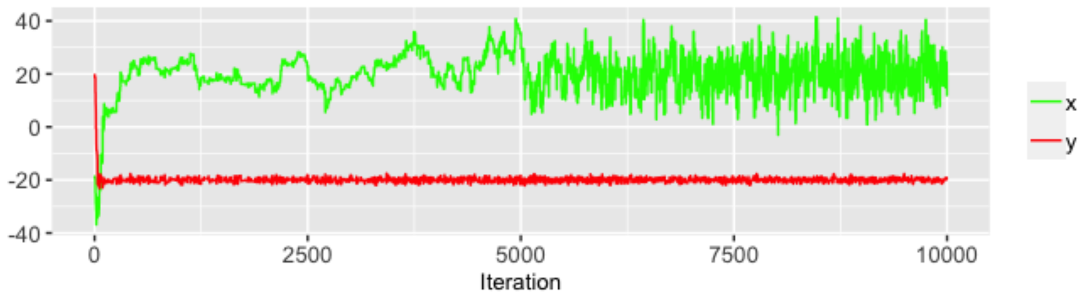

Figure 3.12: PCA Traceplot

we would think our algorithm performed well.

That being said, the way in which the Local RWM adaptive scheme behaves here offers an interesting development to the method of parallel chains. The method works as follows: pick a set of initial positions from an arbitrary distribution in the state space. Then set a MC running independently from each one. After the MC's have finished, the sample mean over all chains still obeys the MC SLLN. This method is useful because it avoids the autocorrelation of just having one long chain. One of its main drawback's is that, with a fixed amount of computing power, each chain may have to be relatively short. This is where Local RWM may come in useful.

We could start a Local RWM algorithm with the positions used in the parallel chains method, and then eliminate components from the mixture if their weight drops beneath a certain value, say $\frac{\epsilon}{r}$ with $\epsilon > 0$ and r is the number of components in the initial mixture, renormalising the remaining weights. This would then achieve a happy medium, in terms of use of computing power, between running one long chain and running parallel chains. The initial distributions would in effect act as 'scouts' for the final chain/chains, dropping out as they outgrow their use. Then, if we get to only one component, we could employ a more suitable adaptive scheme.

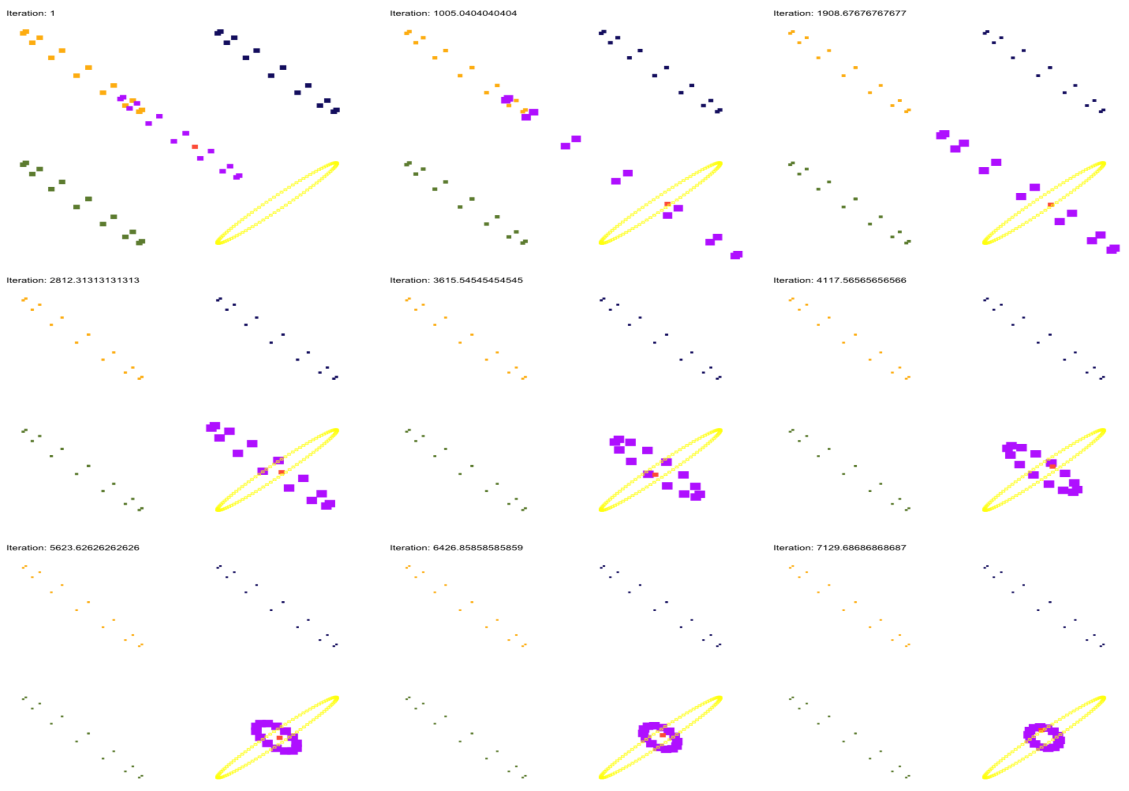

Figure 3.13: Local Random Walk Metropolis Evolution

3.4 Discussion

This final point raises some important questions as to the order in which we employ adaptive schemes: is it ever useful to run an MCMC algorithm with multiple schemes? If so, in what order? Clearly the evolution plot of Local RWM shows that even by the end of its run the covariance has not fully 'stretched' into the target, so it would be a good idea to discard the adaptive scheme and use one which is better at learning the general scale of things, such as GAM CAS. In the tutorial [10] the authors claim to use a combination of adaptive schemes in one test, and, whilst the exact implementation is unclear, it performs well as compared to using Adaptive Metropolis alone.

We can further refine our inquisition: should we run multiple updates from different adaptive schemes in the same step, choosing the most salient one by some decision procedure? Should this decision procedure be deterministic or stochastic? This may be, in fact, a natural extension of the Local RWM framework: instead of having a mixture of distributions with each component local to a region of the state space, we could employ a mixture of samplers and a mixture of adaptive updates to choose from at each step of the Markov chain. Or we could run parallel chains, each with different adaptive schemes, and whittle away those that are deemed unsuccessful using a metric like the Kullback Leibler divergence, for example. This is what an advanced practitioner might do by hand: instead of running the chains simultaneously, they might run them serially, each with a different adaptive scheme, like we have done in this chapter, inspecting each for their quality. Just like running preliminary tuning runs has been automated by adaptive MCMC, we could imagine automating the choice between adaptive schemes.

That brings us to the problem of programming time. Programming a procedure such as the above would be very cumbersome for the practitioner. It is no use extolling complex adaptive schemes if their implementation is beyond comprehension.

In order to gather up and compare the above adaptive schemes, we provide a table with various performance metrics.

Performance Metrics				
Scheme	ESS	ESJD / $\times 10^{-4}$	ESS per second	Duration/seconds
AM	$(233, 361)$	$(3.42, 3.38)$	$(48.5, 75.2)$	4.80
CAM CAS	$(20, 19)$	$(0.26, 0.27)$	$(14.9, 14.2)$	1.34
GAM CAS	$(224, 228)$	$(2.93, 2.87)$	$(36.7, 37.4)$	6.10
PCA	$(82, 445)$	$(6.11, 0.25)$	$(16.0, 86.62)$	5.14
Local RWM	$(423, 367)$	$(1.69, 1.73)$	$(16.3, 14.2)$	25.89

The table shows the AM scheme outclassing all others in every metric, apart from CAM CAS which is quicker in absolute terms. PCA is difficult to compare with the others, since the target distribution is re-oriented in terms of the dimensions with respect to which these metrics are measured. Had we time, we could use the same shape target covariance, but measure each algorithms performance at multiple different orientations i.e. multiple different angles at which the semi-major axis of the 95% contour ellipse subtends the x-axis, say. Then we could summarise the performance of each scheme over all angles, and provide an appropriate data visualisation.

In every scheme we implement, $\{\gamma_k\}$ simply returns a constant value of 0.001. Although the sequence is technically non-increasing, it is not decreasing either. This has the potential to violate the theoretical conditions described in the introduction under which Diminishing Adaptation attains, and so we can't be sure of ergodicity. This isn't as troubling as it first seems, for a couple of reasons. Firstly, the algorithms perform well regardless. Secondly, it may be the case that, *even though* our particular sequence $\{\gamma_k\}$ fails to satisfy the requirements, all we need is for there to exist a sequence that does, and for the update in adaptive parameters to be bounded above by it. For example, it may be the case that in

$$\theta_{i+1} = \theta_i + \gamma_{i+1} H_{i+1}(\theta_0, X_0, ..., Z_i, X_i, Z_{i+1}, X_{i+1})$$

$\{\gamma_k\}$ returns a constant, but due to the form of H you can prove that $\{\gamma_k H_k\}$ converges to 0 at a certain rate.

Another idea is to have $\{\gamma_k\}$ be constant for the time it takes $\gamma_{i+1} H_{i+1}(\theta_0, X_0, ..., Z_i, X_i, Z_{i+1}, X_{i+1}) = \theta_{i+1} - \theta_i$ to 'converge' (under some understanding of convergence) and then have it become some non-increasing sequence. In the context of the recursion

$$\theta_{i+1} = \theta_i + \gamma_{i+1} h(\theta_i) + \gamma_{i+1}\xi_{i+1}$$

we would be waiting for h to go to zero, and we are left with only the $\gamma_{i+1}\xi_{i+1}$ noise terms, at which point we would allow $\{\gamma_k\}$ to become non increasing.

This is motivated by the thought that if $\{\gamma_k\}$ converges to 0 too quickly, the adaptive parameters will not have enough of a chance to 'learn' the true parameters. An instructive example in the tutorial [10] is given where $\theta_{i+1} - \theta_i$ converges 'too quickly' to a bad value, preventing ergodicity.

