Automatically Tuned General-Purpose MCMC via New Adaptive Diagnostics

Jinyoung Yang* and Jeffrey S. Rosenthal†

(Last revised: July 2016.)

Abstract

Adaptive Markov Chain Monte Carlo (MCMC) algorithms attempt to 'learn' from the results of past iterations so the Markov chain can converge quicker. Unfortunately, adaptive MCMC algorithms are no longer Markovian, so their convergence is difficult to guarantee. In this paper, we develop new diagnostics to determine whether the adaption is still improving the convergence. We present an algorithm which automatically stops adapting once it determines further adaption will not increase the convergence speed. Our algorithm allows the computer to tune a 'good' Markov chain through multiple phases of adaption, and then run conventional non-adaptive MCMC. In this way, the efficiency gains of adaptive MCMC can be obtained while still ensuring convergence to the target distribution.

1 Introduction

Markov Chain Monte Carlo (MCMC) is a technique widely used to sample from complex probability distributions, leading to numerous applications and methodological developments and theoretical advances (see e.g. Brooks et al. 2011). It is well-known that some Markov chains work much better than others in terms of convergence speed, asymptotic variance and/or mixing

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

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

speed (see e.g. Rosenthal 2011), leading to questions of how to find more efficient chains.

Adaptive MCMC algorithms attempt to improve the Markov chain 'on the fly', using information from past iterations of the chain. This can significantly improve efficiency in practice (e.g. Haario et al. 2001, 2006; Roberts and Rosenthal 2009; Giordani and Kohn 2010; Vihola 2012; Turro et al. 2007). Unfortunately, most adaptive MCMC algorithms are no longer Markovian, so convergence of the algorithm to the target distribution is much more difficult to establish and can sometimes fail (see e.g. Rosenthal 2004). This question has been investigated extensively, and researchers have proved the ergodicity of adaptive MCMC algorithms under various conditions (e.g. Haario et al. 2001, 2006; Giordani and Kohn 2010; Vihola 2012; Atchadé and Rosenthal. 2005; Andrieu and Moulines 2006; Andrieu and Atchadé 2007; Roberts and Rosenthal 2007; Fort et al. 2011). However, these results all require assumptions such as "Containment" or "simultaneous polynomial drift conditions" which are virtually impossible to verify directly in practical applications, and there aren't many widely applicable convergence results with easily-checkable assumptions. This means that when using adaptive MCMC in practice, there are usually no guarantees of even asymptotic convergence, and the user must simply "hope" that aberrant behaviour such as that exhibited in Rosenthal (2004) does not arise.

In this paper, we present an algorithm which uses new adaptive diagnostics to determine when 'enough' adaption has already been done, i.e. when further adaption is not likely to lead to significant further improvements in efficiency. At this point, the adaption ceases, and the algorithm runs an ordinary non-adaptive MCMC algorithm for which convergence is guaranteed. In this way, our algorithm achieves the efficiency gains of adaptive MCMC, while avoiding the theoretical obstacles of typical adaptive MCMC algorithms which continue to adapt indefinitely. For definitiveness, we focus here on improving the proposal distribution for the Metropolis-Hastings algorithm (Metropolis et al. 1953; Hastings 1970), though similar ideas could also be applied in other adaptive MCMC contexts. We have developed a companion software package 'atmcmc' (Yang 2014), written in the R computer language (R Core Team 2014), to implement the algorithm introduced herein.

We note that various other software packages for adaptive MCMC are already available. One example is 'AMCMC' (Rosenthal 2007a,b) which employs an Adaptive Metropolis-within-Gibbs algorithm from Roberts and Rosenthal (2009). Another is the package 'Grapham' (Vihola 2010a,b) for Adaptive Metropolis-within-Gibbs for graphical models with arbitrary block decompositions. Another example is 'adapMCMC' by Scheidegger (2012),

which is based on the adaptive MCMC algorithm proposed by Vihola (2012) which tries to learn the shape of the target distribution while coercing the acceptance rate at the same time. A fourth example is 'FME' by Soetaert and Petzoldt (2014), based on Soetaert and Petzoldt (2010), whose function 'modMCMC' is an implementation of the Delayed Rejection Adaptive Metropolis (DRAM) method of Haario et al. (2006). These packages are all promising and useful, but they all involve infinite adaption, and thus require careful conditions to ensure convergence – in contrast to the algorithm herein which stops adapting once a 'good' proposal distribution is obtained and thus must converge by traditional Markov chain properties.

Section 2 of this paper explains the idea behind the our algorithm. Section 3 provides some background information for our algorithm. Section 4 presents the details of the algorithm including all of the phases involved. Section 5 applies the algorithm to a number of MCMC examples. Section 6 provides some concluding comments.

2 Approach

It is well-known that a discrete-time Markov chain on a general state space converges eventually to its stationary distribution if it is ϕ-irreducible and aperiodic (e.g. Roberts and Rosenthal 2004). In practice, with time and resource constraints, we can't just rely on this eventual convergence and run a Markov chain to infinity. One concern is the time required to reach convergence.

Some Markov chains take unreasonably long time to reach convergence, especially in high dimension. We can improve the efficiency of a MCMC algorithm considerably by adapting. Since adaption destroys the Markov property of a Markov chain, the convergence of an adaptive MCMC algorithm has to be proven case-by-case. We want a more general algorithm so the user doesn't have to prove the convergence of an adaptive MCMC algorithm every time he tries different example, which sometimes is challenging. Thus, our goal here to make an algorithm which stops adapting once it obtains a 'good' proposal distribution, or, in other words, once the chain is tuned to improve the speed of the convergence.

Our focus in this paper is to find a way to approximate the point where the adaption is not adding much value to the chain, thus allowing us to stop the adaptive algorithm. Also, to efficiently utilize the adaption which takes the past and current values of the chain to mimic the target distribution, we decide to take some pre-steps before the final adaption. The actual methods of adaption are not the main impotance here. You can change the adaption

Figure 1: Algorithm flowchart

methods.

Since we are interested in a finite adaption here, we have to decide when to stop the adaption. To verify if the adaption is indeed improving the convergence speed of a Markov chain, we calculate the squared jumping distance, $(X_n - X_{n-1})^2$, for each iteration n. We want a Markov chain to explore the sample space of the target distribution quickly. In other words, we want a Markov chain to have a high mixing speed. The average squared jumping distances is one measure to show how well the chain is mixing by averaging the magnitudes of the movements from one state to the next. If we see a general increase in the squared jumping distances as the adaptive algorithm runs, we presume the mixing of the chain is getting better. Since we have to account for random fluctuations in squared jumping distances, we calculate the average of squared jumping distances for a fixed number of iterations and see if they are increasing. The more the chain moves for each iteration, the faster the chain will converge because it implies that the chain moves well throughout the entire state space, reducing the time to explore the full state space. Therefore, once the average squared jumping distance stops increasing, we assume further adaption would not significantly improve the mixing of the chain. We then stop the adaption, take the proposal distribution we get from the adaption, and run a standard MCMC algorithm for which all the properties and theories of a true Markov chain apply. We will use the Gelman-Rubin convergence diagnostic (Gelman and Rubin 1992; Brooks and Gelman 1998), which is explained briefly in Section 3, to check for the convergence of the chain when we run a standard MCMC algorithm.

However, in the collection of past values we use to modify the chain, we don't want to include too many values which have low probabilities of occurring in the target distribution. Thus, we want to discard a burn-in period until the chain reaches the mode of target distribution. We call this step to find the mode of the target distribution Transient phase. Only after then, we want to collect the values from the chain to fine-tune the chain.

At the same time we don't want our chain to take forever to reach the mode of the target distribution due to the bad proposal distribution for a specific target distribution. Thus, before we let the chain to find the mode of the target distribution, or before Transient phase, we make the algorithm quickly and roughly adjust the chain using some adaptive method. We call this 1^{st} adaption phase.

Thus, our method consists of 1^{st} adaption phase, Transient phase, 2^{nd} adaption phase, and the sampling phase which we run the standard MCMC algorithm while checking for the convergence with Gelman-Rubin diagnostic. Figure 1 summarizes the key idea of our method.

