Adaptive Component-wise Multiple-Try Metropolis Sampling

Jinyoung Yang[*] , Radu Craiu[†] , and Jeffrey S. Rosenthal[‡]

March, 2016

Abstract

One of the most widely used samplers in practice is the component-wise Metropolis-Hastings (CMH) sampler that updates in turn the components of a vector spaces Markov chain using accept-reject moves generated from a proposal distribution. When the target distribution of a Markov chain is irregularly shaped, a 'good' proposal distribution for one part of the state space might be a 'poor' one for another part of the state space. We consider a component-wise multiple-try Metropolis (CMTM) algorithm that can automatically choose a better proposal out of a set of proposals from different distributions. The computational efficiency is increased using an adaptation rule for the CMTM algorithm that dynamically builds a better set of proposal distributions as the Markov chain runs. The ergodicity of the adaptive chain is demonstrated theoretically. The performance is studied via simulations and real data examples.

[*]Department of Statistics, University of Toronto, Toronto, Ontario, Canada M5S 3G3. Email: jinyoung.yang@mail.utoronto.edu

[†]Department of Statistics, University of Toronto, Toronto, Ontario, Canada M5S 3G3. Email: craiu@utstat.utoronto.ca

[‡]Department of Statistics, University of Toronto, Toronto, Ontario, Canada M5S 3G3. Email: jeff@math.toronto.edu

1 Introduction

Markov chain Monte Carlo (MCMC) methods are widely used to analyze complex probability distributions, especially within the Bayesian inference paradigm. One of the most used MCMC algorithms is the Metropolis-Hastings (MH) algorithm, first developed by Metropolis et al. (Metropolis et al. (1953)), and later expanded by Hastings (Hastings (1970)). At each iteration the MH algorithm samples a candidate new state from a proposal distribution which is subsequently accepted or rejected. When the state space of the chain is high dimensional or irregularly shaped, finding a good proposal distribution that can be used to update *all* the components of the chain simultaneously is very challenging, often impossible. The optimality results for the acceptance rate of the Metropolis-Hastings algorithm (Gelman et al., 1996; Roberts and Rosenthal, 2001) have inspired the development of the so-called *adaptive MCMC (AMCMC)* samplers that are designed to adapt their transition kernels based on the gradual information about the target that is collected through the very samples they produce. Successful designs can be found in Haario et al. (2001), Haario et al. (2006), Turro et al. (2007),Roberts and Rosenthal (2009), Craiu et al. (2009), Giordani and Kohn (2010), and Vihola (2012) among others. Theoretical difficulties arise because the adaptive chains are no longer Markovian so ergodicity properties must be proven on a case-by-case basis. Attempts at streamlining the theoretical validation process for AMCMC samplers have been increasingly successful including Atchadé and Rosenthal. (2005), Andrieu and Moulines (2006), Andrieu and Atchadé (2007), Roberts and Rosenthal (2007), Fort et al. (2011) and Craiu et al. (2015). For useful reviews of AMCMC we refer to Andrieu and Thoms (2008) and Roberts and Rosenthal (2009). Despite many success stories, it is our experience that existing adaptive strategies for MH may take a very long time to "learn" good simulation parameters or may be remain inefficient in high dimensions and with irregular shaped targets.

One can increase the computational efficiency if instead of using a full MH to update all the components at once, one chooses to update the components of the chain one-at-a-time. In this case the update rule follows the MH transition kernel but the acceptance or rejection is based on the target's conditional distribution of that component given all the

other ones. More precisely, if we are interested in sampling from the continuous density $\pi(x) : \mathcal{X} \subset \mathbf{R}^d \to \mathbf{R}_+$; the component-wise MH (CMH) will update the ith component of the chain, x_i, using a proposal $y_i \sim T_i(\cdot|x_i)$ and setting the next value of the chain as

$$
z = \begin{cases} (x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_d) & \text{w.p. } \alpha_i \\ x & \text{w.p. } 1 - \alpha_i \end{cases}
$$

where

$$
\alpha_i = \min\left\{1, \frac{T(x_i|y_i)\pi(y_i|x_{[-i]})}{T(y_i|x_i)\pi(x_i|x_{[-i]})}\right\},
$$

and $\pi(\cdot|x_{[-i]})$ is the target conditional distribution of the ith component given all the other components $x_{[-i]} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$. One can see that the CMH replaces the difficult problem of finding one good proposal in d dimensions with an apparently easier problem of finding d good 1-dimensional proposals. However, the latter task can also prove difficult if the conditional density $\pi(\cdot|x_{[-i]})$ turns out to have different properties as $x_{[-i]}$ varies. Intuitively, imagine that for a region of the sample space of $x_{[-i]}$ the proposal T_i should have a higher spread for the chain to mix well and for the remaining support T_i should have a smaller spread. Some success has been obtained in lower dimensions or for distributions with a well-known structure using the regional adaptive MCMC strategies of Craiu et al. (2009) or Bai et al. (2011), but extending those approaches will be very cumbersome when d is even moderately large and the geography of π exhibits unknown irregularities. Other adaptive MCMC ideas proposed for the CMH too include Haario et al. (2005) where the authors propose to use component-wise random walk Metropolis (RWM) and to use the component-specific sample variance to tune the proposal's variance, along the same lines that were used by Haario et al. (2001) to adapt the proposal distribution for the joint RWM. Another intuitive approach is proposed in Roberts and Rosenthal (2009). The algorithm we propose here will be compared with these existing algorithms in the simulations section.

The strategy we propose here aims to close the gap that still exists between AMCMC and efficient CMH samplers. When contemplating the problem, one may be tempted to try to "learn" each conditional distribution $\pi(\cdot|x_{[-i]})$, but parametric models are likely not flexible enough and nonparametric models will face the curse of dimensionality even for moderate values of d. Note that here the difficult part is understanding how the conditional distribution changes as $x_{[-i]}$ varies, which is a $d-1$-dimensional problem.

Before getting to the technical description of the algorithm, we present here the intuitive idea behind our design. Within the CMH algorithm imagine that for each component we can propose m candidate moves, each generated from m different proposal distributions. Naturally, the latter will be selected to have a diverse range of variances so that we generate some proposals close to the current location of the chain and some that are further away. If we assume that the transition kernel for each component is such that among the proposed states it will select the one that is most likely to lead to an acceptance, then one can reasonably infer that this approach will improve the mixing of the chain provided that the proposal distributions are reasonably calibrated. To mirror the discussion above, in a region where T_i should have small spread, one wants to have among the proposal distributions a majority with small variances, and similarly in regions where T_i should be spread out we want to include among our proposal densities a majority with larger variances.

This intuition can be put to rigorous test using an approach based on the Multiple-try Metropolis (MTM) that originated with Liu et al. (2000) and was further generalized by Casarin et al. (2013). The theoretical validity of the sampler is demonstrated using the results in Craiu et al. (2015) and the refinement brought by Rosenthal and Yang (2016).

Section 2 introduces a component-wise multiple-try Metropolis (CMTM) algorithm; in Section 3 we add the adaptive flavour to CMTM so the proposal distributions will modify according to the local shape of the target distribution and we prove the validity of our construction. Section 4 applies the adaptive CMTM algorithm to numerical examples. Section 5 compares the efficiency of the adaptive CMTM algorithm to other adaptive Metropolis algorithms.

2 Component-wise Multiple-Try Metropolis

2.1 Algorithm

Assume that a Markov chain $\{X_n\}$ is defined on $\mathcal{X} \subset \mathbf{R}^d$ with a target distribution π, and $T_1, \ldots, T_m$ are proposal distributions, each of which generates a new proposal y_j at every iteration. The transition rule for the MTM algorithm is described below.

1. Let $X_n = x$. Draw proposals $y_1, \ldots, y_m$ where $y_j \sim T_j(\cdot|x)$.

2. Compute

$$w_j(y_j, x) = \pi(y_j)T_j(x|y_j)\lambda_j(y_j, x), \tag{2.1}$$

for each y_j, where $\lambda_j(x, y)$ is a nonnegative symmetric function and $\lambda_j(x, y) > 0$ whenever $T_j(y|x) > 0$.

3. Select one $y = y_s$ out of $y_1, \ldots, y_m$ with probabilities proportional to $w_j(y_j, x)$.

4. Draw $x_1^*, \ldots, x_{s-1}^*, x_{s+1}^*, \ldots, x_m^*$ where $x_j^* \sim T_j(\cdot|y)$ and let $x_s^* = x$

5. Accept y with a probability

$$\rho = \min\left[1, \frac{w_1(y_1, x) + \ldots + w_m(y_m, x)}{w_1(x_1^*, y) + \ldots + w_m(x_m^*, y)}\right]$$

Throughout the paper, we use the component-wise multiple-try Metropolis (CMTM) algorithm in which each coordinate is updated using an MTM transition kernel with Gaussian proposals for each coordinate. More precisely, suppose that we are updating the kth component of the current state x using m 1-dimensional proposals, $T_j(\cdot|x)$, $1 \leq j \leq m$. Then a candidate $y_j \sim T_j(\cdot|x)$ is generated by sampling $z_k \sim N(x_k, \sigma_{k,j}^2)$ and by replacing the kth coordinate of current state x with z_k, i.e. $y_j = (x_1, \ldots, x_{k-1}, z_k, x_{k+1}, \ldots, x_d)$. Similarly, within the same iteration we get $x_j^* \sim T_j(\cdot|y)$, by replacing k^{th} coordinate of y with $z_k^* \sim N(y_k, \sigma_{k,j}^2)$.

Whether a proposal distribution is 'good' or not will depend on the current state of the Markov chain, especially if the target distribution π has irregular shaped support. In addition to choosing the m proposals, an added flexibility of the CMTM algorithm is that we can choose any nonnegative symmetric map λ with $\lambda_j(x, y) > 0$ whenever $T_j(y|x) > 0$. In subsequent sections we show that the CMTM algorithm with a particular form of the function $\lambda(x, y)$ influences positively the mixing of the chain and its effects depend in a subtle manner on the local shape, i.e. the geography of the target distribution around the current state, and the proposal's scale.

Our choice of λ is guided by a simple and intuitive principle. Between two candidate moves y_1 and y_2 that are equally far from the current state we favour y_1 over y_2 if $\pi(y_1)$ is greater than $\pi(y_2)$, but if $\pi(y_1)$ is similar to $\pi(y_2)$, we would like CMTM to favour whatever candidate is further away from the current state. These simple rules lead us to

consider

$$\lambda_j(x, y) = T_j(y|x)^{-1}\|(y - x)\|^\alpha, \tag{2.2}$$

where $\|\cdot\|$ is the Euclidean norm. Note that this choice of λ is possible because $T_j(y|x)$ is a symmetric function in x and y as it involves only one draw from a normal distribution with mean x_k.