As an illustration of what might motivate us to implement such a sequence we plot $\gamma_{i+1}(X_{i+1} - \mu_i) = \mu_{i+1} - \mu_i$ in the AM adaptive scheme (where $\gamma_i = 0.001$ for all i) (figure 2.13).

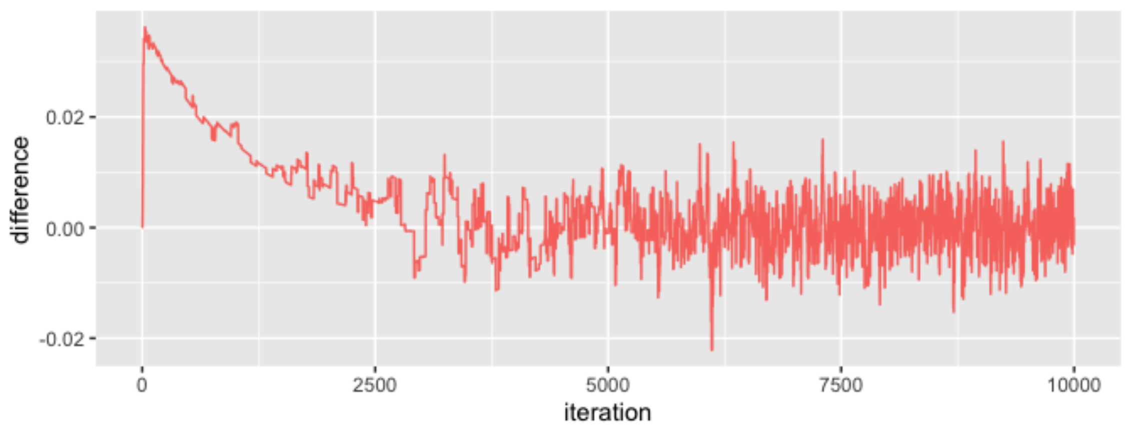

Figure 3.14: μ difference in AM

Somewhat unsurprisingly, the adaptive parameter converges at a similar iteration to the chain itself. We could then imagine transferring to a non-increasing $\{\gamma_k\}$ to eliminate the noise in the latter half of the plot. A natural extension to these ideas would be to implement a random variable sequence $\{\gamma_k\}$ that converges in some stochastic sense.

Chapter 4

Adaptive MCMC applied to Latent Position Network Models

4.1 Introduction

In this chapter we document an attempt to fit a Gaussian Latent Position Network Model (LPM) using Metropolis within Gibbs (MwG) with an adaptive scheme. We are motivated by the article 'Faster MCMC for Gaussian Latent Position Network Models' [12] which uses tailor made strategies per block update in the MC. Specifically they investigate a combination of Split Hamitonian Monte Carlo (Split HMC), and Firefly Monte Carlo to update the model's parameters.

To test their model, they both synthesise data and use a real world example. Seeing as though MwG is a popular way of fitting LPMs, they evaluate the performance of their algorithm by working out the ratio between their ESS/s and that of the MwG. This may seem foolhardy given that they haven't proved the ergodicity of their algorithm, a property which reliable calculation of the ESS relies on, however they claim that it 'remains an intuitive metric for the efficiency of a Markov chain approximation, due to its penalization of autocorrelations' and that recent work [13] suggests 'conservative confidence intervals based on the effective sample size can still be constructed even when geometric ergodicity does not necessarily hold'.

Several obstacles exist if the goal is to compare an adaptive strategy directly with the one implemented in the paper. The first is computer architecture. Given that they use 'wall time' i.e. the actual, in reality time it takes to complete the fit in their algorithms, as the denominator in their ESS/s calculation, comparisons are most useful if the machine the algorithms are run on is kept constant across the algorithms we wish to compare, which it was not. It is also the case that the code implementing the more computationally intensive parts of the algorithm described in the paper is written in `c++`, whereas all of our implementation is in R. However since their paper uses ratios to a baseline performance, we can use a similar strategy to achieve results which make sense to compare.

Regardless of these barriers to interpretability, we still felt that implementing adaptive schemes would be of interest.

4.2 Model Description

The model itself consists of a network of n nodes, where the ith node has a latent position $z_i \in \mathbb{R}^d$ where d is the dimension of the latent space. Associated with each dyad (pair of nodes) is a set of covariates. These can be used to describe characteristics of the relations between the nodes other than those modelled by the edges e.g. a social network where edges represent friendships and the covariates represent familial relationships. The presence or absence of the edges is modelled by draws of an independent Bernoulli variable, whose parameter is influenced by the latent positions of the nodes involved along with any covariates as described above, and other general parameters.

Stacking the latent position vectors on one another, we form a matrix $Z \in \mathbb{R}^{n\times d}$. We use the notation $A \in \mathbb{R}^{n\times n}$ to describe the adjacency matrix of the network. An adjacency matrix is a way of defining a network of n nodes, such that

$$(A)_{ij} := \begin{cases} 1, & \text{if node } i \text{ is connected to node } j \\ 0, & \text{otherwise} \end{cases}$$

where $(A)_{ij}$ is the (i, j)-th element of the matrix A. We define the probability of an edge between nodes i and j as

$$P(A_{ij} = 1|Z) = K(\|z_i - z_j\|, x_{ij})$$

where x_{ij} is the covariate for that edge. K thus acts as a link function. Each covariate encodes a categorical variable (e.g. $x_{ij} = 0$ if the edge is between close relations in our social network example, $x_{ij} = 1$ otherwise) in C categories. We use the adjacency matrix, and the covariates for each edge as the data for the model. That K is non-increasing in its first argument engenders structure in the latent space i.e. that nodes closer in the latent space are more likely to be connected. Specifically, the authors of the paper choose the following:

$$K(\|z_i - z_j\|, x_{ij}) = \tau_{x_{ij}} \exp(-\frac{1}{2\gamma^2}\|z_i - z_j\|^2)$$

where $\tau \in [0, 1]^C$ and $\gamma^2 > 0$. Here $\tau_{x_{ij}}$ is the x_{ij}th component of τ. The latent positions, and the parameters, are the quantities we would like to infer, so that we can better understand the network. That the x_{ij} are categorical now makes more sense since they act as an index over the components of the vector τ. The form of K characterises the model as Gaussian. We can interpret τ as a measure of the overall sparsity of the network, and γ^2 as influencing how quickly the link probability decays with distance in the latent space. We chose to sample from the distributions of Z, τ, and γ^2, and so we need to formulate their prior distributions.

In accordance with the paper, we used a $N(0, \Omega^{-1})$ prior for each of the d columns of Z, where $\Omega \in \mathbb{R}^{n\times n}$ is the precision matrix of a given column. This gives the following posterior:

$$P(Z|A, \tau, \gamma^2) \propto \prod_{(i,j)\in E_A} K(\|z_i - z_j\|) \prod_{(i,j)\notin E_A} (1 - K(\|z_i - z_j\|)) \exp(-\frac{1}{2}\sum_{j=1}^{d} Z_{.j}^T \Omega Z_{.j})$$

where E_A is the set of connected dyads (i.e. (i, j) is in E_A if nodes i and j are connected). We assigned an Inverse Gamma prior to γ^2 and an independent Beta prior to each component of τ.

4.3 Sampling strategy

Our sampling strategy can be roughly summarised as consisting of a MwG algorithm, run variously with and without adaptive schemes.

4.3.1 Baseline MwG

Since the authors kindly published their codebase we were able translate it directly from `c++` and modify it for our own purposes. Their baseline MwG works as follows (we call it 'baseline' since we build adaptive strategies atop it, and since we use its performance as a baseline to compare against).

Sampling γ^2

Given an InverseGamma(a, b) prior on γ^2, the posterior density we achieve has no closed form Gibbs update. However, as detailed in [12], if we parametrise $Z^* = \gamma^{-1}Z$ the posterior distribution of γ^2 becomes

$$P(\gamma^2|A, Z, \tau) = \text{InverseGamma}(\gamma^2|a + \frac{nd}{2}, b + \sum_{j=1}^{d}(Z^*_{.j})^T\Omega^{-1}(Z^*_{.j}))$$

allowing it to be sampled directly in a Gibbs update using out-of-the-box statistical software.

Sampling τ

Assigning a Beta prior with parameters (α_k, β_k) to the kth component of τ gives the following posterior:

$$P(\tau_k|A, Z, x_{ij}, \alpha_k, \beta_k) \propto \tau_k^{\alpha_k+\zeta_k^1-1}(1-\tau_k)^{\beta_k-1} \prod_{x_{ij}=k, A_{ij}=0} (1-\tau_{x_{ij}} exp(-\frac{1}{2}\|z_i - z_j\|))$$

where ζ_k^1 is defined as

$$\zeta_k^1 = |\{(i,j) \in [n]^2 : A_{ij} = 1, x_{ij} = k\}|$$

which is the size of the set of dyads of nodes connected by edges such that their associated covariate is k.

As recommended, each component of τ is updated individually using a uniform proposal centred on its current value; being accepted or rejected on the basis of a Metropolis-Hastings step.