3 Background

For Metropolis-Hastings algorithms, we can consider the acceptance rate, i.e. the rate at which a new proposed value at each iteration is accepted. We don't want the acceptance rate to be too high, which implies each proposed move is too small so the chain moves slowly throughout the whole state space. At the same time, we don't want the acceptance rate to be too low, either, since then the chain tends to get stuck at one value and hardly moves from it. In Roberts et al. (1997) it was shown that, for the symmetric random walk Metropolis algorithm with proposal distribution $N(0, \sigma^2 I_d)$, the optimal asymptotic acceptance rate is 0.234 as $d \to \infty$ if the target density function has the form $\pi(x) = \prod_{i=1}^{d} f(x_i)$ on $\mathbb{R}^d$. In Roberts and Rosenthal (2001), it was proved that the optimal acceptance rate is still 0.234 as $d \to \infty$ if the target density has the form of $\pi(x) = \prod_{i=1}^{d} C_i f(C_i x_i)$ where the $\{C_i\}$ are selected as i.i.d. positive random variables from some fixed probability distribution. Numerical studies also showed the optimal acceptance rate 0.234 is quite robust as it holds if d is as low as 5 (Gelman et al. 1996; Roberts and Rosenthal 2001) or the target density doesn't exactly have the form required to prove the theorem (Roberts and Rosenthal 2001). It was also found by numerical studies that the optimal asymptotic acceptance rate of one dimensional MCMC algorithm is 0.44 not 0.234 (Gelman et al. 1996; Roberts and Rosenthal 2001). Note that Gelman et al. (1996) and Roberts and Rosenthal (2001) demonstrated that it is not worth tuning for the *exact* optimal acceptance rate of 0.234, since if the average acceptance rate is between 0.15 and 0.5 then the Markov chain is at least 80% efficient compared to the optimal.

Another way to improve the convergence speed of a Markov chain is, for the symmetric random walk Metropolis algorithm with proposal distribution $N(0, \Sigma_p)$, to get the Σ_p proportional to the estimated covariance matrix of target distribution. This strategy was first suggested in Haario et al. (2001) and justified in Roberts and Rosenthal (2001). Intuitively, this strategy seems promising since the more the proposal distribution is similar to the target distribution, the more likely the chain would propose a value that the target distribution would propose at each iteration given the current value X_n, and it wouldn't need as long a run of the Markov chain to converge to the target distribution. It is shown in Roberts et al. (1997) that if the target covariance matrix is multiplied by $2.38^2/d$ to obtain the covariance matrix Σ_p, then as $d \to \infty$, the proposal distribution $N(0, \Sigma_p)$ gives optimal convergence (again with acceptance rate 0.234) among all Gaussian proposal distributions.

Of course, in practice we don't know the target covariance matrix. However, we can use the empirical covariance matrix calculated from past chain values instead. That is, after some burn-in period, we use past values of

the chain to estimate the covariance of the proposal distribution, and then multiply this empirically estimated matrix by $2.38^2/d$ and use that as our proposal covariance matrix (Haario et al. 2001; Roberts and Rosenthal 2009). As the chain runs for a long time, the collection of the past values from the chain gets closer to a sample from the true target distribution, and therefore the estimated covariance get closer to the true target covariance.

One last problem is that even when running a standard MCMC algorithm, we don't know when is good time to stop the chain. In other words, we don't know when the chain convergence is achieved and we can take the values generated from the chain as a sample from the target distribution. Gelman and Rubin (1992) proposed a method to detect convergence of a Markov chain. They run several replicative Markov chains from a overdispersed starting distribution. They discard values from first half of the chains and take the second half as a sample and diagnose convergence. They assume convergence is achieved if $\hat{R}_c := (\hat{V}/W)$(c.f.) is close to 1, where

$$\hat{V} \;=\; \frac{n-1}{n}\,W + \frac{B}{n} + \frac{B}{nm}\,,$$

W is the average sample variance from each replicative chain, and B/n is the variance of sample mean from each chain. Here n is the sample size of each chain, m is the number of replicative chains, and the correction factor (c.f.) accounts for the difference in variances between the Student's t distribution and the normal distribution. If $\hat{R}_c$ is close to 1, this implies the variance of the sample mean from each chain is almost negligible, which hopefully indicates convergence to stationarity. The idea behind the Gelman-Rubin diagnostic is that if chains started from all over the state space each give us samples with similar distributions, then each chain must have reached the same distribution, the target. Therefore, starting the replicative chains from an 'overdispersed' starting distribution is crucial.

One pitfall of the Gelman-Rubin diagnostic $\hat{R}_c$ is that it implicitly assumes normality of the target distribution, in the sense that they only monitor means and variances to compare distributions of the replicative chains. To overcome this shortcoming, there is another paper by Brooks and Gelman (1998) extending the Gelman-Rubin diagnostic. They suggest the use of $(1-\alpha) * 100\%$ confidence intervals, and define $\hat{R}_{interval}$ to be the ratio of the length of the total-sequence interval (found when all points from all replicative chains are thrown together as one sample) divided by the average length of the intervals from each of the replicative chains. They assume convergence is achieved if $\hat{R}_{interval}$ (instead of $\hat{R}_c$) is close to 1.

We next present the details of our algorithm, which combines all of the above ideas together to automatically tune MCMC, and hopefully achieve

efficient convergence without sacrificing MCMC's theoretical guarantees.

4 Technical Details

The major breakdown of our algorithm is as follows. It consists of a number of distinct phases. We start with an Adaptive Metropolis-within-Gibbs algorithm (Roberts and Rosenthal 2009) to get a rough idea of scaling for each coordinate of the Markov chain ("1^{st} adaptive phase"). We continue this phase until we get an acceptance rate of every coordinate in the neighbourhood of 0.44 (the optimal acceptance rate for one dimensional Markov chain). Note that we only need a very *rough* scaling estimate, so the neighbourhood range can be quite large. Then, we run a fixed (non-adaptive) Metropolis-within-Gibbs algorithm with this scaling ("transient phase"), and diagnose whether the chain has reached the *mode* of the target distribution. We do this by fitting a regression line to see if the chain values are trending, and continue until the regression signals that the chain becomes flat in every coordinate. Next, we employ an Adaptive Metropolis algorithm (Haario et al. 2001; Roberts and Rosenthal 2009) which adaptively updates the full-dimensional proposal covariance matrix Σ_p ("2^{nd} adaptive phase"). As mentioned earlier, the increase in the averaged squared jumping distance is used as a measure of the adaption improving the chain. We continue the Adaptive Metropolis algorithm until the average squared jumping distance stops increasing. At this point, we run a conventional Metropolis algorithm ("sampling phase"), and apply a Gelman-Rubin convergence diagnostic to divide the remaining run into two halves, one for burn-in and one for actual sampling from the target distribution.

We now describe the details of these various phases, one by one. The phases do involve various choices of approach and parameters, but they all appear to work well in practice and to be robust to different target distributions. Thus, the algorithm described below can be implemented directly (using our companion software package 'atmcmc' (Yang 2014)), without requiring any additional adjusting or tweaking by the user. And, as mentioned, since our algorithm uses only a finite amount of adaption followed by a conventional MCMC algorithm, asymptotic convergence to the target distribution is automatically guaranteed, without requiring the sorts of specialised arguments which are needed for most adaptive MCMC algorithms.