Replacing (2.2) in the weights equation (2.1) results in

$$w_j(y_j, x) = \pi(y_j)T_j(x|y_j)\lambda_j(y_j, x)$$
$$= \pi(y_j)\|(y - x)\|^\alpha. \tag{2.3}$$

With this choice of λ, the selection probabilities are only dependent on the value of the target density at the candidate point y_j and the size of the potential jump of the chain, were this candidate accepted. From (2.2) we can see that the size of α will balance of importance of the attempted jump distance from the current state over the importance of the candidate under π. However, while we understand the trade-off imposed by the choice of α for selecting a candidate move, it is less clear how it will impact the overall performance of the CMTM, e.g acceptance rate or average jump distance.

Therefore, it is paramount to gauge what are good choices for the parameter α for the mixing of the CMTM chain. In the next section we tackle this using the average squared jumping distance as the measure of performance of a Markov chain. An estimate is obtained by averaging over the realized path of the chain. If a new proposal is rejected and $(X_{n+1} - X_n)^2$ is equal to zero, that contribution is not discarded. Because of this, the measure of efficiency takes into account not only the jump distance but also the acceptance rate, a combination that has turned out to be useful in other AMCMC designs (see for instance Craiu et al., 2009).

2.2   Optimal α

In order to study the influence of the parameter α on the CMTM efficiency we have conducted a number of simulation studies, some of which are described here.

We considered first a 2-dimensional mixture of two normal distributions

$$\frac{1}{2} * N(\mu_1, \Sigma_1) + \frac{1}{2} * N(\mu_2, \Sigma_2) \qquad (2.4)$$

where

$$\begin{cases} \mu_1 & = (5,0)^T \\ \mu_2 & = (15,0)^T \\ \Sigma_1 & = diag(6.25, 6.25) \\ \Sigma_2 & = diag(6.25, 0.25) \end{cases}$$
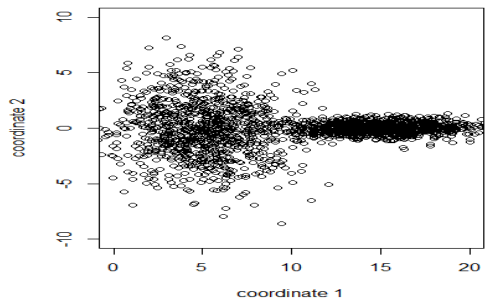

Figure 2.1: Target density plot. 2-dimensional mixture of two normals

An iid sample of size 2000 from (2.4) is plotted in Figure 2.1. We run the CMTM algorithm repeatedly with $\lambda_j(x, y_j)$ functions in (2.2) while changing the value of α from 0.1 to 15. We choose $m = 5$ as the number of proposals for each coordinate, while the proposal variances $\sigma_{k,j}$'s are for each coordinate $1, 2, 4, 8$ and 16.

As we see in Figure 2.2, the proportion of each proposal distribution selected increases/decreases as α changes. As expected, when α increases we see the selection percentages of the proposal distributions with smaller $\sigma_{k,j}$'s drop and those with larger $\sigma_{k,j}$'s increase. Figure 2.2 shows, with larger α's, our algorithm favours proposal distributions with larger scales, which makes sense based on the equation (2.3).
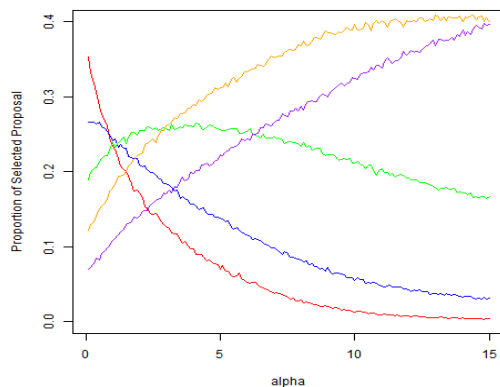

Figure 2.2: Proportion of proposal distribution selected. Coordinate 1: Red, Blue, Green, Orange and Purple lines show behaviour when $\sigma_{k,j} = 1, 2, 4, 8, 16$, respectively.

Figure 2.3 shows how the average squared jumping distance changes as the value of α changes. From Figure 2.3a, we see that the average squared jumping distance peaks in-between $\alpha = 2$ and $\alpha = 4$. Next, we run the CMTM algorithm independently 100

times for each of $\alpha = 2.0, 2.1, \ldots, 3.5$ and average the average squared jumping distances over the 100 runs. From Figure 2.3b we can infer that the highest efficiency is achieved for $\alpha \in (2.5, 3.2)$.
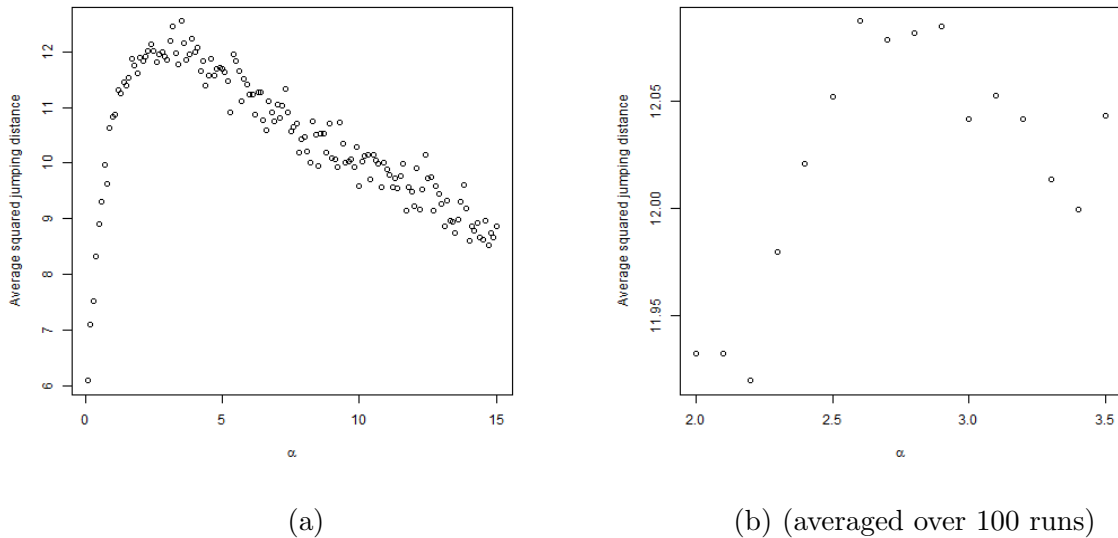

(a)   (b) (averaged over 100 runs)

Figure 2.3: Two-Dimensional Mixture of two Gaussians: Mean squared jumping distance vs. α for one run (left panel) and averaged over 100 runs (right panel).

We also examined a 4-dimensional mixture of two normal distributions as our target density:

$$\frac{1}{2} * N(\mu_1, \Sigma_1) + \frac{1}{2} * N(\mu_2, \Sigma_2),$$

where

$$\begin{cases} \mu_1 &= (5, 5, 0, 0)^T \\ \mu_2 &= (15, 15, 0, 0)^T \\ \Sigma_1 &= diag(6.25, 6.25, 6.25, 0.01) \\ \Sigma_2 &= diag(6.25, 6.25, 0.25, 0.01). \end{cases}$$

The number of proposals, $m = 5$ and $\sigma_{k,j}$'s of the set of proposal distributions for each coordinate are $0.5, 1, 2, 4$ and 8. Figure 2.4 shows the results. Figure 2.4a shows the average squared jumping distance is the largest between $\alpha = 2$ and $\alpha = 4$. After 100

independent replicates for each $\alpha = 2.0, 2.1, \ldots, 3.5$, we can see from Figure 2.4b that the average squared jumping distances are largest for $\alpha \in (2.5, 3.2)$.
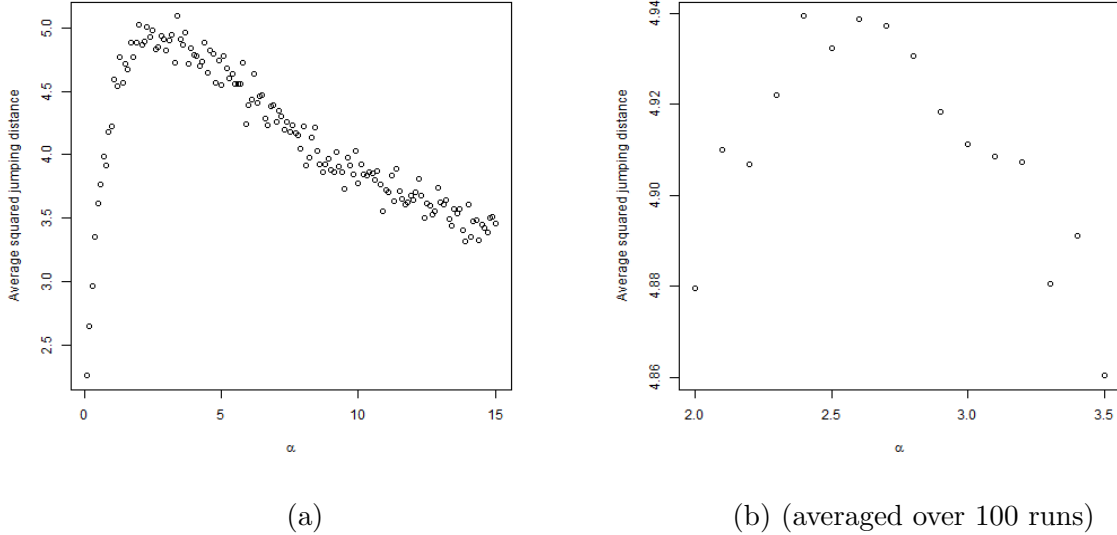

(a) (b) (averaged over 100 runs)

Figure 2.4: 4-Dimensional Mixture of two Gaussians: Means squared jumping distance vs. α for one run (left panel) and averaged over 100 runs (right panel).

Other numerical experiments not reported here agree with the two examples presented and suggest that optimal values of α are between 2.5 and 3.2. Henceforth we fix $\alpha = 2.9$ in all simulations involving CMTM.

3 Adaptive Component-wise Multiple-Try Metropolis

3.1 CMTM Favours Locally 'Better' Proposal Distributions

The intuition behind our construction as described in the Introduction, relies on the idea that CMTM will automatically tend to choose the "right" proposal among the m possible ones. In this section we verify empirically that this is indeed the case.