Sampling Z

Sampling Z is a slightly more complex affair. At iteration k we sample d independent, n-dimensional vectors from $N(0, \Omega^{-1})$ distributions, stack them next to each other, and pre-multiply by a fixed constant to achieve a matrix $Y^k \in \mathbf{R}^{n\times d}$. Then, for each row z_i^k of Z^k, propose $z_i^k + y_i^k$ and accept based on a Metropolis-Hastings step using the following density:

$$P(z_i|A, \tau, \gamma^2) \propto \exp\left(-\sum_{k=1}^{d} Z_{.k}^T \Omega^{-1} Z_{.k}\right) \prod_{j:(i,j)\in E_A} K(\|z_i - z_j\|, \tau, \gamma^2) \prod_{j:(i,j)\notin E_A} (1-K(\|z_i - z_j\|, \tau, \gamma^2)$$

4.3.2 Adaptive Schemes

We built the following adaptive schemes on top of the baseline MwG strategy:

- Adaptive Metropolis (AM)
- Adaptive Metropolis with Global Adaptive Scaling (AM GAS)
- Global Adaptive Metropolis with Componentwise Adaptive Scaling (GAM CAS)

These schemes were applied only to the updating of the Z matrix. In updating τ we simply tuned the global scale of the proposal distribution described above so as to ensure its acceptance rate evolved to be as close as possible to 0.44. This was done in concert with each of the above adaptive schemes. Since sampling γ^2 involves a simple Gibbs update, there was no need to adapt.

Given that the updated Z is proposed on a column-by-column basis, and accepted on a row by row basis, the adaptive schemes had to be transformed from their form in chapter 3.

Adaptive Metropolis

Instead of sampling the columns of Y from $N(0, \Omega^{-1})$ distributions, instead, at iteration k, we sampled from $N(0, \Sigma_k)$ where Σ_k is the empirical covariance of all of the previously updated columns. We needn't update the mean of the proposal, since it can be proved that the posterior expectation of each latent position is 0 [14]. This is due to the fact that the latent positions affect the edge probabilities only through the distances between them, which are invariant under rotation and translation.

Adaptive Metropolis with Global Adaptive Scaling

Here we built upon the Adaptive Metropolis strategy by introducing an n-dimensional vector of scales: one for every node. After constructing our Y matrix at iteration k, instead of proposing $z_i^k + y_i^k$ we proposed $z_i^k + \lambda_i^k y_i^k$ where λ_i^k is the ith component of the scales vector. We then updated each scale as follows:

$$\log(\lambda_i^{k+1}) = \log(\lambda_i^k) + \gamma_k(\alpha - \alpha_*)$$

where α is the acceptance probability in the MH update, α_* is the optimal acceptance 0.234, and $\{\gamma_k\}$ is as described in the introduction.

Global Adaptive Metropolis with Componentwise Adaptive Scaling

This was the most complicated of our adaptive schemes. Again, we used Adaptive Metropolis to update the covariance matrices of the columns of Z. We then applied the following scheme:

Algorithm 9: Pseudocode detailing the Global Adaptive Metropolis with Componentwise Adaptive Scaling algorithm, as applied to the Gaussian LPM

```
Initialise a scales matrix $S^0 \in \mathbf{R}^{n\times d}$ At iteration $k+1$ given $Z^K$, $S^k$, and $\Sigma_1^k$,...,$\Sigma_d^k$ (the
  empirical covariances matrices for each column of $Z$, calculated using Adaptive
  Metropolis) ;
1) Calculate $N^k \in \mathbf{R}^{n\times d}$ such that the jth column follows
  $N_{.j}^k \sim N(0, \mathrm{diag}(S_{.j}^k)^{\frac{1}{2}}\Sigma_j^k \mathrm{diag}(S_{.j}^k)^{\frac{1}{2}})$.;
2) for $i$ in 1,...,$n$ do
    2)i) In updating the ith row of Z, propose $z_i^k + n_i^k$ where $n_i^k$ is the ith row of $N^k$ and
      accept in a M-H step then;
    2)ii) for $j$ in 1,...,$d$ do
        2)ii)a) Calculate $\alpha_{ij}$ which is the M-H acceptance probability of $z_i^k + (N^k)_{ij}e_j$
          where $e_j$ is the d-dimensional vector with zeros everywhere except a one in the jth
          position;
        2)ii)b) Update the scales matrix using
            $\log((S^{k+1})_{ij}) = \log((S^k)_{ij}) + \gamma_k(\alpha_{ij} - \alpha_*)$
    end
end
```

Here $\mathrm{diag}(v)$ is the square matrix with V as its diagonal and zeros elsewhere, $\mathrm{diag}(v)^{\frac{1}{2}}$ is that same matrix, but with the square root of the diagonal elements. $M_{.j}$ is the jth column of the matrix M

Easy Gains on GAM CAS

In part 2)ii)a) of the above algorithm the acceptance probability is calculated by evaluating the log-density of the original vector z_i^k and subtracting it from what we evaluate to be the log density of the componentwise updated vector $z_i^k + (N^k)_{is}e_s$. We provide s as an index, to avoid confusion in the following mathematical derivation. Given that the log-density takes the form of a large sum, we found that most of the components of one log-density cancel the components of the other, or they at least combine to form a more easily computable expression. This presented us with an opportunity to make easy gains in computational time, which we took. The results are shown alongside the others below.

Mathematically, we have that the log-density of the Metropolis-Hastings acceptance ratio for updating the ith vector in the sth position is

$$\mathrm{MH}_i = \mathrm{ld}(z_{i\ \mathrm{new}}) - \mathrm{ld}(z_i)$$

where

$$z_{i\ \mathrm{new}} = z_i + \delta e_s$$

We've defined $(N^k)_{is} := \delta$ for notational efficiency. Here

$$\mathrm{ld}(z_i) = -\sum_{j:(i,j)\in E_A} (M)_{ij} z_i^T z_j + \frac{1}{2}(M)_{ii}\|z_i\|^2 + \sum_{j:(i,j)\notin E_A} \log(1 - \tau_{x_{ij}} \exp(-\frac{1}{2}\|z_i - z_j\|^2))$$

where $M = L_A + \gamma^{-2}\Omega^{-1}$ is an $n \times n$ real matrix. L_A is the combinatorial Laplacian matrix of the network and it is defined as $L_A = D^A - A$ where D_A is the diagonal matrix of node degrees.

To understand where this matrix comes from consists of a formal mathematical argument which we include in the appendix. The presence of γ in M comes from the transformation we make to sample γ^2. From this definition of ld() we get that

$$\text{MH}_i = -\sum_{j:(i,j)\in E_A} (M)_{ij}(z_{i\text{ new}}^T z_j - z_i^T z_j) + \frac{1}{2}(M)_{ii}(\|z_{i\text{ new}}\|^2 - \|z_i\|^2) + \sum_{j:(i,j)\notin E_A} \log(R)$$

where we define the fraction

$$R := \frac{1-\tau_{x_{ij}}\exp(-\frac{1}{2}\|z_{i\text{ new}} - z_j\|^2)}{1-\tau_{x_{ij}}\exp(-\frac{1}{2}\|z_i - z_j\|^2)}$$

We can therefore make the following set of simplifications:

$$\begin{aligned}(z_{i\text{ new}}^T z_j - z_i^T z_j) &= (z_{i\text{ new}} - z_i)^T z_j \\ &= \delta e_s z_j \\ &= \delta (Z)_{js}\end{aligned} \tag{4.1}$$

and

$$\|z_{i\text{ new}}\|^2 - \|z_i\|^2 = 2\delta(Z)_{is} + \delta^2$$

We can also write

$$\log(R) = \log\left(\frac{1-\tau_{x_{ij}}\exp(-\frac{1}{2}(2\delta((Z)_{is}-(Z)_{js})+\delta^2))\exp(-\frac{1}{2}\|z_i - z_j\|^2)}{1-\tau_{x_{ij}}\exp(-\frac{1}{2}\|z_i - z_j\|^2)}\right)$$

because

$$\|z_{i\text{ new}} - z_j\|^2 = \|z_i - z_j\|^2 + 2\delta((Z)_{is} - (Z)_{js}) + \delta^2$$

and so finally we get

$$\text{MH}_i = -\sum_{j:(i,j)\in E_A} (M)_{ij}\delta(Z)_{js} + \frac{1}{2}(M)_{ii}(2\delta(Z)_{is} + \delta^2) + \sum_{j:(i,j)\notin E_A} \log(R)$$

where R is in its updated form, as above.

Initialisation

As in the paper, we initialised as follows:

- In every case, we restricted τ to being one dimensional i.e. a scalar. Hence $\tau_k = \tau \in \mathbf{R}$ which we initialised at its true value, and the hyperparameters in its prior $(\alpha_k, \beta_k) = (\alpha, \beta)$ were initialised at $(1, 1)$.
- As with τ, σ^2 was initialised at its true value. Its hyperparameters (a, b) were initialised at $(1, 1)$
- The prior precision $\Omega^{-1} \in \mathbf{R}^n$ of each column of Z was initialised as an n-dimensional identity matrix, and first Z sample is the posterior MLE.
- The true Z is created using the above prior precision, and the adjacency matrix A is created using the link probability function, along with the true Z, τ, and σ^2.
- The latent space is 2 dimensional in all cases.

4.4 Metrics and Results

Given those caveats mentioned in the introduction, it would be foolish to evaluate our algorithms and compare results directly with those used in the paper. Instead, as in the paper, we established baseline results for each of the RWM and MwG algorithms and measure the factor by which they improve upon implementation of the adaptive schemes, thus giving a metric agnostic to the machine the code runs on allowing us to compare our factors by those in the paper.

As in the paper, the metric by which we judge our algorithms is the Effective Sample Size per second (ESS/s) of the log-probabilities of edges in the network. For a network of n nodes there are $O(n^2)$ dyads, hence we choose a uniformly random subsample of 500 potential edges, whose log-probability we inspect for each algorithm. It is important that the set of dyads is held constant across algorithms, to ensure a reliable comparison.

Having calculated the ESS/s we measured the factor, per dyad, by which the ESS/s increased from the baseline MwG case. Note that each algorithm builds upon the code of the baseline MwG, hence they will be guaranteed to use more floating point operations, increasing the amount of time until completion. So we are interested to find out whether the ESS can more than make up for the losses in time.

Plotted below are the median factors for each algorithm, for each combination of parameters τ and σ^2. Each algorithm was run for 1000 iterations.