4.1 1^{st} Adaption Phase

To begin, we start with the Adaptive-Metropolis-within Gibbs algorithm introduced in Roberts and Rosenthal (2009) to get a rough idea on what is 'good' scale for each coordinate of a Markov chain. We call this step 1^{st} adaption phase. Let X_0, X_1, X_2, ..., be a Markov chain process and Y be a new value proposed by a certain proposal distribution at each iteration. Given a current value of a Markov chain, X_n, Y is proposed by substituting $X_{n,j}$ with Y_j drawn by $Y_j \sim N(X_{n,j}, \sigma_j^2)$ where $X_{n,j}$ and Y_j are the j^{th} coordinate of X_n and Y, respectively, and σ_j^2 is the variance of the proposal distribution for the j^{th} coordinate. Then Y is either accepted ($X_{n+1} = Y$) or rejected ($X_{n+1} = X_n$) by the Metropolis rule. In short,

$$\begin{cases} X_{n+1} = Y & if \quad U < min(1, \pi(y)/\pi(x)) \\ X_{n+1} = X_n & if \quad U \geq min(1, \pi(y)/\pi(x)) \end{cases}$$

where $U \sim U(0,1)$ and $\pi(\cdot)$ is the target density function. This is done for every coordinate j, sequentially. Note that when $\pi_u(x) = 0$, we always accept the new proposal y. Thus, we don't want $\pi_u(x) = 0$ situation because we accept the new proposal y even if $\pi_u(y) = 0$, making $\pi_u(x) = 0$ for next iteration. In this case, there is a chance a Markov chain end up drifting to the 'wrong' direction. To prevent this, we will stop the whole MCMC run if a certain number of iterations has $\pi_u(x) = 0$ consecutively, and this rule will be applied not just for this phase but for any other phase, whether the target distribution is unimodal or not, if the Metropolis rule is used to accept or reject.

To begin, a single Markov chain is run from the randomly chosen initial point X_0. With adaption, for each coordinate j of a Markov chain, we try to achieve the acceptance rate of 0.44, which is known to be an approximately optimal acceptance rate for one dimensional Markov chain (Roberts and Rosenthal 2001). As in Roberts and Rosenthal (2009), we change the variance of the proposal distribution to alter the acceptance rate of a Markov chain since a proposal distribution with a larger variance tend to propose larger values, which would get rejected more often than values proposed by a proposal distribution with a smaller variance, and vice versa. Thus, for every 100 iteration, we calculate the acceptance rate of the past 100 iterations for each coordinate j, and we add $\epsilon = 0.05$ to $\log(\sigma_j)$ if the acceptance rate is higher than 0.44, and subtract ϵ from $\log(\sigma_j)$ if the acceptance rate is lower than 0.44. We do this until the acceptance rate for every coordinate of the Markov chain falls between 0.28 and 0.60. If we get the acceptance rate for every coordinate in between 0.28 and 0.60, we run 100 more iterations with

same σ_j's, which have made the acceptance rates to fall between 0.28 and 0.60, and monitor the acceptance rate for the past 200 iterations. If at least one acceptance rate from the coordinates falls outside of 0.28 and 0.60, then we adjust $\log(\sigma_j)$ for every 200 iterations until the acceptance rate for every coordinate comes between 0.28 and 0.60. Once we have the acceptance rate for every coordinate fall between 0.28 and 0.60, we run 200 more iterations with σ_j's unchanged and monitor the acceptance rate for past 400 iterations. If at least one acceptance rate falls outside of 0.28 and 0.60, we adjust $\log(\sigma_j)$ for every 400 iterations until the acceptance rate for every coordinate fall between 0.28 and 0.60. If the acceptance rate for every coordinate from past 400 iterations falls between 0.28 and 0.60, we stop the chain and save σ_j for every coordinate.

Note that here the acceptance rate is a continuous function of proposal variance. If we want to increase (decrease) the acceptance rate by a bit, we have to decrease (increase) the proposal variance by a bit accordingly. We can always get the acceptance rate fall into some desired range (easier if the range is big) as long as the target density and the proposal density is positive everywhere in the state space and the shift in the proposal variance is not too big at each adjustment.

4.2 Transient Phase

Next, we try to find if there is any transient phase for the Markov chain since we don't want to start final adaption when the values generated, which will be used for the adaption, are far from where the major mass of the target distribution is. We want a burn-in phase to discard the part of the chain, which mostly consists of values in the low probability zones under the target distribution. We call this transient phase.

We employ a standard Metropolis-within-Gibbs algorithm with proposals for each coordinate drawn from $Y_j \sim N(X_n, \sigma_j^2)$ with σ_j^2 determined by the 1^{st} adaption phase. To check if the chain is moving towards the mode of target distribution, for every 200 iteration, the values generated for each coordinate j of X are averaged, and with 5 different averages for each coordinate j, a linear model is fitted to see if there is any trend in the j^{th} coordinate of the chain. The specific values 200 and 5 are somewhat arbitrary choices, but we have found that they work well when we run the algorithm. The user has flexibility to pick some other numbers, as long as he/she makes sure the algorithm has enough points (for example, 5) to run a regression but also at the same time two numbers (for example, 200 and 5) are not too big so it doesn't take too long to test whether the chain is trending or not.

We use a regression method to make sure the chain values are moving to

only one direction, neither increasing nor deceasing. If a regression method confirms that the chain values show a linear trend, we presume that the chain is still moving to a local mode. The p-value for the slope coefficient is used to determine whether there is any linear trend. If p-value for every coordinate is greater than 0.1, the chain gets stopped and this phase ends. We have found that a p-value of 0.1 is a reasonable cutoff for our purpose. The p-value below 0.1 is the point where people start to talk about any sign of significance in statistics, although many prefer a lot lower p-value than 0.1 to actually claim the significance in practice. Here, we are detecting non-significance, so 0.1 seems to be a convenient threshold which appears to be robust in practice. Plus, if for some reason, the p-value cutoff misses a trending chain (which we believe has a very low chance of happening), we still have the last phase which will run a standard MCMC algorithm. Therefore, at a minimum, the algorithm will converge to the target distribution even though we might lose some efficiency.

4.3 2^{nd} Adaption Phase

Here, we slightly modify the Adaptive Metropolis algorithm introduced in Haario et al. (2001) and Roberts and Rosenthal (2009)) to find the proposal distribution that has a similar covariance structure with the target distribution. This phase is called 2^{nd} adaption phase. The proposal, Y, is drawn from $Y \sim N(X_n, c\Sigma_n)$, and again the accept/reject is by the Metropolis rule. Σ_n is found by calculating the covariance matrix of all past values generated by the chain from the point the trending stops in the transient phase to X_{n-1}, and a constant we multiply to Σ_n is $c = 2.38^2/d$. After 200 iterations in this phase, if the acceptance rate is too low (less than 0.02), then we reduce c by a factor of d, to $2.38^2/d^2$, to make the scale of the proposal distribution smaller thus increasing the acceptance rate, and start this phase again with the last value from the transient phase as the starting value. We want to stop the adaption when further adaption does not improve the chain. To check whether the adaption is improving the chain or not, for every 200 iteration, we calculate the average squared jumping distance for each coordinate j, and again with 5 different averages for each coordinate j, we fit a linear model to see if there is any trend in squared jumping distance for each coordinate. If average squared jumping distance stops to increase, we presume the mixing of the chain stops to improve, and we stop this phase. Thus, we stop this phase when, for every coordinate, the p-value for the slope coefficient of the regression is greater than 0.1.

4.4 Sampling Phase

Finally, we apply the symmetric random walk Metropolis algorithm with proposal Y drawn from $Y \sim N(X_n, \Sigma_p)$. Σ_p is the last Σ_n obtained from the 2^{nd} adaption phase multiplied by the constant c. We use both the Gelman-Rubin diagnostic $\hat{R}_c$ and the extension $\hat{R}_{interval}$ to monitor convergence of the Markov chain, since $\hat{R}_{interval}$ doesn't require the normality assumption that $\hat{R}_c$ does. To apply these diagnostics, we run $k = 10$ replicative chains simultaneously. Besides the last value of the 2^{nd} adaption phase used as the initial point of one replicative chain, the initial points for each of $k-1$ replicative chains are drawn from $U(d_j - (e_j - d_j)/4, e_j + (e_j - d_j)/4)$ for each coordinate where d_j and e_j are the minimum and maximum value, respectively, for each coordinate j of the chain found from the point the trending stops in the transient phase to the end of the 2^{nd} adaption phase. After some burn-in period, we discard the values from the first half of this phase, and calculate $\hat{R}$s with the values from the second half of the phase. When both $\hat{R}$s for every coordinate are close to 1, we stop the chain and take the values from the second half of this phase as our sample from the target distribution. For our runs of the algorithm, including runs on the examples described in Section 5, we stopped the algorithm if $\hat{R}_c$ and $\hat{R}_{interval}$ falls in between 0.9 and 1.1. The user of our algorithm can make the cutoff values more strict if he/she desires. We call this phase sampling phase. Note that we throw values from all replicative chains together in our sample.