We consider the same 2-dimensional mixture of normal distributions from Section 2.2 as our target distribution and run the CMTM algorithm. The simulation parameter are

the same, $m = 5$ and $\sigma_{k,j} = 1, 2, 4, 8, 16$. As shown in Figure 2.1, the shape of our target distribution is quite different between the two regions of $\{X_{n,1} < 10\}$ and $\{X_{n,1} \geq 10\}$.

Table 3.1: Proportion of selected proposals

coordinate 1

	$\sigma_{k,j}$				
	1	2	4	8	16
$X_{n,1} < 10$	0.05	0.15	0.28	0.31	0.22
$X_{n,1} \geq 10$	0.04	0.12	0.25	0.34	0.25

coordinate 2

	$\sigma_{k,j}$				
	1	2	4	8	16
$X_{n,1} < 10$	0.06	0.19	0.33	0.26	0.16
$X_{n,1} \geq 10$	0.37	0.29	0.19	0.11	0.05

Table 3.2: Acceptance rate of selected proposals

coordinate 1

	$\sigma_{k,j}$				
	1	2	4	8	16
$X_{n,1} < 10$	0.39	0.50	0.51	0.53	0.50
$X_{n,1} \geq 10$	0.40	0.44	0.44	0.45	0.38

coordinate 2

	$\sigma_{k,j}$				
	1	2	4	8	16
$X_{n,1} < 10$	0.46	0.52	0.50	0.47	0.44
$X_{n,1} \geq 10$	0.44	0.41	0.30	0.28	0.28

Tables 3.1 and 3.2 present the proportion of candidate selection and acceptance rates for each proposal. We compare the proportion of proposals selected in the regions $\{X_{n,1} \geq 10\}$ and $\{X_{n,1} < 10\}$. While these regions are defined based on knowing the target exactly, they do not enter in any way in the design of the CMTM and are used here only to verify that the sampler indeed automatically adapts to local characteristics of the target. We can see that the CMTM favours the proposal distributions with smaller $\sigma_{k,j}$'s in the region $\{X_{n,1} \geq 10\}$ which seems appropriate given that in that region larger moves for the second coordinate will tend to be rejected. This pattern does not hold for the first coordinate for which larger moves are appropriate throughout the sample space. In addition, Table 3.1 shows that in the region where $\{X_{n,1} < 10\}$ the proportions of proposal distribution selected are similar for the two coordinates. This is in line with what is expected since the target variances $(= 6.25)$ are the same in both directions in that region.

This suggests that indeed the CMTM algorithm tends to choose the 'better' proposal distribution out of the available choices provided at each iteration. Also, as shown in Table 3.2, none of proposal distributions once selected has too high or too low acceptance rate even though their variances are quite different. This also hints that the CMTM assigns a selection probability to the multiple proposals at each iteration in such a way that

improves the efficiency of the Markov chain by avoiding too low and too high acceptance rate. This observation will help us design the adaptive version of the CMTM.

3.2   Comparison with a Mixture Transition Kernel

An astute reader may wonder about a different strategy for using the different proposals that one may have at one's disposal. Maybe the most natural alternative is a random mixture of the component-wise Metropolis algorithms. The set of proposal distributions used in both algorithms is the same and we assign equal weights for the proposal distributions in the mixture. This comparison is to see empirically how automatically adjusting the selection probabilities of proposal distributions in the CMTM algorithm improves the efficiency of a MCMC algorithm compared to the uniform selection of proposal distributions. Our target distribution is the 4-dimensional mixture of two normals introduced in Section 2.2. The $\sigma_{k,j}$'s for both algorithms are $0.5, 1, 2, 4$ and 8 for each coordinate, and each independent run has 60000 iterations (15000 updates per coordinate) in total.

A useful comparison of the efficiency of MCMC algorithms is based on estimates of integrated autocorrelation time (ACT) which can be calculated using

$$\tau = 1 + 2 \sum_{k=1}^{\infty} \rho_k,$$

where $\rho_k = Cov(X_0, X_k)/Var(X_0)$ is the autocorrelation at lag k. Higher ACT for a Markov chain implies successive samples are highly correlated, which reduces the effective information contained in any given number of samples produced by the chain.

To compare the efficiency of two algorithms, we compare the ACT calculated from the runs of two algorithms. We also look at the CPU time taken for the runs (for the same number of iterations) and the average squared jumping distances. We calculate the ACT only from the second-half the chain since we discard the first-half as burn-in, but we calculate the average squared jumping distance for the whole length of the chain.

Table 3.3 shows the proportion of each proposal distribution selected for the CMTM algorithm. It is quite different from what can be obtained by the uniform sampling, which is used for the random mixture of the component-wise Metropolis algorithm. By assigning different selection probabilities than uniform, the CMTM algorithm controls

11

the acceptance rate, avoiding too high and too low acceptance rate, as shown in Table 3.4.

Table 3.4: Acceptance rate on selected proposals

(a) Mix. of component-wise Metropolis

	$\sigma_{k,j}$				
	0.5	1	2	4	8
coord1	0.94	0.88	0.76	0.56	0.36
coord2	0.94	0.89	0.76	0.58	0.35
coord3	0.83	0.70	0.54	0.38	0.23
coord4	0.25	0.12	0.06	0.04	0.02

(b) CMTM

	$\sigma_{k,j}$				
	0.5	1	2	4	8
coord1	0.41	0.49	0.55	0.53	0.47
coord2	0.40	0.49	0.55	0.53	0.45
coord3	0.48	0.47	0.50	0.50	0.47
coord4	0.27	0.27	0.27	0.31	0.29

Table 3.5: Performance comparison (averaged over 100 runs)

(a) Mix. of component-wise Metropolis

	Min.	Median	Mean	Max.
cputime(s)	14.92	19.08	19.98	29.94
sq. jump	1.443	1.555	1.553	1.650
	coord1	coord2	coord3	coord4
ACT	1109.67	1105.99	39.93	68.74

(b) CMTM

	Min.	Median	Mean	Max.
cputime(s)	126.3	144.3	146.0	197.5
sq. jump	4.657	4.913	4.921	5.131
	coord1	coord2	coord3	coord4
ACT	316.79	318.28	9.68	13.59

Table 3.5 compares the performance of two algorithms. We see that the average squared jumping distance significantly improves with the CMTM compared to the random mixture of the component-wise Metropolis. We can also see that the ACT is smaller for the CMTM than the random mixture of the component-wise Metropolis. Thus, we conclude that a selection probability assignment by the CMTM algorithm through the $w_j(y_j|x)$ function in (2.3) improves the efficiency of the CMTM algorithm. One concern for the CMTM is the increased computation cost. We see in Table 3.5 the runtime for the CMTM is about 7 to 8 times longer than the random mixture of the component-wise Metropolis. We will discuss the efficiency issue further in Section 5.

Table 3.3: CMTM. Proportion of proposal distribution selected

	$\sigma_{k,j}$				
	0.5	1	2	4	8
coord1	0.02	0.07	0.22	0.37	0.32
coord2	0.02	0.07	0.21	0.36	0.34
coord3	0.13	0.20	0.23	0.25	0.19
coord4	0.56	0.24	0.12	0.06	0.03

3.3 The Adaptive CMTM

Given its propensity to choose the best candidate put forward by the proposal distributions, it is reasonable to infer that CMTM's performance will be roughly aligned with the most suitable proposal for the region where the chain current state lies. The other side of the coin is that a whole set of bad proposals will compromise the efficiency of the CMTM algorithm. Therefore, we focus our efforts in developing an adaptive CMTM (AMCTM) design that aims to minimize, possibly annihilate, the chance of having at our disposal only poorly calibrated proposal distributions in any region of the space.

The adaptation strategy is centred on finding well-calibrated values for the set $S_k = \{\sigma_{k,j} : 1 \leq j \leq m\}$ for every coordinate $1 \leq k \leq d$. Note that S_k varies across coordinates.

Consider an arbitrarily fixed coordinate and suppose we label the m proposal distributions such that $\sigma_{k,1} < \sigma_{k,2} < \ldots < \sigma_{k,m}$. Changes in the kernel occur at fixed points in the simulation process, called *adaption points*. The changes will occur only if an alarm is triggered at an adaption point. An alarm is triggered only if we notice that the candidates generated by the proposal distributions with the smallest scale $\sigma_{k,1}$ or the largest one $\sigma_{k,m}$ are over-selected. For instance, suppose that in an inter-adaptation time interval the candidates generated by $\sigma_{k,1}$ are selected more than 40% of the time. The latter threshold the latter threshold is more than double the selection percentage for the least selected candidate since, if we denote p_j the frequency of selecting the candidate generated using $\sigma_{k,j}$ we have $\sum_{p_j} = 1 \geq m \min p_j$ and $m = 5$ in our implementation. The high selection percentage for $\sigma_{k,1}$ suggests that the chain tends to favour, when updating the kth coordinate, proposals with smaller scale so the ACMTM design requires to: 1) half the value of $\sigma_{k,1}$; 2) not modify the largest element in S_k; 3) recalculate the intermediate values, $\sigma_{k,2}, \ldots, \sigma_{k,m-1}$ to be equidistant between $\sigma_{k,1}$ and $\sigma_{k,m}$ on the log-scale.

Similarly, if the largest element in S_k, $\sigma_{k,m}$, produces proposals that are selected with frequency higher than 40% we consider this indicative of the chain requiring larger scale proposals for the kth coordinate and we replace $\sigma_{k,m}$ by its double, we keep the smallest value in S_k and the intermediate values are recalculated to be equidistant on log-scale.

If neither the smallest nor the largest elements in S_k produce proposals that are

selected more than 40% of the time, we wait until the algorithm reaches the next 'adaption point' and recalculate the proportion of each proposal candidate being selected during the last inter-adaption time interval.

An adaptive algorithm must adapt less and less as the simulation proceeds, a condition known as *diminishing adaptation (DA)* and long recognized to be essential for the chain's valid asymptotic behaviour (Roberts and Rosenthal, 2007). However, the adaption strategy proposed above may not diminish in the long run, so we ensure the DA condition is satisfied by allowing a modification at the rth adaption point with probability $p_r \leq 1$ where

$$p_r = \max(0.99^{r-1}, \frac{1}{\sqrt{r}}), \qquad r = 1, 2, 3, \ldots \tag{3.1}$$

We chose p_r in (3.1) so that it is decreasing slowly and has high values at the beginning of the run when most calibrations will take place. The form of (3.1) along with the Borel-Cantelli lemma allows the adaption to keep occurring for as long as we run the chain since $\sum_{r=1}^{\infty} p_r = \infty$.

There are three minor technical details required in our adaptive CMTM algorithm to ensure the convergence of the algorithm. Those details are to satisfy the conditions listed in Rosenthal and Yang (2016), which extends the theory developed by Craiu et al. (2015). The convergence of our adaptive CMTM algorithm is proved in Section 3.5.