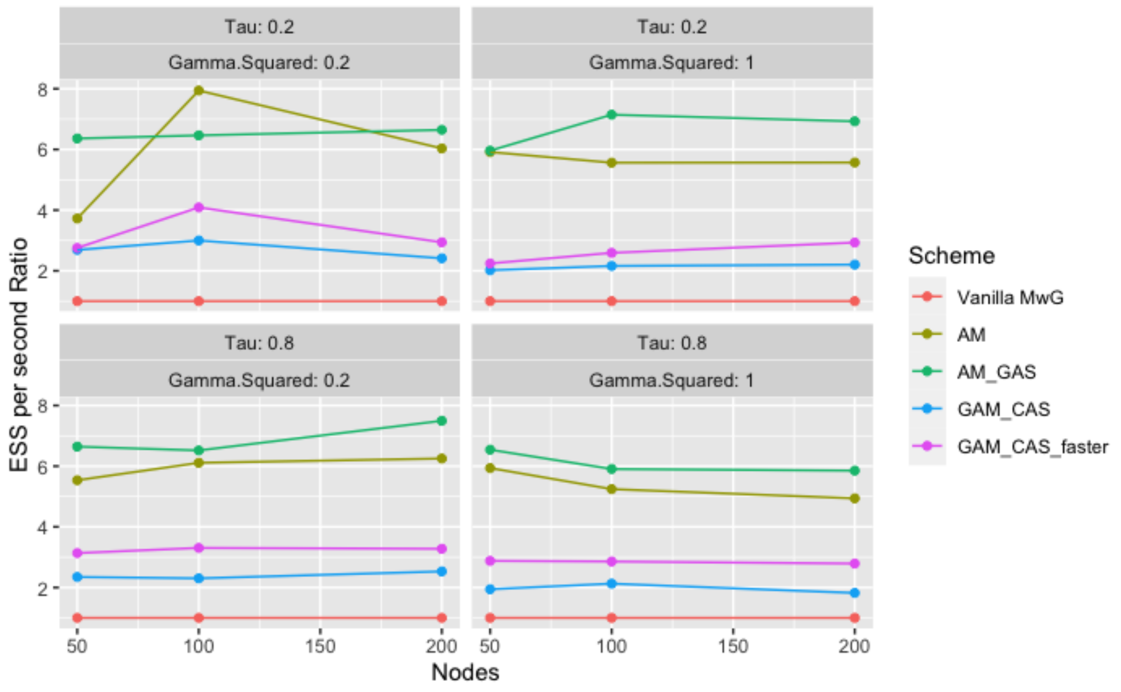

Figure 4.1: Adaptive results at 1000 iterations

Figure 3.1 shows that in each case the additional computation cost was more than compensated by an increase in ESS, although it also betrays the diminished gains that are made by the highly specialised GAM CAS scheme. Even with the optimisation the scheme performs poorly when compared to its simpler counterparts.

The results also show that each scheme is more or less agnostic to the number of nodes, and the combinations of parameters, suggesting that the model has not yet reached a level of size, complexity, or specialisation after which the adaptation becomes too computationally expensive, or unspecialised.

We can compare our results with those achieved by Spencer, Junker and Sweet. The algorithms they used were as follows:

- Split HMC
  - Split HMC [15] is a variant of Hamiltonian Monte Carlo that splits the gaussian components from the other components in the posterior, updating separately over these in the integrator step of the HMC.
- No U Turn Sampler (NUTS)
  - A NUTS [16] version of HMC was implemented which adaptively tuned the integration time. The algorithm uses Hamiltonian dynamics along with the constraint that the path sampled along doesn't turn back on itself within a short period.

- Metropolis within Gibbs
- Elliptical Slice - Elliptical Slice samplers[17] work by exploiting posteriors with gaussian priors. They take hyper-elliptical slices from these prior densities, sampling on these ellipses, and accept or reject based on the likelihood.
- Elliptical Slice within Gibbs
- Stan
  - The authors included the Stan [18] implementation of HMC so that they could compare the algorithms conceived in the paper against one which was 'out-of-the-box'.

Each of these algorithms were implemented with and without Firefly [19] sampling of the non-edges in the network, with the exception of Stan. The firefly sampling scheme is embedded within the HMC. It involves a dummy variable, whose state dictates whether a non-edge in question is to be sampled. Seeing as though the non-edge computations are the bottleneck in the HMC, implementing Firefly, as explained in the paper, is desirable in sparse networks.

The split HMC algorithm implemented in the paper outperforms all else, getting increasingly competitive compared against the baseline as the number of nodes increases. The ratio between the split HMC and the baseline is at or under 10 for all combinations of parameter values, in all node numbers under 500. However, when applied to a network with 500 nodes, the ratio increases slightly such that in all cases it resides just above 10, with an exception being for $(\tau, \gamma^2) = (0.8, 0.2)$ where the split HMC algorithm performs 100 times better. The story is the same as when they implement firefly sampling, although in no cases does the algorithm perform 100 times better than the baseline.

In their case, as in ours, some of the algorithms suffer from the curse of excess computation, namely the elliptical slice, elliptical slice within Gibbs, and Stan underperforming even the baseline MwG.

4.4.1 Exploring Computational Bottlenecks

Whilst our results are in the same order of magnitude as the paper's, we felt it necessary to test at least one adaptive algorithm to a higher number of iterations. This would allow a proper amount of time for the adaptive parameters to be learnt. However, there remained the problem of computing time. We had decided to implement using only 1000 iterations because the time in which the algorithms ran in numbers higher were infeasible on the author's personal computer. Not to be fazed, we set about using the `profvis` function in R on the baseline MwG algorithm.

Identifying Bottlenecks

R provides a package called `profvis` containing a function of the same name which allowed us to submit as an argument a piece of code we wished to optimise. The function then runs the code and takes samples at random points in time to inspect which functions are being called. It then takes the user to an R environment in which multiple interactive windows containing the results are shown. In these windows the user can see both a timeline of all the functions used, showing their duration and position. Also shown is a table of functions, each with the amount of time spent inside them in seconds. This allowed us to identify exactly which functions in the code were the computational bottlenecks.

Rectifying Bottlenecks

The major bottleneck we identified was calculating the log densities of the rows of Z. So, as in the GAM CAS case, we set about calculating the log-density of a row by hand, seeing whether there were any cancellations or simplifications. Fortunately there were, resulting in a minor decrease in runtime. Undeterred, we decided to use the `Rcpp` package to write the difference in log densities in `c++`. Thankfully this achieved a four-fold decrease in the runtime, allowing us to increase the number of iterations to 5000. We chose to implement the updated `c++` code in the AM GAS scheme as it had been the most successful in our preliminary runs. The optimisation was applied equally to the baseline, and the adaptive scheme, to ensure fairness of comparison. The results

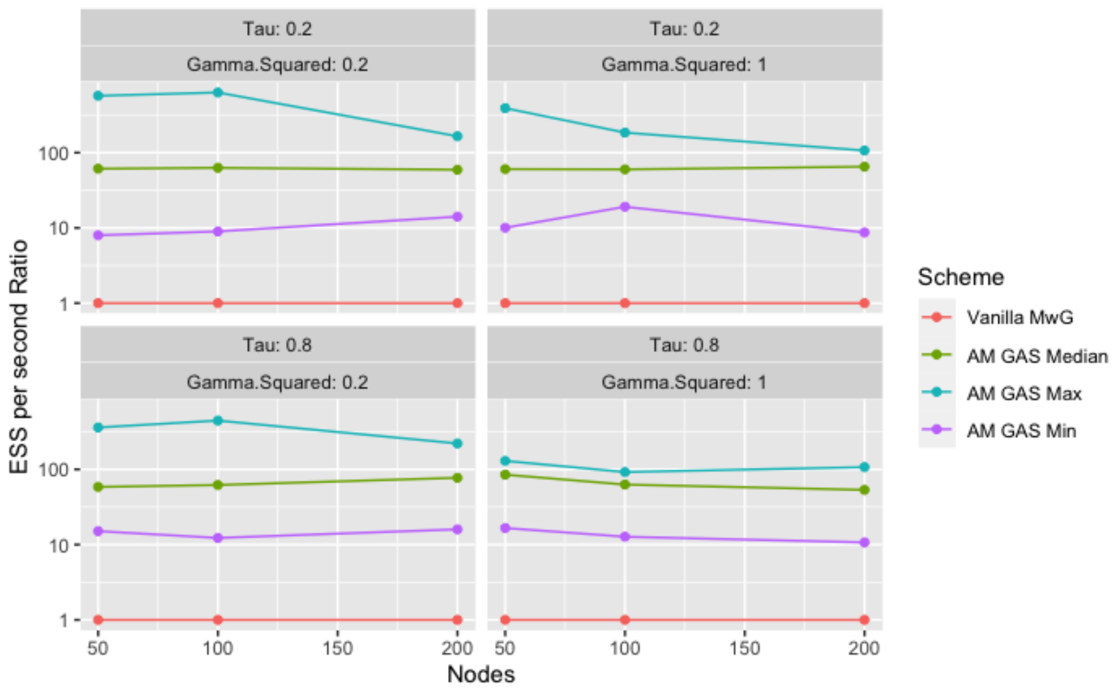

Figure 4.2: AM GAS results at 5000 iterations as compared to the baseline MwG. These results are after optimisation to both algorithms.

can be seen in figure 3.4. Note the logarithmic scale this time, and the addition of the maximum and minimum values.