Note that most of numerical values used in the algorithm can be changed by the user in the R package 'atmcmc'. The choice of these numerical values does affect the performance of MCMC, and the numbers specified in this section are the default values built in the R package. For details, see the reference manual of the R package 'atmcmc' (Yang 2014).

4.5 Extension for Strongly Multimodal Targets

Suppose a target distribution has multiple local modes, and there is at least one mode that is strongly separated from any of other modes. In other words, the region between this mode and any other modes is at low density. Since accepting a proposal value that is at much lower density region compared to the current state is a low probability event, unless a direct proposal of a point in another mode occurs from time to time, there is a chance a Markov chain gets stuck in that mode. We will call a target distribution like this is 'strongly multimodal'.

With a 'strongly multimodal' target distribution, we run multiple Markov chains with different initial points drawn randomly from $U(a, b)$, $a, b \in \mathbb{R}^d$, for

the 1^{st} adaption phase and transient phase as the chains find different modes. It is important to run enough number of chains with widespread initial points to find all modes in the target distribution. (Note that we do *not* assume that we know the location of the modes in advance, nor even the number of modes.) Once all these different chains find different modes, we calculate the mean and standard deviation of each Markov chain from the segment off the transient phase where the chain is not trending, and we use these means and standard deviations to determine if the chains found different modes. For any two chains, for at least one coordinate, if the difference in two chain means is greater than at least one of two chain standard deviations, we consider two chains reached different modes. With this selection process, we only collect the chains with different modes.

Then we go into the 2^{nd} adaption phase with the chains we are left with after the selection process. Once each of these chains stops in the 2^{nd} adaption phase, we get different Σ_i for each chain. Again, with the values generated from the 2^{nd} adaption phase, we check if each Markov chain still stays at the different mode to each other.

After this, we start with some initial point, and run a MCMC algorithm that we are about to describe. The initial points are the last point of the 2^{nd} adaption phase from each chain and some randomly drawn points from $\sum_{i=1}^{r} U(a_i, b_i)/r$, where i indexes for each of r chains with different modes and (a_i, b_i) are decided based on the values generated from the 2^{nd} adaption phase. We don't use values from the flat part of the transient phase here since there is a chance that two or more chains with different modes merge during the 2^{nd} adaption phase, giving us a different number of 'unique' chains after the 2^{nd} adaption phase than right after transient phase. Keep in mind that we want the overdispersed starting distribution for the Gelman-Rubin diagnostic. As the Markov chain run, we need to evaluate which mode the current value $X_n = x$ is the closest to. The rule for determining this is

$$\text{mode}(x) = \underset{i}{\text{argmin}}\ D_i$$

where $D_i = \max_j d_{ij}$ with $d_{ij} = |x_j - m_{ij}|/\sigma_{ij}$. Here i indexes the multiple chains with different modes, and j indexes for the coordinate of X. Also m_{ij} and σ_{ij} are the mean and standard deviation of chain i, $i \in \{1, ..., r\}$, and coordinate j, $j \in \{1, ..., s\}$, calculated from the values generated from 2^{nd} adaption phase. Again, we don't include values from the flat part of the transient phase when calculating the mean and standard deviation of each chain i.

Suppose $\text{mode}(X_n) = k$. Then we update the Markov chain in the sampling phase as follows.

1. With a $1-\alpha$ probability, we propose Y by

$$Y|X_n \sim N(X_n, c\Sigma_k)$$

where Σ_k is the variance obtained from the 2^{nd} adaption phase for the chain k. We shall write $q_{1k}(x,y)$ for the probability of proposing y given x using this rule. That is,

$$q_{1k}(x,y) = \begin{cases} \frac{1}{\sqrt{(2\pi)^s |c\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2c}(y-x)^T {\boldsymbol{\Sigma}_k}^{-1}(y-x)\right), & \text{if mode}(x) = \text{mode}(y) = k \\ 0, & \text{otherwise} \end{cases}$$

Note that here $q_{1k}(x,y) = q_{1k}(y,x)$, since q_{1k} depends only on $|y-x|$. y is rejected if mode$(x) \neq$ mode(y). Otherwise, y is accepted or rejected based on the Metropolis rule.

2. With a α probability, we propose Y by proposing each coordinate of Y as

$$Y_j|X_{n,j} = \frac{\sigma_{lj}}{\sigma_{kj}}(X_{n,j} - m_{kj}) + m_{lj}$$

Here l is a random draw from all different chain i's excluding chain k, where the probability of drawing some chain l is uniform. Thus, σ_{lj} is the univariate standard deviation of the chain l and the coordinate j, from the sample obtained for the $2^n d$ adaption phase, and σ_{kj} is the univariate standard deviation of the chain k and the coordinate j. In other words, if there are r different chains with all different modes,

$$q_2(x,y) = \begin{cases} \frac{1}{r-1}, & \text{if } y_j = \frac{\sigma_{lj}}{\sigma_{kj}}(x_j - m_{kj}) + m_{lj},\ j = 1,...,s,\ l = 1,...,r,\ l \neq k \\ 0, & \text{otherwise} \end{cases}$$

where $q_2(x,y)$ is the probability of suggesting y given x using this rule. Note that for all (x,y), $q_2(x,y) = q_2(y,x)$. y is rejected if mode$(x) =$ mode(y). Otherwise, y is accepted or rejected based on the Metropolis rule.

Thus, if $P(x,\cdot)$ is the transition probability for x and mode$(x) =$ mode$(y) = k$, then

$$\begin{aligned} \pi(dx)P(x,dy) &= [s^{-1}\pi_u(x)dx][q(x,y)a(x,y)dy] \\ &= [s^{-1}\pi_u(x)][(1-\alpha)q_{1k}(x,y)][\min(1, c^{-1}\pi_u(y)/c^{-1}\pi_u(x))]dxdy \\ &= s^{-1}[(1-\alpha)q_{1k}(x,y)][\min(\pi_u(x), \pi_u(y))]dxdy \\ &= s^{-1}[(1-\alpha)q_{1k}(y,x)][\min(\pi_u(y), \pi_u(x))]dxdy \\ &= [s^{-1}\pi_u(y)dy][q(y,x)a(y,x)dx] \\ &= \pi(dy)P(y,dx) \end{aligned}$$

for some normalizing constant s for $\pi_u(\cdot)$, an unnormalized density function of π. $q(x, y)$ is the probability of suggesting y given x for the Markov chain of interest, and $a(x, y)$ is the probability of accepting y given x.

If mode$(x) \neq$ mode(y), then

$$\begin{aligned}
\pi(dx)P(x, dy) &= [s^{-1}\pi_u(x)dx][q(x, y)a(x, y)dy] \\
&= [s^{-1}\pi_u(x)][\alpha q_2(x, y)][\min(1, c^{-1}\pi_u(y)/c^{-1}\pi_u(x))]dxdy \\
&= s^{-1}[\alpha q_2(x, y)](\min(\pi_u(x), \pi_u(y)))dxdy \\
&= s^{-1}[\alpha q_2(y, x)](\min(\pi_u(y), \pi_u(x)))dxdy \\
&= [s^{-1}[\pi_u(y)dy][q(y, x)a(y, x)dx] \\
&= \pi(dy)P(y, dx)\,.
\end{aligned}$$

Recall that when $\pi_u(x) = 0$, we always accept the new proposal y. If $\pi_u(x) = 0$ and $\pi_u(y) \neq 0$, then $\pi(dx)P(x, dy) = \pi(dy)P(y, dx) = 0$ since $P(y, dx) = 0$ and vice versa. If $\pi_u(x) = 0$ and $\pi_u(y) = 0$, then it is trivial.

To sum up, the Markov chain we construct above is reversible with respect to $\pi(\cdot)$.