1. Fix a (large) constant $D > 0$. We reject Y_{n+1} if $|Y_{n+1} - X_n| > D$. If $|Y_{n+1} - X_n| < D$, we accept/reject Y_{n+1} by the usual rule for the CMTM algorithm described in Section 2.1.

2. Fix a (large) constant $L > 0$ and a (really small) constant $\epsilon > 0$. Let $\sigma_{n,k,j}$ be the $\sigma_{k,j}$ used at n-th iteration in our adaptive CMTM algorithm. If $\sigma_{n,k,j}$ obtained from the adaption is greater than L, set $\sigma_{n,k,j} = L$. If $\sigma_{n,k,j}$ obtained from the adaption is less ϵ, set $\sigma_{n,k,j} = \epsilon$. (Of course the initial $\sigma_{n,k,j}$, $\sigma_{0,k,j}$, should be $\epsilon \leq \sigma_{0,k,j} \leq L$.)

3. Define a non-empty bounded (large) subset $K \subset \mathcal{X}$. If $X_n \notin K$, then $Y_{n+1,k,j} \sim N(X_{n,k}, \sigma_{k,j}^{*2})$, where $Y_{n,k}$ and $X_{n,k}$ is the kth coordinate of Y_n and X_n respectively and $\sigma_{k,j}^*$ is fixed with $\epsilon \leq \sigma_{k,j}^* \leq L$. If $X_n \in K$, then $Y_{n+1,k,j} \sim N(X_{n,k}, \sigma_{n,k,j}^2)$.

3.4   To Adapt or Not To Adapt?

We compare the ACMTM algorithm with the CMTM algorithm without adaption to see if the adaption indeed improves the efficiency of the algorithm. We use the 4-dimensional mixture of two normal distributions from Section 2.2 as our target distribution. The $\sigma_{k,j}$'s for the non-adaptive algorithm are given in Table 3.6a and they are the same with the starting $\sigma_{k,j}$'s for the adaptive algorithm. We also provide in Table 3.6b the final versions of the $\sigma_{k,j}$'s obtained after the last adaption in one random run of ACMTM. For this particular run, the last adaption occurred right after 3600 iterations out of 40000 iterations in total. The comparison is based on 100 independent replicates, each of which yields the average squared jumping distance and the ACT.

Performance comparison is shown in Table 3.7.

Table 3.6: Ending $\sigma_{k,j}$'s

(a) Non-adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	16	16	16	1
prop2	32	32	32	2
prop3	64	64	64	4
prop4	128	128	128	8
prop5	256	256	256	16

(b) Adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	2.0000	2.0000	0.2500	0.0625
prop2	6.7272	6.7272	1.4142	0.2500
prop3	22.6274	22.6274	8.0000	1.0000
prop4	76.1093	76.1093	45.2548	4.0000
prop5	256.0000	256.0000	256.0000	16.0000

Table 3.7: Performance comparisons (averaged over 100 runs)

(a) Non-adaptive CMTM

	Min.	Median	Mean	Max.
cputime(s)	79.21	90.98	92.88	114.60
sq. jump	3.463	3.688	3.683	4.036

	coord1	coord2	coord3	coord4
ACT	336.82	332.02	19.39	26.00

(b) Adaptive CMTM

	Min.	Median	Mean	Max.
cputime(s)	77.85	92.49	92.61	120.10
sq. jump	4.180	4.715	4.693	5.181

	coord1	coord2	coord3	coord4
ACT	249.08	249.96	13.56	11.47

Table 3.8: Proportion of proposal distribution selected

(a) Non-adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	0.56	0.55	0.56	0.57
prop2	0.24	0.25	0.24	0.24
prop3	0.11	0.12	0.11	0.11
prop4	0.06	0.05	0.06	0.05
prop5	0.03	0.03	0.03	0.03

(b) Adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	0.33	0.33	0.17	0.31
prop2	0.46	0.45	0.44	0.50
prop3	0.16	0.16	0.32	0.14
prop4	0.05	0.05	0.06	0.04
prop 5	0.01	0.01	0.01	0.01

Table 3.9: Acceptance rate on selected proposals

(a) Non-adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	0.23	0.24	0.16	0.17
prop2	0.23	0.23	0.17	0.16
prop3	0.24	0.22	0.15	0.18
prop4	0.27	0.24	0.16	0.18
prop5	0.24	0.21	0.17	0.22

(b) Adaptive CMTM

	coord1	coord2	coord3	coord4
prop1	0.49	0.47	0.43	0.45
prop2	0.41	0.41	0.44	0.42
prop3	0.37	0.39	0.41	0.38
prop4	0.34	0.40	0.35	0.37
prop5	0.33	0.34	0.39	0.46

We notice that 'prop1' was the most favoured proposal distribution for every coordinate if there was no adaption (Table 3.8a) whereas with adaption it was 'prop2' selected most often (Table 3.8b). The $\sigma_{k,j}$'s got smaller with adaption (Table 3.6), and the two smallest $\sigma_{k,j}$'s obtained only after the multiple adaptions are the most favoured ones in the adaptive run (Table 3.8b). In return, the acceptance rates in the adaptive algorithm increase to the range of 0.36 to 0.50 compared to 0.15 to 0.27 in the non-adaptive algorithm, as shown in Table 3.9. It is important to note that the increase in acceptance rates is coupled with an increase in average square distance, as shown in Table 3.7. Altogether we can see from the same Table that these result in important reductions for the ACT. The runtime between the non-adaptive and the adaptive algorithm is pretty much the same as seen in Table 3.7. Altogether, the results show that using an adaptive strategy made a significant difference.

3.5 Convergence of Adaptive CMTM

To confirm the convergence of the adaptive CMTM algorithm described in Section 3.3, we verify the conditions listed in Theorem 2 of Rosenthal and Yang (2016). As explained in Section 3.3, Diminishing Adaptation condition holds by the construction of the adaption mechanism.

Theorem 1. *Consider the adaptive CMTM algorithm in Section 3.3 to sample from state space $\mathcal{X}$ that is an open subset of $\mathbf{R}^d$ for some $d \in \mathbf{N}$. Let π be a target probability distribution, which has a continuous positive density on $\mathcal{X}$ with respect to the Lebesgue measure. Then, the adaptive CMTM algorithm converges to stationarity as in*

$$\lim_{n \to \infty} \sup_{A \in \mathcal{F}} |\mathbf{P}(X_n \in A) - \pi(A)| = 0. \tag{3.2}$$

Proof. To prove the convergence of the algorithm as in (3.2), we follow the Theorem 2 of Rosenthal and Yang (2016).

Let m be the number of proposals at each iteration. Let $\mathcal{Y}$ be the collection of all $d \times m$ matrices which elements are positive real numbers in $[\epsilon, L]$ which implies that $\mathcal{Y}$ is compact. Let $\gamma \in \mathcal{Y}$ and Q_γ be a set of $d \times m$ Markov chain proposal kernels with each kernel $Q_{k,j}$ corresponding to the jth proposal kernel for k^{th} coordinate. By the description of the adaptive algorithm in Section 3.3, $Q_\gamma = Q^*$ whenever $X_n \notin K$, where Q^* is the set of $d \times m$ proposal kernels with each kernel $Q^*_{k,j}$ fixed throughout the simulation. Also, for all $Q_{k,j}$'s and $Q^*_{k,j}$'s, the densities of them equals to zero whenever $|Y_{n+1} - X_n| > D$. Then $\{Q_\gamma\}_{\gamma \in \mathcal{Y}}$ is the collection of all possible sets of proposal kernels to update all d coordinates in the adaptive CMTM algorithm. Based on the adaption rule, choosing which set of Q_γ's to be on at each iteration n is determined by the past and/or current information obtained from the chain.

As long as the Metropolis-Hastings algorithm is reversible and π is everywhere positive on $\mathcal{X}$, the transitions being truncated at some fixed distance D does not affect the ergodicity to π if the Metropolis-Hastings algorithm with the proposal kernels before truncated was ergodic to π. (Rosenthal and Yang (2016)) Thus, a non-adaptive CMTM algorithm with any Q_γ, even though the proposal kernels are truncated, is ergodic to π (Casarin et al. (2013), Liu et al. (2000)).

The condition (a), (b), (c), (d) and (e) from Rosenthal and Yang (2016) are verified as follows:

- Condition (a), the bounded jump condition, is satisfied because we reject Y_{n+1} if $|Y_{n+1} - X_n| > D > 0$.

- Condition (b), no adaption outside of a non-empty bounded subset $K \subset \mathcal{X}$, is satisfied since, as described in Section 3.3, we generate $Y_{n+1,k,j} \sim N(X_{n,k}, \sigma_{k,j}^{*2})$ with $\sigma_{k,j}^* \in \mathcal{Y}$ fixed whenever $X_n \notin K$.

- Also, we need to meet the condition (c) and (d) that the fixed proposal kernel $Q_{k,j}^*$ is bounded above and below. $Q_{k,j}^*(x, dy)$ needs to be bounded above by $MLeb(dy)$ for some $M > 0$ for $\forall x \in K_D \backslash K$ and $\forall y \in K_{2D} \backslash K$, where K_a is a set containing all $x \in \mathcal{X}$ with $\eta(x, K) \leq a$. (η is a metric defined on $\mathcal{X}$, e.g. the Euclidean distance when $\mathcal{X} \subset \mathbf{R}^d$.) Indeed, $Q_{k,j}^*(x, dy) \geq MLeb(dy)$ everywhere since our transition kernel $Q_{k,j}^*$ has a truncated normal density, satisfying condition (c) of Rosenthal and Yang (2016). We also need $Q_{k,j}^*(x, dy)$ to be bounded below. More specifically, we need some $\epsilon', \delta' > 0$ such that $Q_{k,j}^*(x, dy) \geq \epsilon' Leb(dy)$ whenever $|y - x| < \delta'$ and $x, y \in J$, where J is some bounded rectangle with $K_{2D} \backslash K_D \subset J \subset \mathcal{X}$. It is condition (d') in Rosenthal and Yang (2016), which implies condition (d) in our case since $\mathcal{X}$ is an open subset of $\mathbf{R}^d$. Because $Q_{k,j}^*$ is a normal distribution and π has a continuous positive density on $\mathcal{X}$ satisfies condition (d'), we conclude that condition (d) is satisfied.

- Our proposal density functions satisfy condition (e) from Rosenthal and Yang (2016) since they satisfy their conditions for combocontinuous (in particular, truncated) functions.

Since the ACMTM algorithm satisfies the diminishing adaptation condition as explained in Section 3.3, we can conclude that it satisfies all the conditions listed by Theorem 2 in Rosenthal and Yang (2016) so the adaptive CMTM algorithm converges to π as in (3.2). $\qquad \square$

4   Applications

In the following examples we compare the CMTM and the AMCTM when started with the same set of $\sigma_{k,j}$'s.