Again the performance is agnostic to parameter combination and node number. It remains consistently very high, and would no doubt increase if given more iterations. The fact that the minima never dip below the baseline indicate that it is never not a good idea to implement a simple adaptive scheme, so long one expects the time to program is sufficiently low. It is our contention that the performance would eventually peak at a maximum value, as it varies with iteration number. To investigate how the performance changed with computing time, we ran a similar algorithm at 50 nodes with $(\tau, \gamma^2) = (0.8, 1.0)$, stopping every 500 iterations and measuring the ESS. Recalling the introduction, the ESS is proportional to n i.e. the number of iterations, and so, seeing as though we expect the adaptive algorithm to produce a higher ESS/s, the further it gets in its run, we might reasonably expect the ESS to increase on average, and to betray some notion of how well the parameters are being learned at each stage in its run. Figure 4.3 shows that this is true, although the maxima, minima, and other quantiles are not strictly increasing. This is due to the fact that increasing the number of iterations at which you measure the ESS, gives more lags at which the correlation can contribute.

We can, in fact, easily construct cases where the ESS is initially comparatively good compared to the total number of iterations, but fares terribly the further along the chain we inspect it. Take, for example, a successful MCMC run $\{X_1, ..., X_k\}$ (such as, say, the one depicted in figure 2.4). Measuring the ESS at the end of such a run gives a reasonable value as compared to the length of the chain. Then suppose we continue along the chain, to find that it is of the form $\{X_1, ..., X_k, X_1, ..., X_k, X_1, ...\}$ i.e. it is comprised of a concatenation of copies of the original chain. This introduces large correlations at all lags which are multiples of the period $k + 1$ and hence the ESS as compared to the full chain length gets considerably worse.

Figure 4.4 shows the ESS over time, calculated using only the previous 500 samples of the chain. This provides an account of the ‘samples’ gained through time, without worrying about how samples far into the past of the chain influence the correlations in the infinite sum calculated in the denominator of the ESS. However it is more ‘noisy’ because there are less samples to estimate the correlation with, per calculation of the ESS.

Figure 4.7 shows a histogram of our ESSs for an example run of the AM GAS adaptive scheme. It depicts two main groupings. To fully understand the evolution of the adaptive algorithm, we selected a point from each grouping and plotted the evolution of their ESSs, along with the

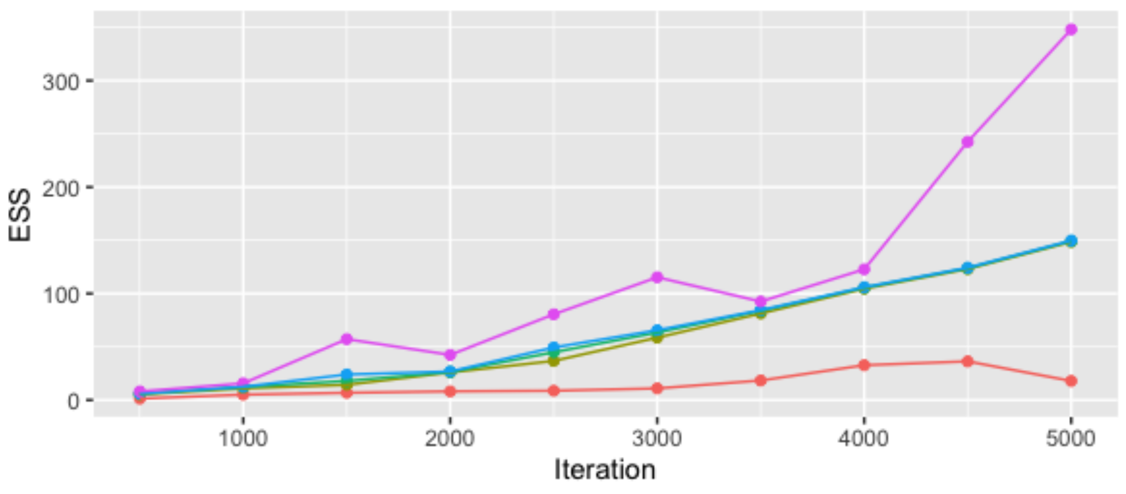

Figure 4.3: AM GAS results over time. After every 500 iterations the algorithm takes the entire length of chain so far, and calculates the ESSs of each dyad. Each line on the graph depicts a quantile in $\{0.0, 0.25, 0.5, 0.75, 1.0\}$

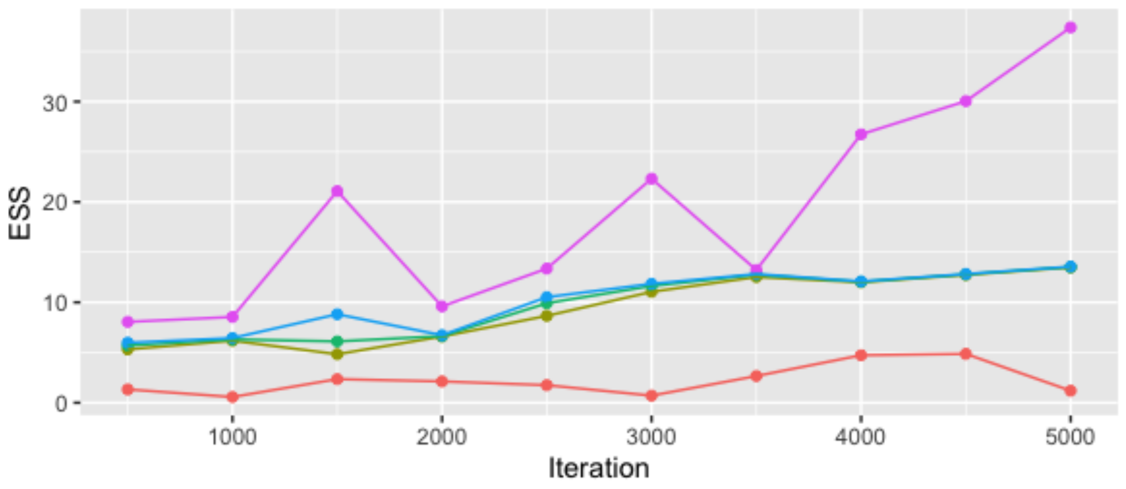

Figure 4.4: AM GAS results over time, calculated in chunks of 500 iterations. Each line on the graph depicts a quantile in $\{0.0, 0.25, 0.5, 0.75, 1.0\}$

associated adaptive parameters over time in figure 4.5. Since each dyad consists of two nodes, and the latent position of each node has an associated global scale for its proposal distribution, the adaptive parameters are the two scales for each dyad (here depicted as scalenode1 and 2), with the dyad in the low ESS grouping on the top row, and the dyad in the high ESS grouping in the bottom row. The main thing that this visualisation tells us is that the adaptive parameters are nowhere near maturing, giving weight to the notion that the adaptive schemes would improve the output even further given a longer iteration time.

4.5 Discussion

In all of the above results we have used the same metric of success as the paper. Whilst this is useful in providing an immediate insight into performances, to provide a comprehensive review of the algorithms would need providing an ensemble of metrics, as discussed in the introduction, using their absolute values and not their ratios to a baseline. The fact that we were able to achieve an almost 100-fold increase to MwG is initially impressive, but less so if you look at what we recorded to be the ESSs. In figure 4.6 we plot a histogram of the 500 ESSs of the log-probabilities for a 5000 iteration run of MwG at 50 nodes, with $(\tau, \gamma^2) = (0.8, 1.0)$.

The histogram shows that, from 5000 iterations, the Metropolis within Gibbs mostly only

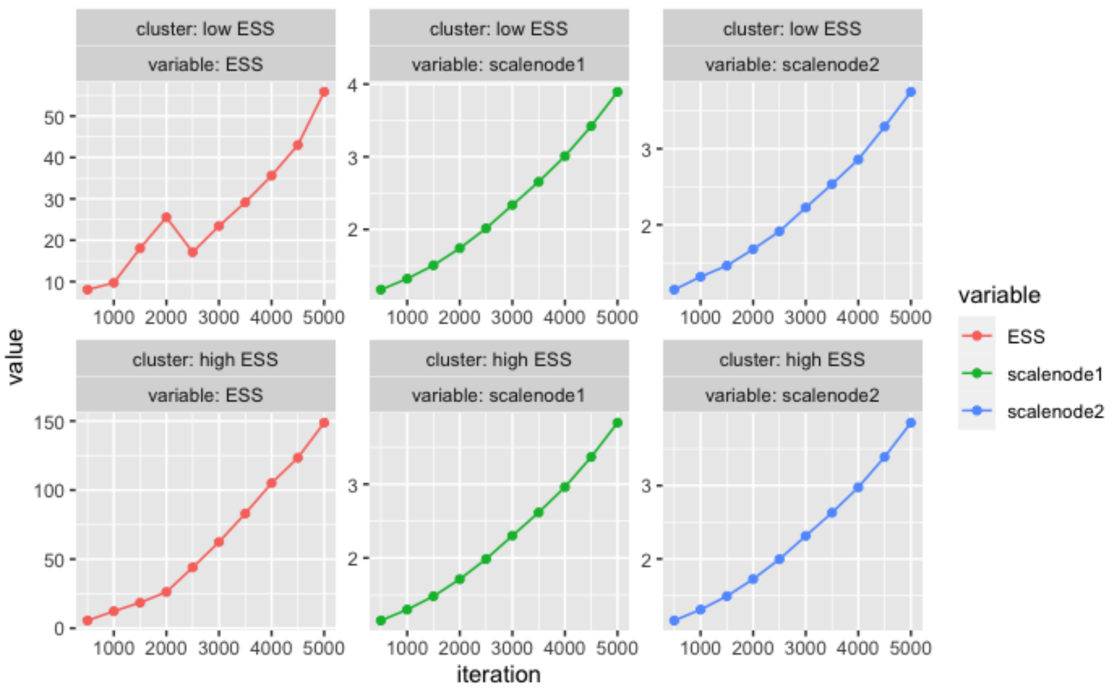

Figure 4.5: ESS of the Log-Probabilities of two dyads over time. The rightmost four graphs depict the four adaptive parameters: two per dyad, so one for each node in the dyad. They are the global scales of the proposal distribution for the latent position of the nodes.

effectively yields under 2.5 samples, which is very poor. So, whilst our results surpass the paper's in all parameter and node combinations but one, they are less impressive viewed absolutely. We show the histogram of our ESSs with the same parameter values in figure 4.7 to compare.