In our runs of the proposed scheme, we used $\alpha = 0.05$. α cannot be too big since it is not desired the the mode-to-mode jump happening too frequently, and it cannot be too small resulting mode-to mode-jump happening only few times, which reduces the effectiveness of our algorithm for the 'strongly multimodal' targets. The user of our scheme (and the user of the R package 'atmcmc') has some freedom to choose α as long as it is not too big or too small due to the reasons just described.

Convergence of the Markov chain is still diagnosed the same way explained in main algorithm, and, again, the final sample obtained is the collection of all values from the second half of all replicative chains created for Gelman-Rubin diagnostics. We assume throwing all points from all replicative chains together will not distort the end result as we believe the convergence test by Gelman-Rubin can be only passed when the mixing of each replicative Markov chain is 'good'.

5 Applications

In this section, we present a number of different applications of our algorithm. The plots of the sampling phases come from one replicative chain out of 10, which uses the last value of the 2^{nd} adaption phase as the starting point for the sampling phase. For ease of identification, each phase is coloured differently in the trace plots: purple for the 1^{st} adaption phase, orange for

the transient phase, red for the 2^{nd} adaption phase, and blue and green for the first and second halves of the sampling phase. Thus, the green segment in each trace plot is what we take as our actual sample, and is also what we use to calculate the displayed acceptance rates in the tables. Also note that in the trace plots, the iteration number for the 1^{st} adaption and transient phases is counted coordinate-by-coordinate as the trace plots are presented coordinate-by-coordinate. More clearly, the first update of coordinate 1 is labeled iteration 1 in the trace plot for coordinate 1, and the first update of the coordinate 2 (second update for the whole algorithm) is labeled iteration 1 in the trace plot for the coordinate 2 and so on.

In each example, two tables are given. First one displays the 10 estimates of parameters obtained from 10 independent runs of the full algorithm in Section 4. The second one displays the summary statistics of these 10 estimates: mean, standard deviation, minimum and maximum. The second tables also contain the true values of the parameters being estimated or estimates of the parameters using by some other method, so the reader can compare and see how the algorithm presented in this paper has performed.

Note that all the starting points for the 1^{st} adaption phases are arbitrary chosen as $0.1 * \mathbf{1}$, except in the 'strongly multimodal' case. For the examples presented in Section 5.1, 5.2, 5.3, and 5.4, we did not use the scheme for 'strongly multimodal targets' from Section 4.5. Only for the example in Section 5.5 the scheme from Section 4.5 was applied.

5.1 Multivariate Normal Distribution

First, we take a 9-dimensional multivariate normal distribution as our target distribution. Each component of target mean, μ, was randomly drawn from $N(0, 1000^2)$ and target variance were constructed by $\Sigma\Sigma^t$ where each component of Σ was drawn from $N(0, 20^2)$. The target distribution of interest is $N(\mu, \Sigma\Sigma^t)$. The results of 10 different runs are shown in Table 1 and Table 2.

Table 1: Results of MCMC: Multivariate Normal. Results of 10 independent runs of full algorithm in Section 4.

	Estimates									
	106.41	104.39	100.40	103.95	103.68	100.95	105.98	102.33	99.95	102.40
	-525.14	-524.19	-524.41	-524.70	-526.58	-525.22	-525.52	-524.31	-522.74	-523.95
	-863.71	-863.87	-859.42	-856.61	-862.88	-862.62	-866.63	-859.77	-868.21	-866.57
	407.99	403.73	403.35	404.56	406.61	409.45	409.35	406.99	406.59	404.20
μ	976.56	971.01	975.57	967.18	975.00	976.11	977.07	974.82	979.93	975.85
	-451.47	-448.95	-449.23	-445.73	-449.28	-446.40	-450.20	-445.58	-446.73	-451.32
	641.23	640.33	637.31	636.82	643.90	644.32	646.91	641.47	648.55	644.17
	-556.63	-557.20	-559.81	-564.46	-560.43	-561.50	-561.68	-562.48	-562.97	-556.53
	796.44	797.07	794.05	797.12	795.07	787.79	798.65	792.79	791.75	796.50
	Acceptance Rates									
	0.2685	0.2565	0.2616	0.2771	0.2469	0.2750	0.2765	0.2755	0.2775	0.3031
	Runtime (in seconds)									
	30.28	29.54	28.90	28.95	33.64	29.59	30.08	29.55	31.54	31.28

Table 2: Summary Statistics for the Estimates in Table 1: Multivariate Normal. μ is the true mean of the target distribution.

Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	μ
103.04	2.23	99.95	106.41	103.54
-524.68	1.03	-526.58	-522.74	-524.46
-863.03	3.63	-868.21	-856.61	-862.79
406.28	2.25	403.35	409.45	405.96
974.91	3.50	967.18	979.93	974.04
-448.49	2.23	-451.47	-445.58	-448.01
642.50	3.81	636.82	648.55	642.51
-560.37	2.79	-564.46	-556.53	-561.15
794.72	3.23	787.79	798.65	796.02

As we see from Table 2, the estimates from the runs look to be close enough with the true mean μ. Trace plots for the first run can be found in Figure 2. The mixing of Markov Chain for the sampling phase, the blue and green phase in the figure, look to be good.

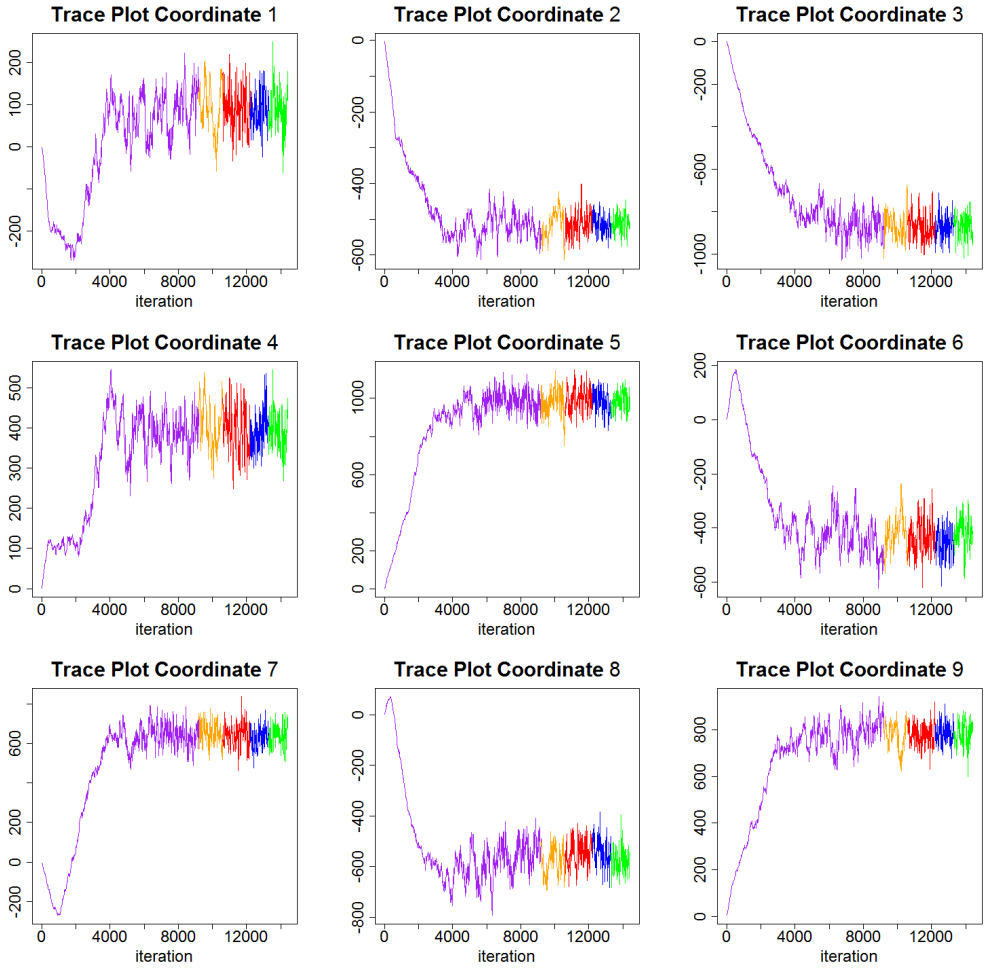

Figure 2: Trace Plots for Multivariate Normal

5.2 Logistic Regression

Next, we run a MCMC with a simple logistic regression model. The data used here is from data(logit) in the R package 'mcmc' (Geyer and Johnson 2014). It is a simulated logistics regression dataset, with a hundred data points and five variables. We name the five variables y, x_1, x_2, x_3 and x_4. y is a Bernoulli response and x_1, x_2, x_3 and x_4 are quantitative predictors. We know the basic structure of a logistic regression is

$$\begin{cases} f(y_1, ..., y_n|\beta, X) = \prod_{i=1}^{n} P(y_i = 1|\beta, X)^{y_i} P(y_i = 0|\beta, X)^{1-y_i} \\ P(y_i = 1|\beta, X) = 1/(1 + exp(-\eta)) \\ \eta = ln[P(y_i = 1|\beta, X)/P(y_i = 0|\beta, X)] = \beta X. \end{cases}$$