In a second comparison we also consider the CMTM and the CMH samplers in which we use the $\sigma_{k,j}$'s identified by the ACMTM adaptive process in the following way: we run one ACMTM sampler started at $\sigma_{k,j} = 2^j$ for $1 \leq j \leq m$ and $1 \leq k \leq d$ and we use the final $\sigma_{k,j}$'s from the run to set up the CMTM and the m CMH samplers. If we assume the labelling such that for each coordinate k we have $\sigma_{k,1} \leq \ldots \leq \sigma_{k,m}$ then the jth CMH sampler uses $\sigma_{k,j}$ to generate moves in the kth coordinate, $1 \leq k \leq d$. For the CMTM we use all the $\sigma_{k,j}$'s identified by the adaptive process. Throughout this section we use $m = 5$.

For all the examples we use the effective sample size (ESS) to compare the efficiency of MCMC algorithms. Since ESS $= w/\tau$, where w is the number of samples obtained from a Markov chain and τ is the ACT, one can see that ESS is tightly connected to the degree of correlation of the samples. One may intuitively interpret ESS the number of iid samples from the target that would contain the same amount of information about the target as the MCMC sample. Only the second half of the chain's run is used to calculate the ACT. We average the ACT over 50 or 100 independent runs, and with that average we calculate ESS.

4.1   Variance Components Model

The Variance Components Model (VCM) is a typical hierarchical model, well-used in Bayesian statistics community. Here, we use the data on batch to batch variation in dyestuff yields. The data were introduced in Davies (1967) and later analyzed by Box and Tiao (1973). The Bayesian set-up of the Variance Components Model on dyestuff yields is also well-described in Roberts and Rosenthal (2004). The data records yields on dyestuff of 5 samples, from each of 6 randomly chosen batches. The data is shown in Table 4.1.

19

Table 4.1: Dyestuff Batch Yield (in grams)

Batch 1	1545	1440	1440	1520	1580
Batch 2	1540	1555	1490	1560	1495
Batch 3	1595	1550	1605	1510	1560
Batch 4	1445	1440	1595	1465	1545
Batch 5	1595	1630	1515	1635	1625
Batch 6	1520	1455	1450	1480	1445

Let y_{ij} be the yield on the dyestuff batch, with i indicating which batch it is from and j indexing each individual sample from the batch. The Bayesian model is then constructed as:

$$y_{ij}|\theta_i, \sigma_e^2 \sim N(\theta_i, \sigma_e^2), \qquad i = 1, 2, ..., K, \quad j = 1, 2, ..., J$$

where $\theta_i|\mu, \sigma_\theta^2 \sim N(\mu, \sigma_\theta^2)$. θ_i's are conditionally independent of each other given μ, σ_θ^2. The priors for the $\sigma_\theta^2, \sigma_e^2$ and μ are: $\sigma_\theta^2 \sim IG(a_1, b_1)$, $\sigma_e^2 \sim IG(a_2, b_2)$ and $\mu \sim N(\mu_0, \sigma_0^2)$. Thus, the posterior density function of this VCM model is

$$f(\sigma_\theta^2, \sigma_e^2, \mu, \theta_i|y_{ij}, a_1, a_2, b_1, b_2, \sigma_0^2) \propto$$

$$(\sigma_\theta^2)^{-(a_1+1)} e^{-b_1/\sigma_\theta^2} (\sigma_e^2)^{-(a_2+1)} e^{-b_2/\sigma_e^2} e^{-(\mu-\mu_0)^2/2\sigma_0^2} \prod_{i=1}^{K} \frac{e^{(\theta_i-\mu)^2/2\sigma_\theta^2}}{\sigma_\theta} \prod_{i=1}^{K}\prod_{j=1}^{J} \frac{e^{(y_{ij}-\theta_i)^2/2\sigma_e^2}}{\sigma_e}$$

We set the hyperparameters $a_1 = a_2 = 300$ and $b_1 = b_2 = 1000$, making inverse gamma priors very concentrated. We also set $\sigma_0^2 = 10^{10}$.

Figure 4.1 shows ESS of the CMTM algorithms with and without adaption. For both CMTM algorithms (with and without adaption), the starting $\sigma_{k,j}$'s are 0.1, 0.2, 0.4, 0.8 and 1.6 for every coordinate.
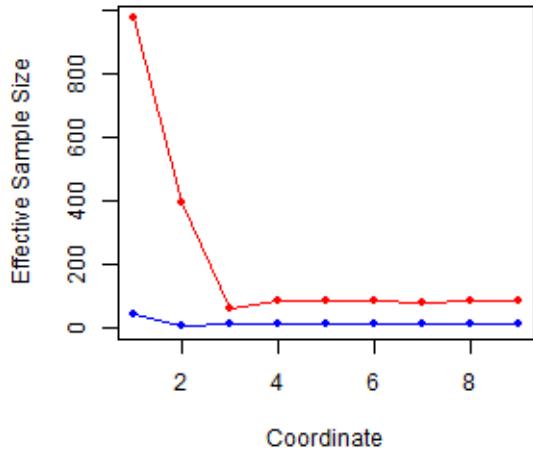
Figure 4.1: Adaptive CMTM vs. non-adaptive CMTM. Variance components model. The red represents the adptive CMTM runs and the blue represents the non-adaptive CMTM runs. ESS is calculated after averaging ACT over 50 independently replicated runs.

Note that ACMTM yields ESS's that are considerably larger than the corresponding values for CMTM.

Next, we want to compare the standard (non-adaptive) CMTM algorithm with the standard CMH.

The $\sigma_{k,j}$'s used to set up the CMH and CMTM samplers can be found in Table 4.2. With this proposal scales, we run the standard CMTM and the standard component-wise Metropolis and compare the ESS from these two algorithms, averaged over 100 runs.

Table 4.2: $\sigma_{k,j}$'s used in comparing the standard CMTM and the standard component-wise Metropolis. Variance components model

	coord1	coord2	coord3	coord4	coord5	coord6	coord7	coord8	coord9
prop1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
prop2	3.36	4.76	2.83	2.38	3.36	3.36	3.36	3.36	3.36
prop3	11.31	22.63	8.00	5.66	11.31	11.31	11.31	11.31	11.31
prop4	38.05	107.63	22.63	13.45	38.05	38.05	38.05	38.05	38.05
prop5	128.00	512.00	64.00	32.00	128.00	128.00	128.00	128.00	128.00
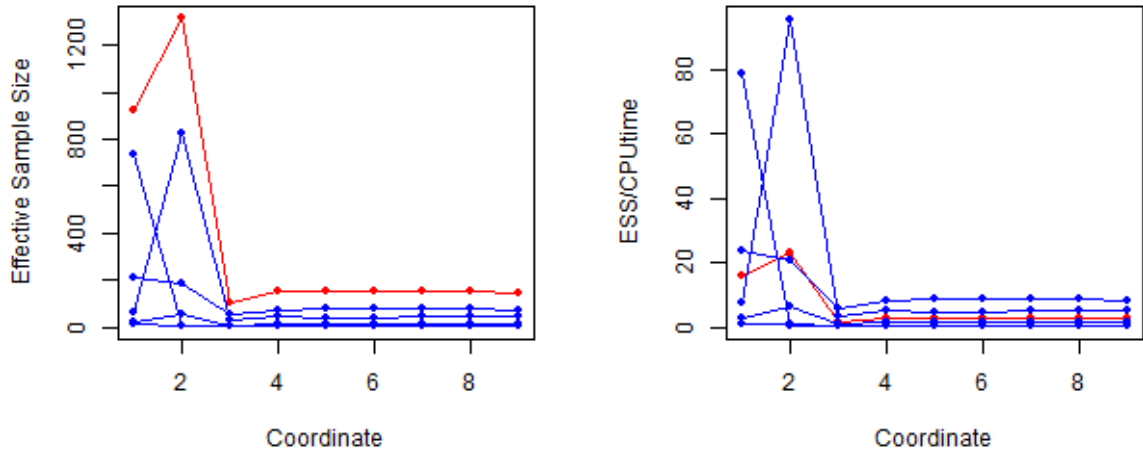
Figure 4.2: Standard (non-adaptive) CMTM vs. standard component-wise Metropolis. Variance components model. The red represents the CMTM runs and the blue represents the component-wise Metropolis runs. ESS is calculated after averaging ACT over 100 replicative runs.

Figure 4.2 shows that if we ignore the CPUtime loss for the CMTM algorithm compared to the component-wise Metropolis algorithm, the CMTM algorithm is the most efficient one based on ESS. However, when we divide ESS by CPUtime to account for the computation time, the CMTM becomes the third best algorithm compared with the 5 different component-wise Metropolis algorithms. We will discuss about this efficiency issue further in Section 5.

4.2 "Banana-shaped" Distribution

The "Banana-shaped" distribution was originally presented in Haario et al. (1999) as an irregularly-shaped target that may call for different proposal distributions for the different parts of the state space.

The target density function of the "banana-shaped" distribution is constructed as $f_B = f \circ \phi_B$, where f is the density of $d-$dimensional multivariate normal distribution $N(\mathbf{0}, \mathrm{diag}(100, 1, 1, \ldots, 1))$ and $\phi_B(\mathbf{x}) = (x_1, x_2 + Bx_1^2 - 100B, x_3, \ldots, x_d)$. $B > 0$ is the

nonlinearity parameter and the non-linearity or "bananacity" of the target distribution increases with B. The target density function is

$$f_B(x_1, x_2, \ldots, x_d) \propto exp[-x_1^2/200 - \frac{1}{2}(x_2 + Bx_1^2 - 100B)^2 - \frac{1}{2}(x_3^2 + x_4^2 + \ldots + x_d^2)].$$

We set $B = 0.01$ and $d = 10$ and set the starting $\sigma_{k,j}$'s equal to $0.1, 0.2, 0.4, 0.8$ and 1.6 for every coordinate. The results are shown in Figure 4.3. You see that the ESS is larger in every coordinate for the adaptive CMTM algorithm compared to the non-adaptive CMTM algorithm, confirming that the adaption improved the efficiency in this example.
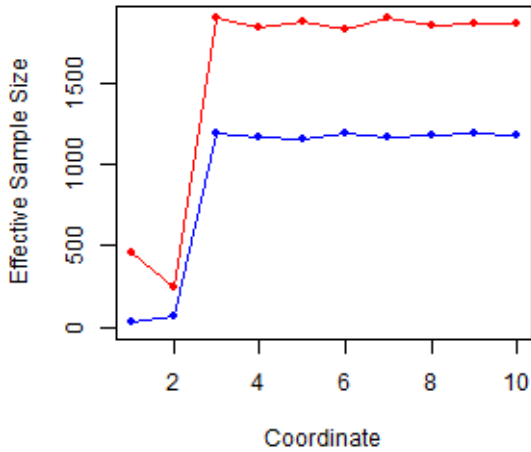
Figure 4.3: Adaptive CMTM vs. non-adaptive CMTM. "Banana-shaped" distribution. The red represents the adptive CMTM runs and the blue represents the non-adaptive CMTM runs. ESS is calculated after averaging ACT over 50 replicative runs.