Another obfuscation in comparing comes from the fact that the paper's authors only start measuring 'wall time' after tuning. The whole conceit of adaptive MCMC is that tuning is achieved during the chain, so the comparison would be more illuminating if the authors of the paper began measuring the wall time from the beginning of the tuning period. There is evidence in their codebase that they record both the time to tune and the time to run, but because they don't present the information in the paper we are still in the dark. The success of the adaptive schemes comes with a small caveat. Even though the ESS/s is high compared to the baseline MwG, they still run in considerably longer times, preventing a practitioner whose absolute time is restricted. That being said, the algorithms were run on the author's personal computer, so access to a powerful enough computational cluster eliminates this problem.

The other issue is with programmer time. Even if after translation into `c++` the algorithms ran in a quarter of the time, the gain in overall efficiency is negligible in the eyes of a practitioner who has spent too long learning a new programming language. That being said, it took the author less than a day to learn and implement the relevant code. With this in mind, it is instructive to note that the results are presented with code common to both the baseline and the adaptive algorithm translated into `c++`. We could make further gains to our performance by translating the code that handles the update step of the adaptation, code which sometimes turns out to be very computationally intensive.

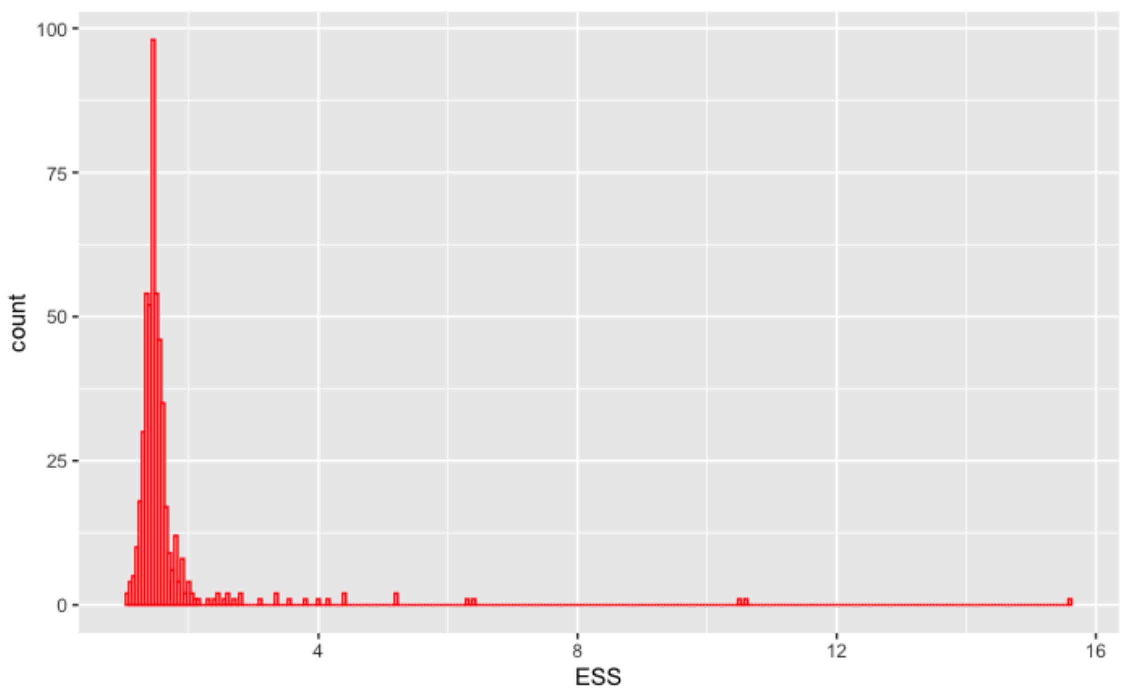

Figure 4.6: ESS histogram for 5000 iterations at 50 nodes with $(\tau, \gamma^2) = (0.8, 1.0)$ under Metropolis within Gibbs

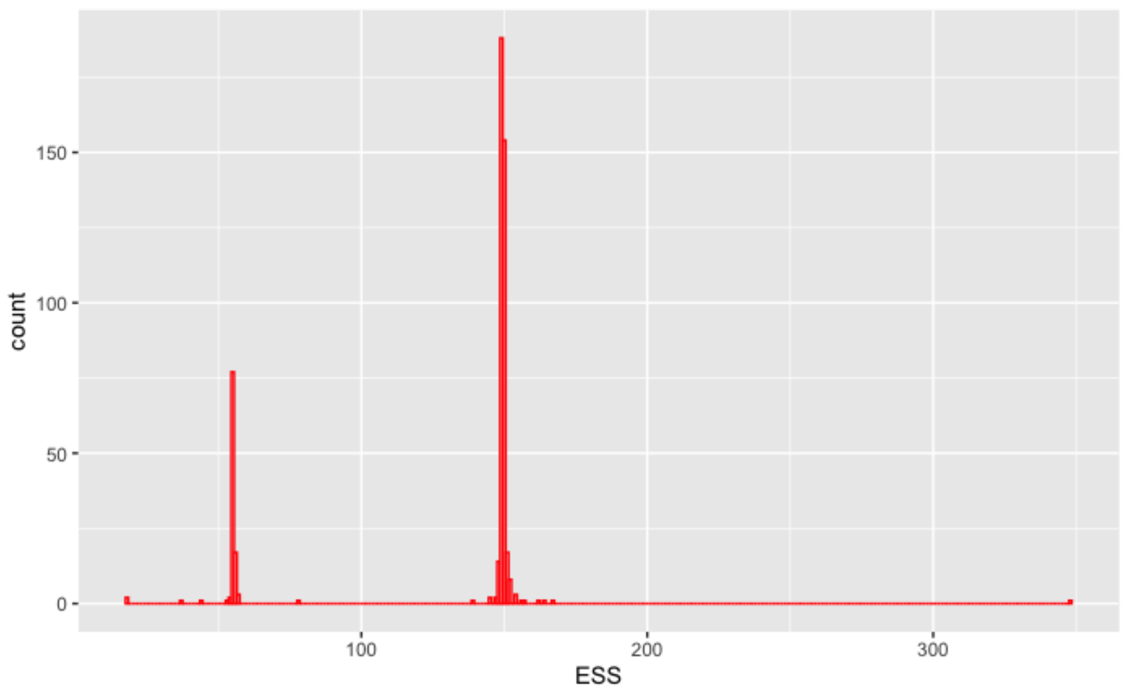

Figure 4.7: ESS histogram for 5000 iterations at 50 nodes with $(\tau, \gamma^2) = (0.8, 1.0)$ under the AM GAS adaptive scheme

Chapter 5

Discussion

5.1 Overview

In this report we provide a theoretical introduction, and a short foray into adaptive Markov Chain Monte Carlo methods. In the first chapter we gave a compact theoretical introduction to Markov Chains and their asymptotic theory, with the motivation to provide an estimate of an intractable expectation. We also introduced the conditions under which these theorems hold. We then gave the basic framework for a generic MCMC algorithm, after which we introduced concepts by which we could measure the efficiency of its output.

In the second chapter we gave a simple motivating example for the notion of adaptive MCMC, moving to describe a similar generic framework. We carefully outlined the conditions under which moving to an adaptive frame leaves the ergodicity of the chain unharmed. Under the guidance of [10] we recast the adaptation in the Robbins-Monro update scheme, both as an explanatory tool and as a way to easily identify when the conditions for ergodicity are satisfied.

In the third chapter we set about introducing, implementing, and testing five adaptive schemes. They ranged from the basic to more exotic, and each stimulated discussion as to their performance and applicability. We went on to discuss their natural extensions.

The penultimate chapter is motivated by [12] to introduce the Gaussian Latent Position Network Model. We describe the model, and the strategies by which we sample under the baseline Metropolis within Gibbs, which we compare against to measure the success of our adaptive strategies. We then describe the adaptive schemes, as modified from the prior chapter, identifying an easy optimisation that serves to lessen the computational load of the most complex scheme. After detailing the state in which the algorithms are initialised, we go on to describe the metric by which we compare them. We describe our results, analysing the strengths and failings of the schemes, and compare to those in [12]. We identify that the algorithms cut short before the adaptive parameters learn properly. This motivates us to identify and optimise computational bottlenecks, going so far as to translate our code from `R` into `c++`. This grants us a significant increase in performance, on both our prior results, and those in [12]. We then explore the performance as it evolves with number of iterations, and discuss the validity of the success metrics.

5.2 Limitations and Further Work

There is much in this report we would like to develop. For instance we realise that the proof of reversibility in the first chapter is conditional on the MCMC algorithm moving to a new state, where we would have liked to develop a proof for the full, unconditional Markov chain kernel. We would also like to explore mathematically the connection between the measures of success described: identifying the statistical distributions of their estimates, and relating them in a coherent, mathematical manner. For example, given a particular value of the estimated ESJD, what can we infer about the ESS? We could also introduce more recent measures such as the Generalised Speed Measure [20], and relate them back to those discussed in this report.