If we put a prior $\beta = (\beta_0, \beta_1, ..., \beta_k)^T \sim N(0, 4)$, the posterior distribution of interest is

$$f(\beta|y_1, ..., y_n, X) \propto exp(-\sum \beta_j^2/8)[(\frac{1}{1+exp(-\beta X)})^{\sum y_i}][(1 - \frac{1}{1+exp(-\beta X)})^{n-\sum y_i}].$$

Again, the results of 10 different runs are shown in Table 3 and Table 4, and the trace plots for the first run are shown in Figure 3. We see our algorithm worked fine as we compare its results with MCMC estimates via R package 'mcmc' and with GLM estimates. Trace plots show a good mixing of the Markov chain.

Table 3: Results of MCMC:Logistic Regression. Results of 10 independent runs of full algorithm in Section 4.

	Estimates									
β_0	0.6618	0.6469	0.6671	0.6498	0.6486	0.6494	0.6678	0.6566	0.6654	0.6541
β_1	0.7975	0.7878	0.8240	0.7982	0.8129	0.8039	0.8216	0.8089	0.8082	0.8191
β_2	1.1848	1.2173	1.1621	1.1729	1.1761	1.1593	1.1635	1.1804	1.1574	1.1583
β_3	0.5192	0.5039	0.5120	0.5127	0.5016	0.5063	0.4967	0.5114	0.5077	0.4873
β_4	0.7095	0.7188	0.6875	0.7174	0.7148	0.7192	0.7347	0.7093	0.7229	0.7180
	Acceptance Rates									
	0.2728	0.2925	0.2927	0.2901	0.3028	0.2863	0.2876	0.2901	0.2786	0.2891
	Runtime (in seconds)									
	6.22	6.47	6.18	6.06	6.24	6.16	6.21	6.18	6.33	6.05

Table 4: Summary Statistics for the Estimates in Table 3: Logistic Regression. Results of a run from R package 'mcmc' and GLM estimates are also presented, for comparison.

	Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	Rpackage mcmc	GLM
β_0	0.6567	0.0082	0.6469	0.6678	0.6634	0.6328
β_1	0.8082	0.0117	0.7878	0.8240	0.7629	0.7390
β_2	1.1732	0.0183	1.1574	1.2173	1.2074	1.1137
β_3	0.5059	0.0091	0.4873	0.5192	0.5315	0.4781
β_4	0.7152	0.0121	0.6875	0.7347	0.7408	0.6944

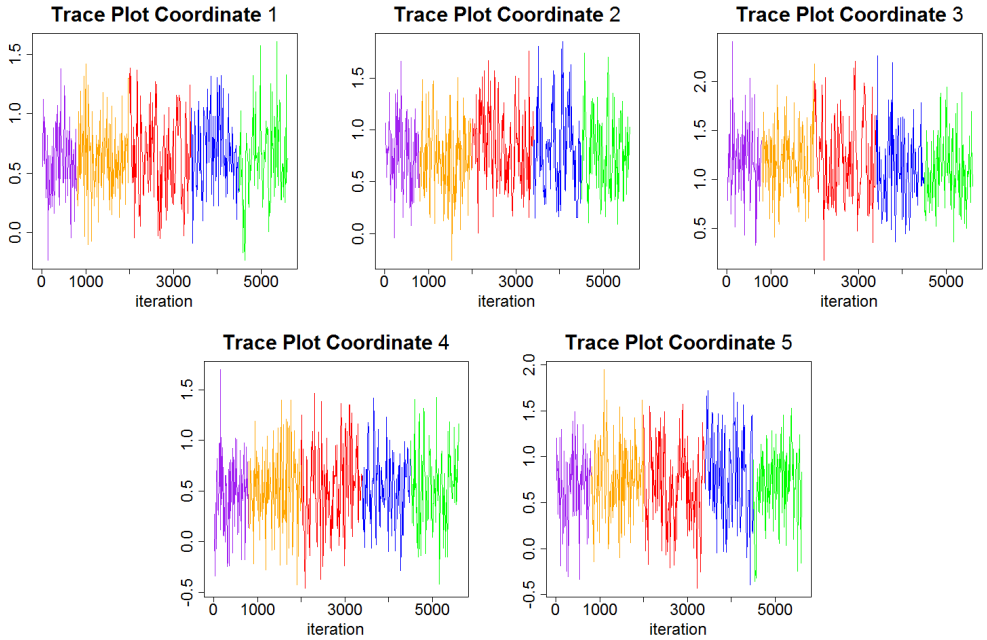

Figure 3: Trace Plots for Logistic Regression. Each coordinate corresponds to the parameters listed in Table 3 as ordered.

5.3 Pump Failure Data

This is the data from Gaver and O'Muircheartaigh (Gaver and O'Muircheartaigh 1987). It is shown in Table 5

Table 5: Pump Failure Data

Obs. no.	1	2	3	4	5	6	7	8	9	10
y_i	5	1	5	14	3	19	1	1	4	22
t_i	94.320	15.720	62.880	125.760	5.240	31.440	1.048	1.048	2.096	10.480

We followed the Bayesian set-up from George et al. (1993) to construct the posterior distribution.

$$f(y_1, ..., y_n|\lambda_1, ..., \lambda_n) = \prod_{i=1}^{n} Poisson(\lambda_i t_i)$$

where n=10, $\lambda_i \sim G(\alpha, \beta)$, $\alpha \sim exp(1)$ and $\beta \sim G(0.1, 1)$. Thus, the posterior distribution of the parameters is

$$f(\lambda_1, ..., \lambda_n, \alpha, \beta | y_1, ..., y_n) \propto e^{-\alpha} \beta^{0.1-1} e^{-\beta} \prod_{i=1}^{n} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} (\lambda_i t_i)^{y_i} e^{-\lambda_i t_i}$$

The results are shown in Table 6, Table 7 and Figure 4. We bring the results from OpenBUGS website (Lunn et al. 2009) to compare with our estimates.

Table 6: Results of MCMC: Pump Failure Data. Results of 10 independent runs of full algorithm in Section 4.