In the second comparison we notice that in 45 out of 50 independently replicated runs, the starting $\sigma_{k,j}$'s did not change under the adaptive CMTM algorithm. Therefore, we take 1,2,4,8,16 for every coordinate as our default $\sigma_{k,j}$'s for the standard CMTM algorithm we run here. And for the standard CMH algorithms, we run with five different sets of proposal standard deviation, $(1, 1, \cdots, 1)$, $(2, 2, \cdots, 2)$, $(4, 4, \cdots, 4)$, $(8, 8, \cdots, 8)$, and $(16, 16, \cdots, 16)$, with one set at a time. As we see in Figure 4.4, ESS is larger in every coordinate for the CMTM compared to the component-wise Metropolis if CPUtime

is ignored. If we look at ESS/CPUtime, the CMTM is in the middle of the pack in terms of efficiency.
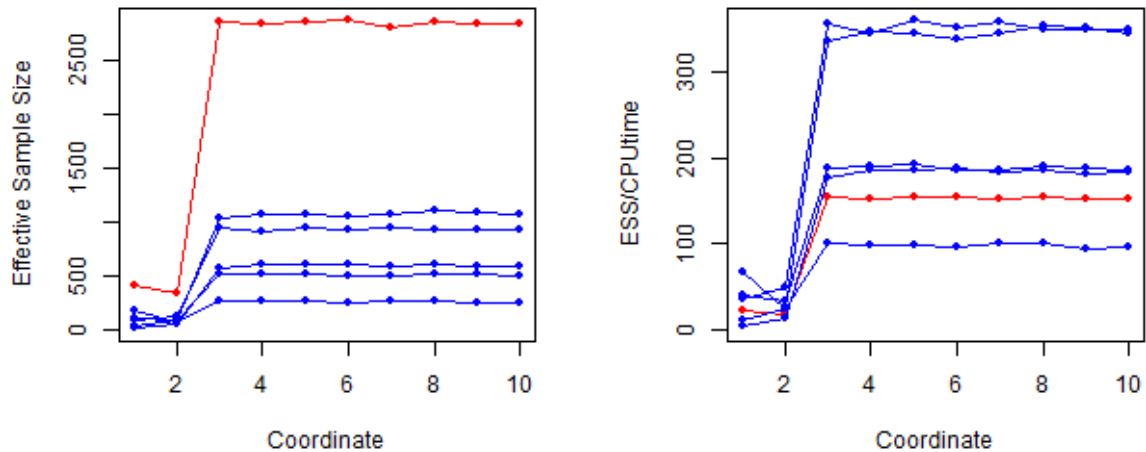

Figure 4.4: Standard (non-adaptive) CMTM vs. standard component-wise Metropolis. "Banana-shaped" distribution. The red represents the CMTM runs and the blue represents the component-wise Metropolis runs. ESS is calculated after averaging ACT over 50 replicative runs.

4.3   Orange Tree Growth Data

The orange tree growth data was first presented by Draper and Smith (1981) and was further analyzed by Lindstrom and Bates (1990). Here, we follow the Bayesian model set-up specified in Craiu and Lemieux (2007). The data consists of the measures of trunk circumferences of five different orange trees on seven different time points. You can find the data in Table 4.3.

Table 4.3: Growth of Orange Trees

Age (days)	Circumference (mm)				
	Tree1	Tree2	Tree3	Tree4	Tree5
118	30	33	30	32	30
484	58	69	51	62	49
664	87	111	75	112	81
1004	115	156	108	167	125
1231	120	172	115	179	142
1372	142	203	139	209	174
1582	145	203	140	214	177

Let y_{ij} be the trunk circumference measure, where $i = 1, \ldots, 5$ indexes for the five different trees and $j = 1, \ldots, 7$ indexes for the seven different time points. The logistic growth model has $y_{ij} \sim N(\mu_{ij}, \sigma_c^2)$ where

$$\mu_{ij} = \frac{exp(\theta_{i1})}{1 + (exp(\theta_{i2}) - 1)exp(-exp(\theta_{i3})x_j)}$$

with x_j being the time point. The priors for the σ_c^2 and the θ_{ik}, $k = 1, 2, 3$, are: $\sigma_c^2 \sim IG(0.001, 0.001)$ and $\theta_{ik} \sim N(0, 100)$. Note that θ_{ik}'s are independent of each other and y_{ij} are conditionally independent of each other given μ_{ij} and σ_c^2. Thus, the posterior density of interest is

$$f(\theta_{ik}, \sigma_c^2 | y_{ij}, x_j) \propto (\sigma_c^2)^{-(0.001+1)} e^{-0.001/\sigma_c^2} \prod_{i=1}^{5} \prod_{k=1}^{3} \frac{e^{(\theta_{ik})^2/200}}{10}$$
$$\times \prod_{i=1}^{5} \prod_{j=1}^{7} \frac{e^{(y_{ij} - \frac{exp(\theta_{i1})}{1+(exp(\theta_{i2})-1)exp(-exp(\theta_{i3})x_j)})^2/2\sigma_c^2}}{\sigma_c}.$$

First, we compare the adaptive CMTM with the non-adaptive CMTM. The starting $\sigma_{k,j}$'s here for the both algorithms are 1, 2, 4, 8, and 16 for every coordinate and the results, shown in Figure 4.5, confirm that the ESS is larger in every coordinate for the adaptive CMTM algorithm.
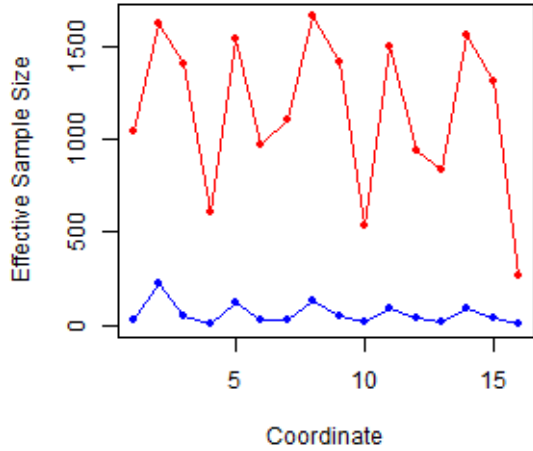
Figure 4.5: Adaptive CMTM vs. non-adaptive CMTM. Orange tree growth data. The red represents the adptive CMTM runs and the blue represents the non-adaptive CMTM runs. ESS is calculated after averaging ACT over 50 replicative runs.

When comparing the CMTM with the CMH the proposal scales used can be found in Table 4.4

Table 4.4: $\sigma_{k,j}$'s used in comparing the standard CMTM and the standard component-wise Metropolis. Orange tree growth data

	coord1	coord2	coord3	coord4	coord5	coord6	coord7	coord8
prop1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
prop2	2.38	2.83	2.83	2.38	2.83	2.38	2.38	2.83
prop3	5.66	8.00	8.00	5.66	8.00	5.66	5.66	8.00
prop4	13.45	22.63	22.63	13.45	22.63	13.45	13.45	22.63
prop5	32.00	64.00	64.00	32.00	64.00	32.00	32.00	64.00
	coord9	coord10	coord11	coord12	coord13	coord14	coord15	coord16
prop1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
prop2	2.38	2.38	2.83	2.38	2.38	2.83	2.38	22.63
prop3	5.66	5.66	8.00	5.66	5.66	8.00	5.66	512.00
prop4	13.45	13.45	22.63	13.45	13.45	22.63	13.45	11585.24
prop5	32.00	32.00	64.00	32.00	32.00	64.00	32.00	262144.00

The results for the runs are shown in Figure 4.6. Again, ESS is larger in every coordinate for the CMTM algorithm compared to the component-wise Metropolis algorithm, but ESS/CPUtime is only the fourth largest for the CMTM algorithm.
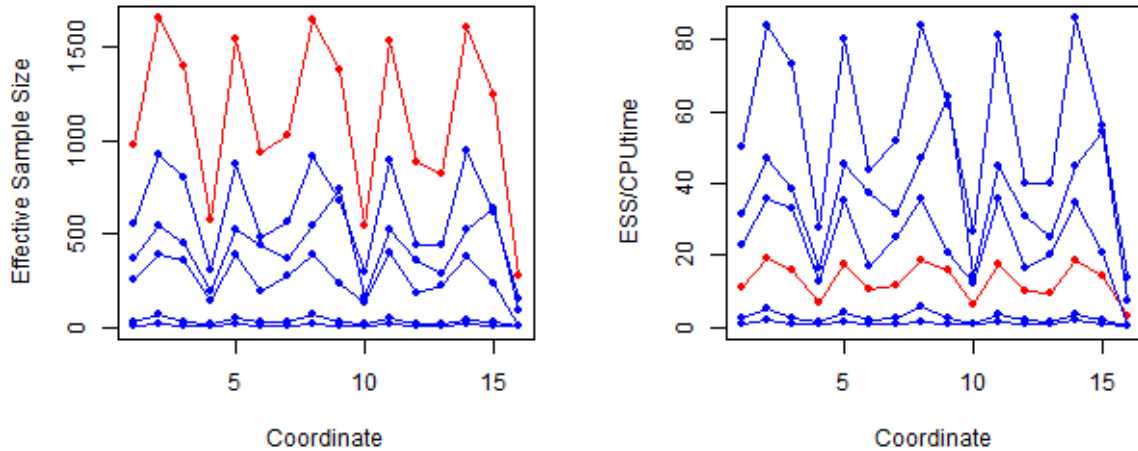
26

Figure 4.6: Standard (non-adaptive) CMTM vs. standard component-wise Metropolis. Orange tree growth data. The red represents the CMTM runs and the blue represents the component-wise Metropolis runs. ESS is calculated after averaging ACT over 100 replicative runs

5 Comparsion of Adaptive Algorithms

The CMTM algorithm has a higher computational cost compared to the standard component-wise Metropolis algorithm. For the same number of iterations, CMTM takes longer CPU time as it has to evaluate $2m - 1$ points under the target distribution at each iteration compared to 1 for the simple Metropolis algorithm. (m is the number of proposals for each iteration.) On the other hand, CMTM does improve efficiency per iteration as implied by the increase in ESS in Table 4.2, Table 4.4, and Table 4.6. In Section 4, the magnitude of ESS difference between the CMTM and the standard component-wise Metropolis, or equivalently the magnitude of efficiency gain, was dependent on the proposal variances of the standard component-wise Metropolis. Moreover, finding the best or close to the best proposal scales for the component-wise Metropolis is not always easy. If we fail to find a 'good' scale for the component-wise Metropolis or even the full-dimensional Metropolis these samplers will do a lot worse than the CMTM algorithm, even after accounting for

27

the additional computation cost with the CMTM. In a practical application one cannot know which one of the CMH's designed along the lines used in the previous section is better than the ACMTM until all the CMH's have been run and compared. Even then there is no guarantee that the most efficient CMH will be surpass the ACMTM. In our applications we can see that the ACMTM represents a solid bet in terms of efficiency even after accounting for CPU time.