In chapter two we encounter the Robbins-Monro update scheme

$$\theta_{i+1} = \theta_i + \gamma_{i+1} H_{i+1}(\theta_0, X_0, ..., Z_i, X_i, Z_{i+1}, X_{i+1})$$

such that our choice of $\{H_k\}$ converges to zero when we achieve optimality of some kind. We would like to prove this for some example sequences, and some notions of optimality. To illustrate this, it would be instructive to prove that $\lim_{i\to\infty} \mu_i$ is the target mean under the scheme

$$\mu_{i+1} = \mu_i + \frac{1}{i+1}(X_{i+1} - \mu_i)$$

using the MC SLLN or the MC CLT, and assumptions about Diminishing Adaptation, the principal condition for ergodicity we introduced in chapter two. We could also explore stopping conditions, and implement them on our toy examples, investigating how their formulation affects the actual performance of the algorithm. For example, since we expect $\{H_k\}$ to converge, we could stop once the difference between terms is less than some $\epsilon > 0$, which we would tune. Since the algorithm is stochastic, we may want to stop after this condition occurs for a preset number of consecutive terms in the sequence.

There are many natural extensions to the work in chapter three. For instance, we could at least substantially increase the dimensionality of the target, to more accurately portray the models faced by modern practitioners. The form of the adaptive schemes also warrant the inclusion of a non-gaussian target. For instance the formulation of the Local-RWM scheme lends itself to investigating its performance on a multimodal distribution. We could borrow from the literature and use something like the 'banana' distribution [10], whose form is dictated by transforming a normally distributed $X \sim N(0, \Sigma)$ for $\Sigma \in \mathbb{R}^{n\times n}, n > 1$ as such:

$$Y = (X_1, X_2 + b(X_1^2 - 100), X_3, ..., X_n)$$

A promising line of research may be to explore both in the space of adaptive parameters and adaptive schemes as the algorithm runs. Formulation of this problem would require substantial work in itself. Having laid the theoretical groundwork for the problem, we could then attempt to implement the meta-algorithm as an off-the-shelf piece of software, motivated by the ease of use this would provide practitioners. We note the similar work done in [21], where the authors introduce an optimisation procedure that learns over the space of models per stage in the clinical prognostic pipeline i.e. it learns over data-imputation models, then it learns over feature processing models etc.

We could then introduce the notion of an adaptive agenda, whereby different adaptive schemes are phased in and out at different stages of the algorithm. We may, for instance, want to use a more general adaptive scheme at the beginning of the MCMC run, such as Adaptive Metropolis, to quickly learn the general form of the distribution, and then use more specialised schemes, the more we think we know about the target.

The natural extension to the work in chapter four would be to run the adaptive models for a much higher number of iterations. On the computing architecture the author currently uses, this would probably necessitate further optimisation. This would probably take the form of further mathematical reasoning, like that which is explained in the chapter, and then implementing into `c++`. This would allow the adaptive schemes to learn more fully, and we would get a better picture of the evolution of the adaptive parameters.

Appendix A

A.1 Mathematical Proofs

Theorem 1: Reversibility implies Invariance

Let p be the density of a MC that is π-reversible. Then p is π-invariant.

Proof

For all $A \subseteq \chi$, if $X_n \sim \pi$

$$\begin{aligned}
P(X_{n+1} \in A) &= \int_{x\in\chi}\int_{y\in A} \pi(x)p(x,y)dxdy \\
&= \int_{x\in\chi}\int_{y\in A} \pi(y)p(y,x)dxdy \qquad \text{(reversibility)} \\
&= \int_{y\in A} \pi(y)\left(\int_{x\in\chi} p(y,x)dx\right)dy \\
&= \int_{y\in A} \pi(y)dy \\
&= \pi(A)
\end{aligned} \tag{A.1}$$

□

Theorem 2: Existence of an Upper Bound for the Distance between Ideal Expectation, and Expectation under the MC

If the sequence of parameter values $\{\theta_k\}$ is such that there is a deterministic sequence $\{\gamma_k\}$ for which $|\theta_{i+1} - \theta_i| \leq \gamma_i$ with $\gamma_i \to 0$ as $i \to \infty$, we can provide an upper bound for

$$|\hat{E}_*(f(X_i)) - E_\pi(f(X))|$$

Proof

The conceit to the proof is to establish a parallel process to the adaptive one, but which stops adapting at a certain point $k_i < i$. Then X_{k_i} is the state when the halt occurs and we denote $P^{i-k_i}_{\theta_{k_i}} f(X_{k_i})$ as the state of the process after $i - k_i$ steps under zero adaptation, conditional on $\theta_0, X_0, X_1, ..., X_{k_i}$. This allows us to make the decomposition

$$\hat{E}_*(f(X_i)) - E_\pi(f(X)) = \hat{E}_*(P^{i-k_i}_{\theta_{k_i}} f(X_{k_i}) - E_\pi(f(X))) + \hat{E}_*(f(X_i) - P^{i-k_i}_{\theta_{k_i}} f(X_{k_i}))$$

Which, with the triangle inequality, implies

$$|\hat{E}_*(f(X_i)) - E_\pi(f(X))| \leq |\hat{E}_*(P^{i-k_i}_{\theta_{k_i}} f(X_{k_i}) - E_\pi(f(X)))| + |\hat{E}_*(f(X_i) - P^{i-k_i}_{\theta_{k_i}} f(X_{k_i}))|$$

Now, since the adaptation has halted, we would expect the first term on the right hand side to converge to zero by the Markov Chain CLT, so long as the non-adaptive chain is ergodic to π. We will assume that it is, although the tutorial [10] examines the instances of non-ergodicity carefully. Another thing to note is that the tutorial assumes $|f| \leq 1$, and so do we [1]. So we would like to provide an upper bound for the second term on the right hand side of the above inequality. We have that

$$\begin{aligned}
\hat{E}_*(f(X_i) - P_{\theta_{k_i}}^{i-k_i} f(X_{k_i})) &= \sum_{j=k_i}^{i-1} \hat{E}_*(P_{\theta_{k_i}}^{i-(j+1)} f(X_{j+1})) - \hat{E}_*(P_{\theta_{k_i}}^{i-j} f(X_j)) \\
&= \sum_{j=k_i}^{i-1} \hat{E}_*(P_{\theta_j} P_{\theta_{k_i}}^{i-(j+1)} f(X_j)) - \hat{E}_*(P_{\theta_{k_i}}^{i-j} f(X_j)) \\
&= \sum_{j=k_i}^{i-1} \hat{E}_*((P_{\theta_j} - P_{\theta_{k_i}}) P_{\theta_{k_i}}^{i-j-1} f(X_j)) \\
\implies |\hat{E}_*(f(X_i) - P_{\theta_{k_i}}^{i-k_i} f(X_{k_i}))| &\leq \sum_{j=k_i}^{i-1} |\hat{E}_*((P_{\theta_j} - P_{\theta_{k_i}}) P_{\theta_{k_i}}^{i-j-1} f(X_j))|
\end{aligned} \tag{A.2}$$

where the implication is due again to the triangle inequality. Note that the summand in the final equality can be written

$$\begin{aligned}
\hat{E}_*((P_{\theta_j} - P_{\theta_{k_i}}) P_{\theta_{k_i}}^{i-j-1} f(X_j)) &= \sum_{k=j}^{k_i+1} \hat{E}_*(P_{\theta_k} P_{\theta_{k_i}}^{i-j-1} f(X_j)) - \hat{E}_*(P_{\theta_{k-1}} P_{\theta_{k_i}}^{i-j-1} f(X_j)) \\
&= \sum_{k=k_i+1}^{j} \hat{E}_*(P_{\theta_k} P_{\theta_{k_i}}^{i-j-1} f(X_j)) - \hat{E}_*(P_{\theta_{k-1}} P_{\theta_{k_i}}^{i-j-1} f(X_j))
\end{aligned} \tag{A.3}$$

where the order of the sum is reversed in the final equality. Yet another application of the triangle inequality then yields

$$|\hat{E}_*(f(X_i) - P_{\theta_{k_i}}^{i-k_i} f(X_{k_i}))| \leq \sum_{j=k_i}^{i-1} \sum_{k=k_i+1}^{j} |\hat{E}_*((P_{\theta_k} - P_{\theta_{k-1}}) P_{\theta_{k_i}}^{i-j-1} f(X_j))| \tag{A.4}$$

We now propose the notation

$$\Delta_{\gamma_k} = \Delta(\gamma_k) = \sup_{|g| \leq 1} \sup_{x \in \chi, \{\theta, \theta' \in \Theta^2 : |\theta - \theta'| \leq \gamma_k\}} |P_\theta g(x) - P_{\theta'} g(x)|$$

Now if we let $P_{\theta_{k_i}}^{i-j-1} f(X_j) = g(x)$ we reach the conclusion

$$|\hat{E}_*(f(X_i) - P_{\theta_{k_i}}^{i-k_i} f(X_{k_i}))| \leq \sum_{j=k_i}^{i-1} \sum_{k=k_i+1}^{j} \Delta_{\gamma_k}$$

relying on the fact that

$$|\hat{E}_*((P_{\theta_k} - P_{\theta_{k-1}}) P_{\theta_{k_i}}^{i-j-1} f(X_j))| \leq \Delta_{\gamma_k}$$

□

Theorem 3: The Laplacian

In chapter 4, the log-density of the posterior of a row of Z is calculated using the Laplacian matrix L^A. It is defined

$$L^A = D^A - A$$

[1] this seems sensible, since the Roberts and Rosenthal (2005)[22] result assumes bounded f, so you can redefine $f_{new} := f/sup(f)$

where A is an $n \times n$ adjacency matrix, D^A is a diagonal matrix containing the degrees of each node i.e. $D^A_{ii} = \sum_{j=1}^n A_{ij}$ where A_{ij} is the (i,j)th element of A. The paper [12] equates the two following posteriors:

$$P(Z|A,\tau,\gamma^2) \propto \prod_{(i,j)\in E_A} \tau \exp\left(-\frac{\|z_i - z_j\|^2}{2\gamma^2}\right) \prod_{(i,j)\notin E_A} \left(1 - \exp\left(-\frac{\|z_i - z_j\|^2}{2\gamma^2}\right)\right) \exp\left(-\frac{1}{2}\sum_{l=1}^d Z_{.l}^T \Omega Z_{.l}\right)$$

and

$$P(Z|A,\tau,\gamma^2) \propto \prod_{(i,j)\in E_A} \tau \exp\left(-\frac{1}{2}\sum_{l=1}^d Z_{.l}^T (\Omega + \frac{1}{\gamma^2}L^A) Z_{.l}\right) \prod_{(i,j)\notin E_A} \left(1 - \exp\left(-\frac{\|z_i - z_j\|^2}{2\gamma^2}\right)\right)$$