	Estimates									
λ_1	0.0579	0.0626	0.0598	0.0591	0.0603	0.0593	0.0589	0.0595	0.0574	0.0598
λ_2	0.1023	0.1042	0.1014	0.1077	0.1011	0.1042	0.0938	0.1033	0.0965	0.1065
λ_3	0.0889	0.0911	0.0926	0.0923	0.0873	0.0869	0.0903	0.0856	0.0900	0.0870
λ_4	0.1159	0.1163	0.1165	0.1160	0.1126	0.1176	0.1173	0.1150	0.1153	0.1187
λ_5	0.5629	0.6008	0.5884	0.5700	0.5936	0.5954	0.5947	0.6074	0.6090	0.5966
λ_6	0.6030	0.6071	0.5956	0.6140	0.6085	0.5955	0.5941	0.6101	0.6122	0.5973
λ_7	0.9148	0.8865	0.8217	0.8945	0.8879	0.8324	0.8332	0.8539	0.8697	0.8685
λ_8	0.9548	0.8217	0.8764	0.8093	0.8847	0.8935	0.9589	0.9104	0.9402	0.9671
λ_9	1.5930	1.5932	1.5768	1.6202	1.5692	1.4670	1.5332	1.5542	1.6088	1.6102
λ_{10}	2.0065	1.9909	1.9897	2.0038	1.9872	2.0169	1.9948	1.9624	1.9961	2.0507
α	0.6963	0.7026	0.7009	0.7123	0.6864	0.6878	0.7039	0.6965	0.6996	0.6817
β	0.9183	0.9355	0.9080	0.9497	0.9152	0.9498	0.9453	0.8963	0.9304	0.9180
	Acceptance Rates									
	0.1690	0.1837	0.1660	0.1612	0.1694	0.1533	0.1558	0.1593	0.1740	0.1978
	Runtime (in seconds)									
	31.79	24.53	24.23	23.03	25.06	23.64	25.22	22.69	25.95	30.55

Table 7: Summary Statistics for the Estimates in Table 6: Pump Failure Data. Results from the OpenBUGS website, obtained via Gibbs sampler, are also presented, for comparison.

	Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	BUGS
λ_1	0.0595	0.0014	0.0574	0.0626	0.05986
λ_2	0.1021	0.0042	0.0938	0.1077	0.1015
λ_3	0.0892	0.0024	0.0856	0.0926	0.08899
λ_4	0.1161	0.0017	0.1126	0.1187	0.1156
λ_5	0.5919	0.0149	0.5629	0.6090	0.6043
λ_6	0.6037	0.0076	0.5941	0.6140	0.6121
λ_7	0.8663	0.0306	0.8217	0.9148	0.899
λ_8	0.9017	0.0557	0.8093	0.9671	0.9095
λ_9	1.5726	0.0458	1.4670	1.6202	1.587
λ_{10}	1.9999	0.0229	1.9624	2.0507	1.995
α	0.6968	0.0092	0.6817	0.7123	0.6867
β	0.9267	0.0184	0.8963	0.9498	0.9024

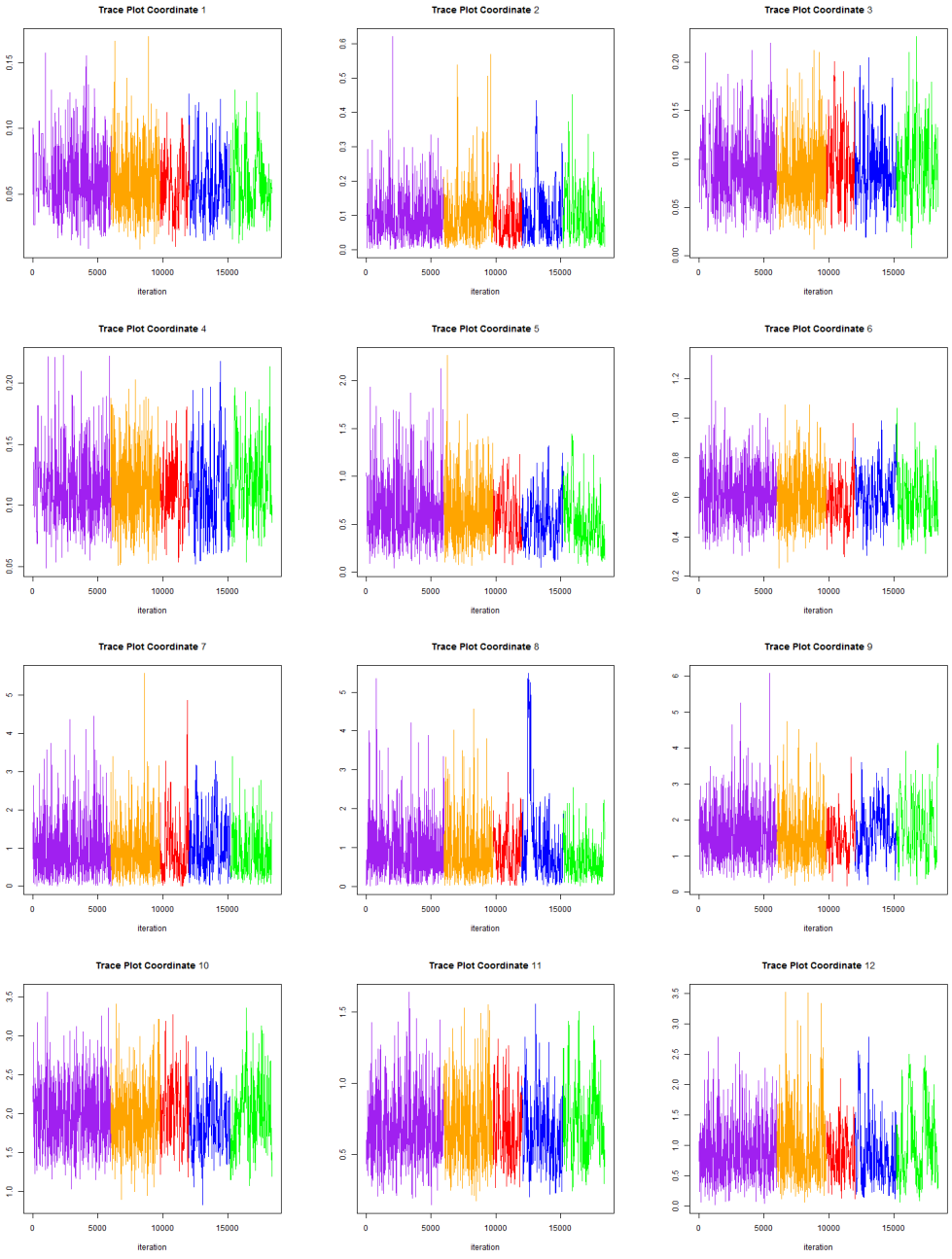

Figure 4: Trace Plots for Pump Failure Data. Each coordinate corresponds to the parameters listed in Table 6 as ordered.

5.4 Variance Components Model (VCM)

The Variance Components Model (VCM) is a well-known example in Bayesian statistics. The structure of the model can be found in Roberts and Rosenthal (2004) and Gelfand and Smith (1990). In short, the model is constructed as:

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

where $\theta_i|\mu, \sigma_\theta^2 \sim N(\mu, \sigma_\theta^2)$. $\theta_i|\mu, \sigma_\theta^2$ are independent of each other. The distributions of hyperparameters 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 full posterior for the VCM is

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

First, we set a_1 and a_2 to 0.001, b_1 and b_2 to 1000, μ_0 to 0, and σ_0^2 to 10^{10}, making the inverse gamma priors flat and uninformative. The results can be found in Table 8, Table 9 and Figure 5. We compare our estimates to the estimates from Gibbs samplers ran for 1.1 million iterations (which is a lot higher than the number of iterations in any of our algorithm runs in Table 8) with last 0.1 million iterations used as a sample. One small problem we found was that our model underestimated σ_θ^2 compared to the Gibbs sampler. This error is correctable if we apply tighter cutoffs for the $\hat{R}_c$ and $\hat{R}_{interval}$ of Gelman-Rubin diagnostics, to make our algorithm run longer. For consistency, we here let the cutoffs for the $\hat{R}_c$ and $\hat{R}_{interval}$ be the same as other examples. Other than that, our estimates and trace plots show that our model worked fine.

Table 8: Results of MCMC:VCM, flat inverse gamma priors. Results of 10 independent runs of full algorithm in Section 4.