Researchers have tried several adaptive Metropolis algorithms to find a 'good' scale for the Metropolis algorithm. We compare a few known adaptive Metropolis algorithms with the ACMTM algorithm.

The adaptive Metropolis algorithm we examine are: Adaptive Metropolis-within-Gibbs (AMwG) algorithm from Roberts and Rosenthal (2009), Single Component Adaptive Metropolis (SCAM) algorithm from Haario et al. (2005), and Adaptive Random Walk Metropolis (ARWM) algorithm from Haario et al. (2001). The AMwG algorithm uses the finding that the optimal acceptance rate for one-dimensional Metropolis algorithm is 0.44 (Gelman et al. (1996), Roberts and Rosenthal (2001)) and adjusts the proposal variance to get the acceptance rate close to 0.44 for each coordinate. The SCAM algorithm finds the empirical variance for each coordinate from the entire history of the Markov chain and tunes the proposal variance for each coordinate based on this. The ARWM algorithm also looks at the entire history of the chain and tunes the proposal's full-dimensional covariance matrix using the empirical covariance matrix calculated from the samples collected up to that iteration.

We look at the three examples from Section 4 and compare how these adaptive algorithms fare against the ACMTM algorithm. For the latter the default $\sigma_{k,j}$'s are 0.1, 0.2, 0.4, 0.8 and 1.6 for every coordinate.

For the AMwG algorithm, the default σ_j is 1 for each coordinate, and it gets adjusted upward if the acceptance rate is higher than 0.44 based on the recent "batch" of 100 iterations and adjusted downward if the acceptance rate is lower than 0.44. The σ_j is adjusted by adding or subtracting $\min(0.05, \sqrt{(h)})$ in log scale based on the h-th "batch" of 100 iterations. One can see how this strategy may not be ideal when the chain alternates between regions in which the proposal must have different scales.

For the SCAM algorithm, the default σ_j for each coordinate is 1, and the chain runs

for $10d$ iterations (10 updates per coordinate) with the default σ_j's. After $10d$ iterations, the proposal variance is tuned based on the empirical variance for each coordinate from the entire history of the chain, multiplied by the scaling parameter 2.38^2 given by Gelman et al. (1996).

For the ARWM, the default Σ is an identity matrix, and after 100 iterations with default Σ, the algorithm starts to tune the proposal covariance matrix by multiplying the empirical covariance matrix with the choice of scaling parameter $2.38^2/d$ (d is the dimension of the Markov chain.) given by Gelman et al. (1996). The results of the comparison of these four adaptive algorithms are shown in Figure 5.1 and Figure 5.2.
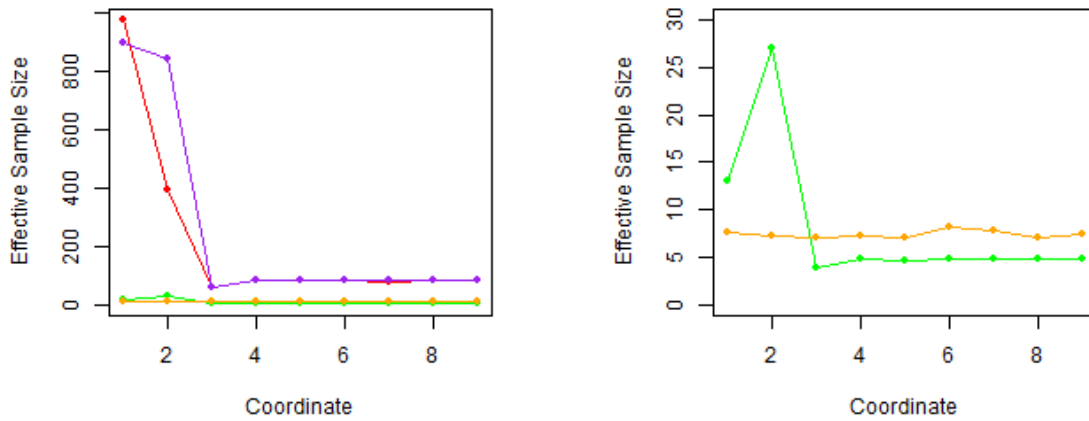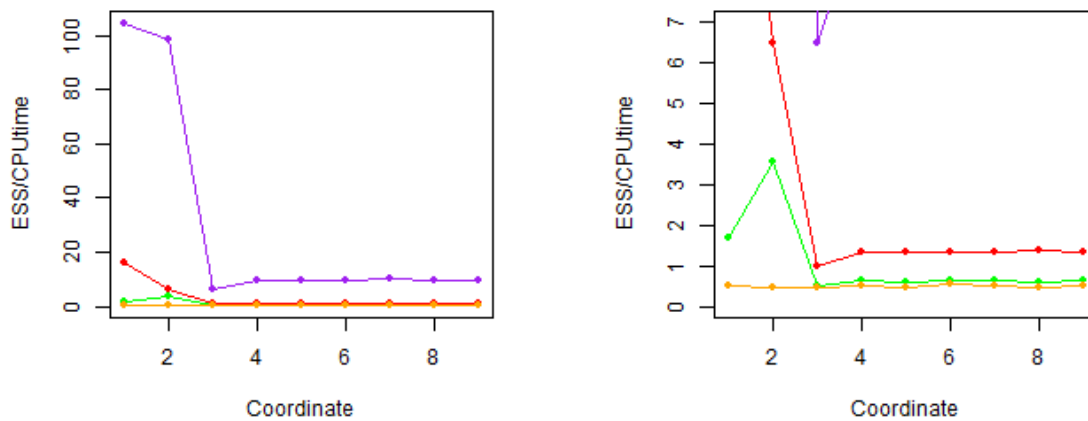
(a) Variance components model
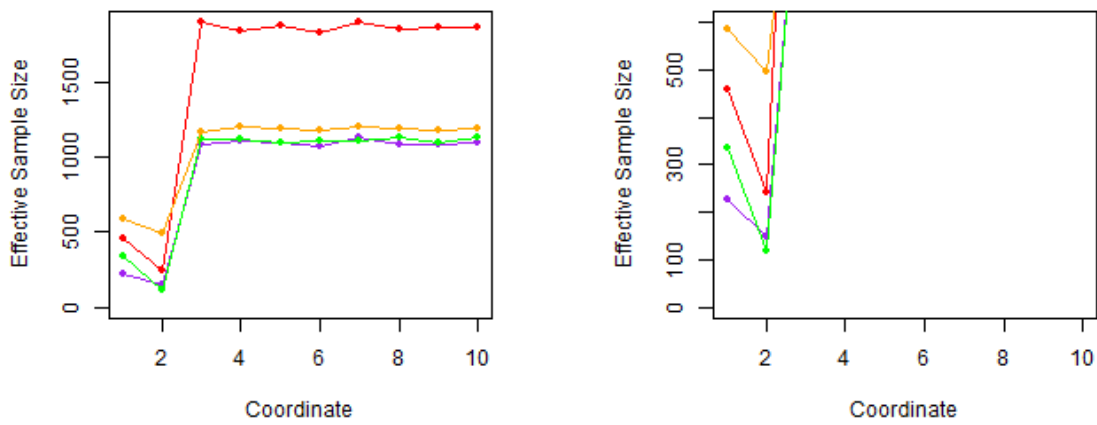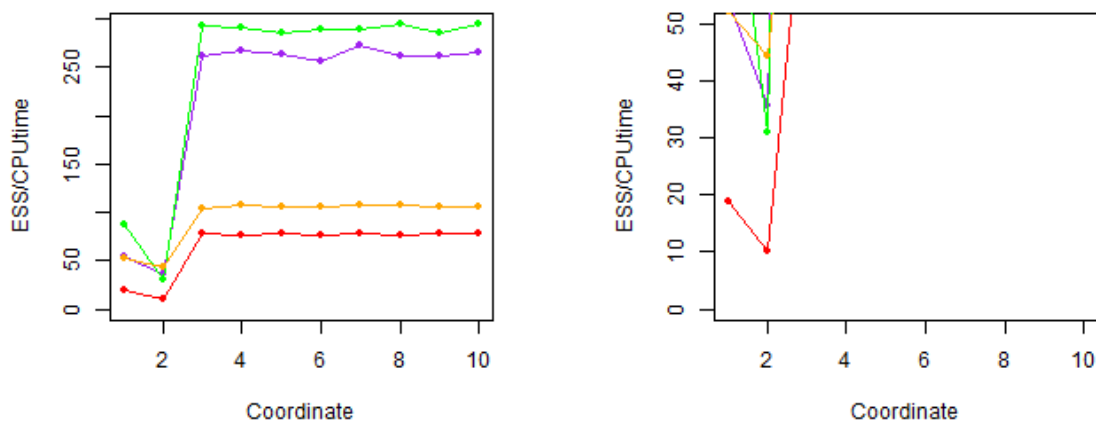