So we need to prove

$$\sum_{(i,j)\in E_A} \|z_i - z_j\|^2 = \sum_{l=1}^d Z_{.l}^T L^A Z_{.l}$$

Proof

Since $D^A_{ii} = \sum_{j=1}^n A_{ij}$ we have that

$$\begin{aligned} Z_{.l}^T D^A Z_{.l} &= \sum_{i=1}^n D^A_{ii} Z_{il}^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n A_{ij} Z_{il}^2 \end{aligned} \tag{A.5}$$

which makes the right hand side of the equation we wish to prove into

$$\begin{aligned} \sum_{l=1}^d Z_{.l}^T L^A Z_{.l} &= \sum_{l=1}^d \left(\sum_{i=1}^n \sum_{j=1}^n A_{ij} Z_{il}^2 - Z_{il} A_{ij} Z_{jl}\right) \\ &= \sum_{l=1}^d \sum_{i=1}^n \sum_{j=1}^n A_{ij} Z_{il} (Z_{il} - Z_{jl}) \end{aligned} \tag{A.6}$$

Since E_A is the set of all edge dyads we have that $(i,j) \in E_A \iff A_{ij} \neq 0$, so we can rewrite:

$$\sum_{(i,j)\in E_A} = \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n A_{ij}$$

and reformulate the left hand side of the equation into

$$\begin{aligned} \sum_{(i,j)\in E_A} \|z_i - z_j\|^2 &= \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n A_{ij} \|z_i - z_j\|^2 \\ &= \frac{1}{2}\sum_{l=1}^d \sum_{i=1}^n \sum_{j=1}^n A_{ij} (Z_{il} - Z_{jl})^2 \end{aligned} \tag{A.7}$$

After some algebraic manipulation, the statement we're trying to prove becomes

$$\sum_{l=1}^d \sum_{i=1}^n \sum_{j=1}^n A_{ij} Z_{il}^2 = \sum_{l=1}^d \sum_{i=1}^n \sum_{j=1}^n A_{ij} Z_{jl}^2$$

and so swapping i and j in one of the sides gives us the desired result, since A is symmetric.

□

A.2 Code

What follows is the result of translating from R into C plus plus, a function which calculates a difference in log densities.

```
#include <RcppArmadillo.h>
using namespace Rcpp;

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
double ldzi_nofirefly_update_cpp(int i, arma::mat Alabels, arma::mat z, arma::vec epsilon,
arma::mat connecteds, int connectednum, arma::mat disconnecteds,
int disconnectednum, arma::vec tau, arma::mat gaussmat){
  // make the adjustment into 0 based indices
  i -= 1;
  double res = 0;
  double square = dot(z.row(i), z.row(i));
  for(int j = 0; j < connectednum; ++j){
    res -= gaussmat(i, connecteds(i, j) - 1) * dot(epsilon, z.row(connecteds(i, j) - 1));
  }
  res += 0.5 * gaussmat(i, i) * (2 * dot(epsilon, z.row(i)) + dot(epsilon, epsilon));

  // no edge dyads
  for(int j = 0; j < disconnectednum; ++j){
    int d_ij = disconnecteds(i, j) - 1;
    // again, you have to make the adjustment, since disconnecteds stores indices in R
    // format not in c++
    double normal_exponent = exp(- 0.5 * (square + dot(z.row(d_ij), z.row(d_ij))
    - 2 * dot(z.row(d_ij), z.row(i))));
    // double correction = exp(- 0.5 * (2 * dot(epsilon, (z.row(i) - z.row(d_ij)))
    + dot(epsilon, epsilon)));
    double normal_exp_with_correction = exp(- 0.5 * (square + dot(z.row(d_ij), z.row(d_ij))
    - 2 * dot(z.row(d_ij), z.row(i)) + 2 * dot(epsilon, (z.row(i) - z.row(d_ij)))
    + dot(epsilon, epsilon)));

    int tau_curr = tau[0];

    res += log((1 - tau_curr * normal_exp_with_correction) / (1 - tau_curr * normal_exponent))
  }
  return(res);
```

Bibliography

[1] Ruslan Salakhutdinov. “Learning Deep Boltzmann Machines using adaptive MCMC”. In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning* (July 2010), pp. 943–950.

[2] Boby Mathew et al. “Bayesian adaptive Markov chain Monte Carlo estimation of genetic parameters”. In: *Heredity* 109 (July 2012), pp. 235–45. DOI: `10.1038/hdy.2012.35`.

[3] Gareth O. Roberts and Jeffrey S. Rosenthal. “General state space Markov chains and MCMC algorithms”. In: *Probab. Surveys* 1 (2004), pp. 20–71. DOI: `10.1214/154957804100000024`. URL: `https://doi.org/10.1214/154957804100000024`.

[4] Nicholas Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: 21.6 (June 1953), pp. 1087–1092. DOI: `10.1063/1.1699114`.

[5] “Monte Carlo Sampling Methods using Markov Chains and their Applications”. In: *Biometrika* 57.1 (Apr. 1970), pp. 97–109. DOI: `10.1093/biomet/57.1.97`.

[6] Andrew Gelman and Donald B. Rubin. “Inference from Iterative Simulation Using Multiple Sequences”. In: *Statist. Sci.* 7.4 (Nov. 1992), pp. 457–472. DOI: `10.1214/ss/1177011136`. URL: `https://doi.org/10.1214/ss/1177011136`.

[7] G. O. Roberts, A. Gelman, and W. R. Gilks. “Weak convergence and optimal scaling of random walk Metropolis algorithms”. In: *Ann. Appl. Probab.* 7.1 (Feb. 1997), pp. 110–120. DOI: `10.1214/aoap/1034625254`. URL: `https://doi.org/10.1214/aoap/1034625254`.

[8] Gareth O. Roberts and Jeffrey S. Rosenthal. “Optimal scaling for various Metropolis-Hastings algorithms”. In: *Statist. Sci.* 16.4 (Nov. 2001), pp. 351–367. DOI: `10.1214/ss/1015346320`. URL: `https://doi.org/10.1214/ss/1015346320`.

[9] Jeffrey S. Rosenthal Gareth O. Roberts. “Coupling and Ergodicity of Adaptive Markov Chain Monte Carlo Algorithms”. In: *Journal of Applied Probability* 44.2 (2007), pp. 458–475. DOI: `https://www.jstor.org/stable/27595854`.

[10] Johannes Thoms Christophe Adrieu. “A Tutorial on Adaptive MCMC”. In: *Stat Comput* 322.18 (2008), pp. 343–373. DOI: `10.1007/s11222-008-9110-y`.

[11] Heikki Haario, Eero Saksman, and Johanna Tamminen. “An adaptive Metropolis algorithm”. In: *Bernoulli* 7.2 (Apr. 2001), pp. 223–242. URL: `https://projecteuclid.org:443/euclid.bj/1080222083`.

[12] Neil A. Spencer, Brian Junker, and Tracy M. Sweet. “Faster MCMC for Gaussian Latent Position Network Models”. In: (2020). arXiv: `2006.07687 [stat.CO]`.

[13] Jeffrey S. Rosenthal. “Simple confidence intervals for MCMC without CLTs”. In: *Electron. J. Statist.* 11.1 (2017), pp. 211–214. DOI: `10.1214/17-EJS1224`. URL: `https://doi.org/10.1214/17-EJS1224`.

[14] Susan Shortreed, Mark Handcock, and Peter Hoff. “Positional Estimation Within a Latent Space Model for Networks”. In: *Methodology: European Journal of Research Methods for the Behavioral and Social Sciences* 2 (Jan. 2006), pp. 24–33. DOI: `10.1027/1614-2241.2.1.24`.

[15] Babak Shahbaba et al. “Split Hamiltonian Monte Carlo”. In: (2011). arXiv: `1106.5941 [stat.CO]`.

[16] Matthew D. Hoffman and Andrew Gelman. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo”. In: (2011). arXiv: `1111.4246 [stat.CO]`.

[17] Iain Murray, Ryan Prescott Adams, and David J. C. MacKay. “Elliptical slice sampling”. In: (2009). arXiv: `1001.0175 [stat.CO]`.

[18] Bob Carpenter et al. “Stan: A Probabilistic Programming Language”. In: *Journal of Statistical Software, Articles* 76.1 (2017), pp. 1–32. ISSN: 1548-7660. DOI: 10.18637/jss.v076.i01. URL: https://www.jstatsoft.org/v076/i01.

[19] Dougal Maclaurin and Ryan P. Adams. “Firefly Monte Carlo: Exact MCMC with Subsets of Data”. In: (2014). arXiv: 1403.5693 [stat.ML].

[20] Michalis K. Titsias and Petros Dellaportas. *Gradient-based Adaptive Markov Chain Monte Carlo*. 2019. arXiv: 1911.01373 [stat.ML].

[21] Ahmed M. Alaa and Mihaela van der Schaar. *AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning*. 2018. arXiv: 1802.07207 [cs.LG].

[22] Jeffrey S. Rosenthal Gareth O. Roberts. “Examples of Adaptive MCMC”. In: *Journal of Computational and Graphical Statistics* 18.2 (2009), pp. 349–367. DOI: https://www.jstor.org/stable/25651249.