(b) "Banana-shaped" distribution
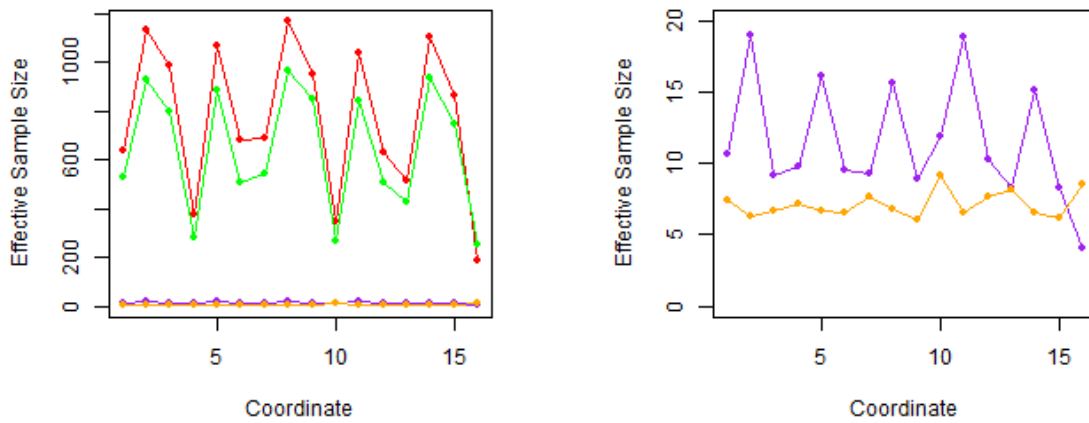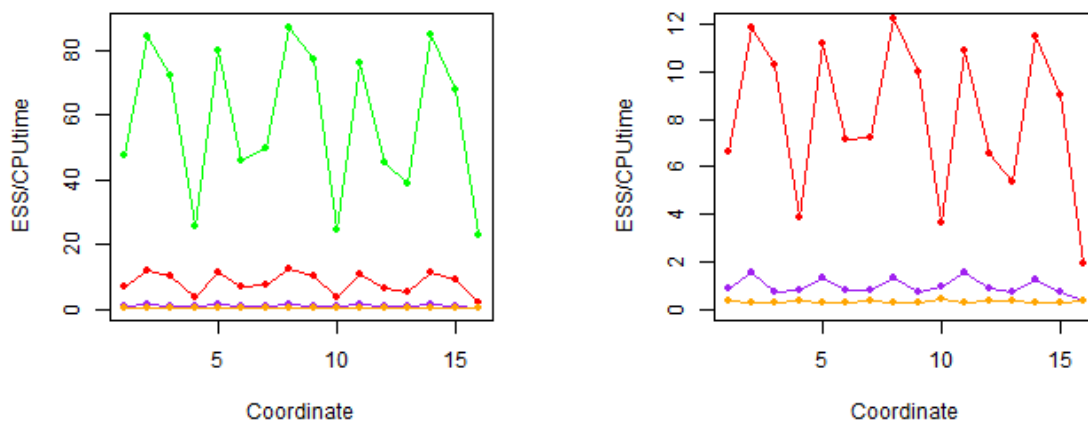

(c) Orange tree growth data

Figure 5.1: Comparison of ESS for different adaptive schemes. The red represents the ACMTM algorithm; the purple represents the AMwG algorithm; the green represents the SCAM algorithm; and the orange represents the ARWM algorithm. In each row, the

(a) Variance components model

(b) "Banana-shaped" distribution

(c) Orange tree growth data

31

Figure 5.2: Comparison of ESS/CPUtime for different adaptive schemes. The red represents the ACMTM algorithm; the purple represents the AMwG algorithm; the green represents the SCAM algorithm; and the orange represents the ARWM algorithm. In

Comparison between samplers is achieved by comparing the worst coordinate-wise ESS and ESS/CPUtime. To be precise, we compare the

$$\min_{1 \leq k \leq d} ESS_k$$

and

$$\min_{1 \leq k \leq d} \frac{ESS_k}{CPUtime}$$

for each algorithm. We see in Figure 5.1 that if we disregard the additional computational cost for the CMTM, adaptive CMTM is the best choice out of 4 adaptive algorithms in every examples we run. To account for the computational cost, we calculate the ESS/CPUtime, and the results are shown in Figure 5.2. Adaptive CMTM is not the best algorithm based on this measure for each individual example. But, if we look at all three examples together, it is hard to say which one is the best algorithm out of all. The SCAM algorithm is the best one for the Orange tree growth data, but it is close to the worst one for the Variance Components Model. The Adaptive Metropolis-within-Gibbs algorithm is the best algorithm for the Variance Components Model, but it clearly did a lot worse than the CMTM algorithm for the Orange tree growth data. The Adaptive Metropolis did worse than the CMTM algorithm in two our of three examples. Once again, while not uniformly dominating, the ACMTM seems to perform robustly in all examples thus minimizing the risk of using an inefficient sampler in a given example.

6   Conclusion and Discussion

It is known that adaptive algorithms can be highly influenced by initial values given to their simulation parameters and by the quality of the chain during initialization period, i.e. the period during which no modifications of the transition kernel take place. ACMTM is no exception, but there certain of its features can be thought of as means towards a more robust behaviour. For instance, the fact that we can start with multiple proposals makes it less likely that all five will be poor choices for a given coordinate. The motivation for ACMTM was given by situations in which the sampler requires very different proposals across coordinates and across regions of the state space. In such situations, traditional

adaptive samplers are known to fail unless special modifications are implemented (Craiu et al., 2009; Bai et al., 2011), but even these tend to underperform when d is high.

The adaption mechanism is very rapid as, once an alarm is triggered, the scales can change in multiple of 2's. The adaption mechanism is also stable since modifications to the kernel occur only if over selection from one of the boundary scale proposals is detected. Thus, if proposal scales are not perfect but good enough, they wouldn't be changing under this adaptive design since it is hard to keep choosing one proposal distribution out of multiple proposal distributions if each of them is useful in a big enough region of the state space. In other words, once the adaption reaches a good level, the proposal distributions would be stable and would not be constantly changing.

A general recommendation has been made by Craiu et al. (2009) to run, at least for a while, a number of adaptive samplers in parallel in order to insulate against a poor exploration of the state space during the initialization period. This idea can be extended easily to the ACMTM, especially in this age in which parallel processing is the norm rather than the exception. The increase in CPU time is the price we pay for the added flexibility of having multiple proposals and the ability to dynamically choose the ones that fit the region of the space so that acceptance rate and mixing rates are improved. And while this tend to attenuate the ACMTM's efficiency dominance, one cannot find among the algorithms we used for comparison in this paper one that is performing better *on average* even after taking CPU time into account. This makes ACMTM a safe choice for multivariate MCMC sampling.

Finally, it is the authors belief that AMCMC samplers will be more used in practice if their motivation is intuitive and their implementation is easy enough. We believe that the ACMTM fulfills these basic criteria and further modifications can be easily implemented once new needs are identified.

Acknowledgement

References

ANDRIEU, C. and ATCHADÉ, Y. F. (2007). On the efficiency of adaptive MCMC algorithms. *Electronic Communications in Probability* **12** 336–349.

ANDRIEU, C. and MOULINES, E. (2006). On the ergodicity properties of some adaptive Markov Chain Monte Carlo algorithms. *The Annals of Applied Probability* **16** 1462–1505.

ANDRIEU, C. and THOMS, J. (2008). A tutorial on adaptive MCMC. *Statist. Comput.* **18** 343–373.

ATCHADÉ, Y. F. and ROSENTHAL., J. S. (2005). On adaptive Markov Chain Monte Carlo algorithms. *Bernoulli* **11** 815–828.

BAI, Y., CRAIU, R. V. and DI NARZO, A. (2011). Divide and C onquer: A mixture-based approach to regional adaptation for MCMC. *J. Comput. Graph. Statist.* **20** 63–79.

BOX, G. E. P. and TIAO, G. C. (1973). *Bayesian inference in statistical analysis.* Addison-Wesely, Reading, MA.

CASARIN, R., CRAIU, R. V. and LEISEN, F. (2013). Interacting multiple try algorithms with different proposal distributions. *Statistics and Computing* **23** 185–200.

CRAIU, R. V., GRAY, L., LATUSZYNSKI, K., MADRAS, N., ROBERTS, G. O. and ROSENTHAL, J. S. (2015). Stability of adversarial markov chains, with an application to adaptive mcmc algorithms. *Annals of Applied Probability* **25** 3592–3623.

CRAIU, R. V. and LEMIEUX, C. (2007). Acceleration of the multiple-try metropolis algorithm using antithetic and stratified sampling. *Statistics and Computing* **17** 109–120.

CRAIU, R. V., ROSENTHAL, J. S. and YANG, C. (2009). Learn from thy neighbor: Parallel-chain adaptive and regional MCMC. *J. Amer. Statist. Assoc.* **104** 1454–1466.

Davies, O. L. (1967). *Statistical methods in research and production.* Oliver & Boyd, Edinburgh and London.

Draper, N. and Smith, H. (1981). *Applied regression analysis.* John Wiley, New York.

Fort, G., Moulines, E. and Priouret, P. (2011). Convergence of adaptive and interacting Markov chain Monte Carlo algorithms. *The Annals of Statistics* **39** 3262–3289.

Gelman, A., Roberts, G. O. and Gilks, W. R. (1996). Efficient Metropolis jumping rules. In *Bayesian Statistics* (J. M. B. et al., ed.), vol. 5. Oxford University Press, 599–607.

Giordani, P. and Kohn, R. (2010). Adaptive independent Metropolis–Hastings by fast estimation of mixtures of normals. *Journal of Computational and Graphical Statistics* **19** 243–259.

Haario, H., Laine, M., Mira, A. and Saksman, E. (2006). DRAM: efficient adaptive MCMC. *Statistics and Computing* **16** 339–354.

Haario, H., Saksman, E. and Tamminen, J. (1999). Adaptive proposal distribution for random walk metropolis algorithm. *Computational Statistics* **14** 375–396.

Haario, H., Saksman, E. and Tamminen, J. (2001). An adaptive Metropolis algorithm. *Bernoulli* **7** 223–242.

Haario, H., Saksman, E. and Tamminen, J. (2005). Componentwise adaptation for high dimensional MCMC. *Computational Statistics* **20** 265–273.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57** 97–109.

Lindstrom, M. J. and Bates, D. M. (1990). Nonlinear mixed effects models for repeated measures data. *Biometrics* **46** 673–687.

Liu, J. S., Liang, F. and Wong, W. H. (2000). The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association* **95** 121–134.

METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H. and TELLER, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics* **21** 1087–1092.

ROBERTS, G. O. and ROSENTHAL, J. S. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical science* **16** 351–367.

ROBERTS, G. O. and ROSENTHAL, J. S. (2004). General state space Markov chains and MCMC algorithms. *Probability Surveys* **1** 20–71.

ROBERTS, G. O. and ROSENTHAL, J. S. (2007). Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of Applied Probability* **44** 458–475.

ROBERTS, G. O. and ROSENTHAL, J. S. (2009). Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics* **18** 349–367.

ROSENTHAL, J. S. and YANG, J. (2016). Ergodicity of discontinuous adaptive MCMC algorithms. Submitted for publication. Available at `http://probability.ca/jeff/ftpdir/adversarial.pdf`.

TURRO, E., BOCHKINA, N., HEIN, A. M. K. and RICHARDSON, S. (2007). BGX: a Bioconductor package for the Bayesian integrated analysis of Affymetrix GeneChips. *BMC bioinformatics* **8** 439–448.

VIHOLA, M. (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing* **22** 997–1008.