	Estimates									
σ_θ^2	3687.3	4113.2	4039.6	3273.6	3907.1	3710.9	3291.5	3346.2	3714.8	3771.4
σ_e^2	2763.2	2832.4	2649.5	2729.7	2713.0	2773.8	2810.0	2779.7	2764.6	2760.1
μ	1527.9	1528.0	1529.7	1527.2	1529.8	1529.5	1527.0	1527.1	1527.3	1528.1
θ_1	1509.7	1509.1	1506.6	1508.9	1509.0	1509.0	1509.7	1510.2	1509.1	1509.8
θ_2	1527.4	1525.7	1528.4	1528.7	1527.0	1527.3	1529.8	1527.6	1528.3	1529.0
θ_3	1556.1	1556.4	1558.3	1556.9	1558.2	1557.2	1557.1	1556.5	1555.8	1557.1
θ_4	1504.2	1505.0	1504.0	1505.2	1503.9	1503.9	1503.5	1504.1	1503.6	1502.7
θ_5	1586.4	1584.9	1587.3	1585.8	1585.7	1587.6	1584.8	1585.6	1585.6	1587.7
θ_6	1480.6	1482.3	1479.3	1481.6	1480.1	1481.2	1481.6	1481.5	1482.8	1479.8
	Acceptance Rates									
	0.1714	0.1891	0.1557	0.2001	0.1822	0.2203	0.2037	0.2135	0.1728	0.1413
	Runtime (in seconds)									
	37.27	28.69	20.36	21.28	21.55	21.22	35.58	52.48	24.66	20.09

Table 9: Summary Statistics for the Estimates in Table 8: VCM, flat inverse gamma priors. Results from a Gibbs sampler run are also presented, for comparison.

	Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	Gibbs
σ^2_θ	3685.6	299.3	3273.6	4113.2	3891.8
σ^2_e	2757.6	51.2	2649.5	2832.4	2769.1
μ	1528.2	1.1	1527.0	1529.8	1527.4
θ_1	1509.1	1.0	1506.6	1510.2	1509.5
θ_2	1527.9	1.2	1525.7	1529.8	1527.9
θ_3	1557.0	0.8	1555.8	1558.3	1556.8
θ_4	1504.0	0.7	1502.7	1505.2	1503.8
θ_5	1586.1	1.1	1584.8	1587.7	1585.6
θ_6	1481.1	1.1	1479.3	1482.8	1481.2

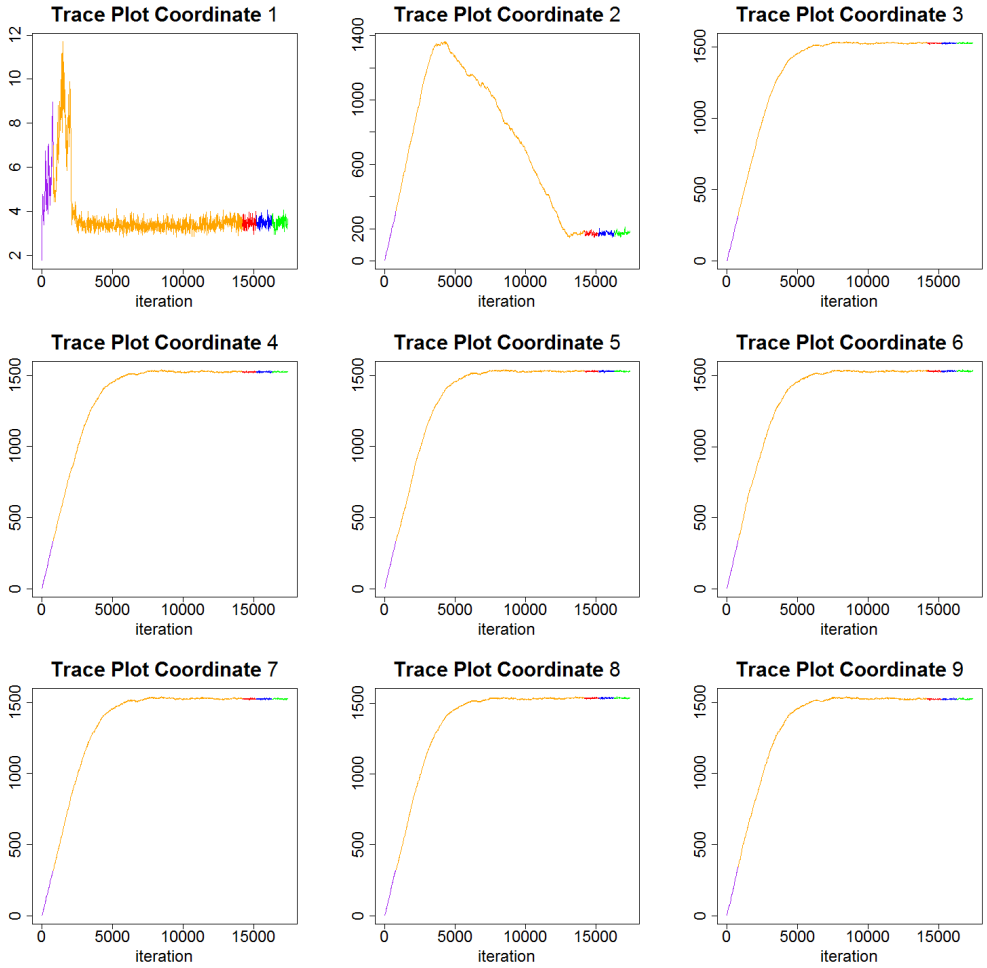

Figure 5: Trace Plots for Variance Components Model. Each coordinate corresponds to the parameters listed in Table 8 as ordered.

This time, we set a_1 and a_2 to 300, b_1 and b_2 to 100 while everything else remained same as before. This makes the inverse gamma priors really concentrated, and the results are shown in Table 10, Table 11 and Figure 6. As we can see from Figure 6, in this example, the transient phase was crucial to find the mode so in the 2^{nd} adaption phase we can avoid using 'bad' values to estimate the covariance of the target distribution.

Table 10: Results of MCMC:VCM, concentrated inverse gamma priors. Results of 10 independent runs of full algorithm in Section 4.

	Estimates									
σ_θ^2	3.5057	3.4959	3.4998	3.5326	3.5163	3.5169	3.5004	3.5144	3.5067	3.5008
σ_e^2	171.38	171.76	170.70	170.55	171.16	171.31	171.07	170.58	171.38	171.55
μ	1527.5	1527.3	1527.3	1527.7	1527.5	1527.9	1527.6	1527.4	1527.5	1527.7
θ_1	1525.3	1525.3	1525.3	1525.5	1525.4	1525.8	1525.4	1525.3	1525.3	1525.6
θ_2	1527.6	1527.5	1527.3	1527.6	1527.5	1527.8	1527.4	1527.3	1527.6	1527.7
θ_3	1531.0	1530.7	1530.7	1531.1	1531.0	1531.2	1530.8	1530.8	1531.0	1531.0
θ_4	1524.7	1524.8	1524.6	1524.8	1524.7	1525.1	1524.7	1524.4	1524.7	1524.9
θ_5	1534.2	1534.2	1534.0	1534.4	1534.3	1534.7	1534.1	1534.2	1534.3	1534.4
θ_6	1522.2	1522.1	1522.0	1522.1	1522.2	1522.7	1522.2	1521.7	1522.3	1522.3
	Acceptance Rates									
	0.2659	0.2631	0.2559	0.2520	0.2705	0.2858	0.2972	0.2635	0.2673	0.2816
	Runtime (in seconds)									
	18.33	17.28	18.57	18.20	24.06	23.33	20.54	16.83	21.69	14.16

Table 11: Summary Statistics for Estimates in Table 10: VCM, concentrated inverse gamma priors. Results from Gibbs sampler are also presented, for comparison.

	Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	Gibbs
σ_θ^2	3.5089	0.0111	3.4959	3.5326	3.5060
σ_e^2	171.14	0.42	170.55	171.76	171.08
μ	1527.5	0.2	1527.3	1527.9	1527.5
θ_1	1525.4	0.2	1525.3	1525.8	1525.4
θ_2	1527.5	0.2	1527.3	1527.8	1527.5
θ_3	1530.9	0.2	1530.7	1531.2	1530.8
θ_4	1524.7	0.2	1524.4	1525.1	1524.7
θ_5	1534.3	0.2	1534.0	1534.7	1534.2
θ_6	1522.2	0.2	1521.7	1522.7	1522.1

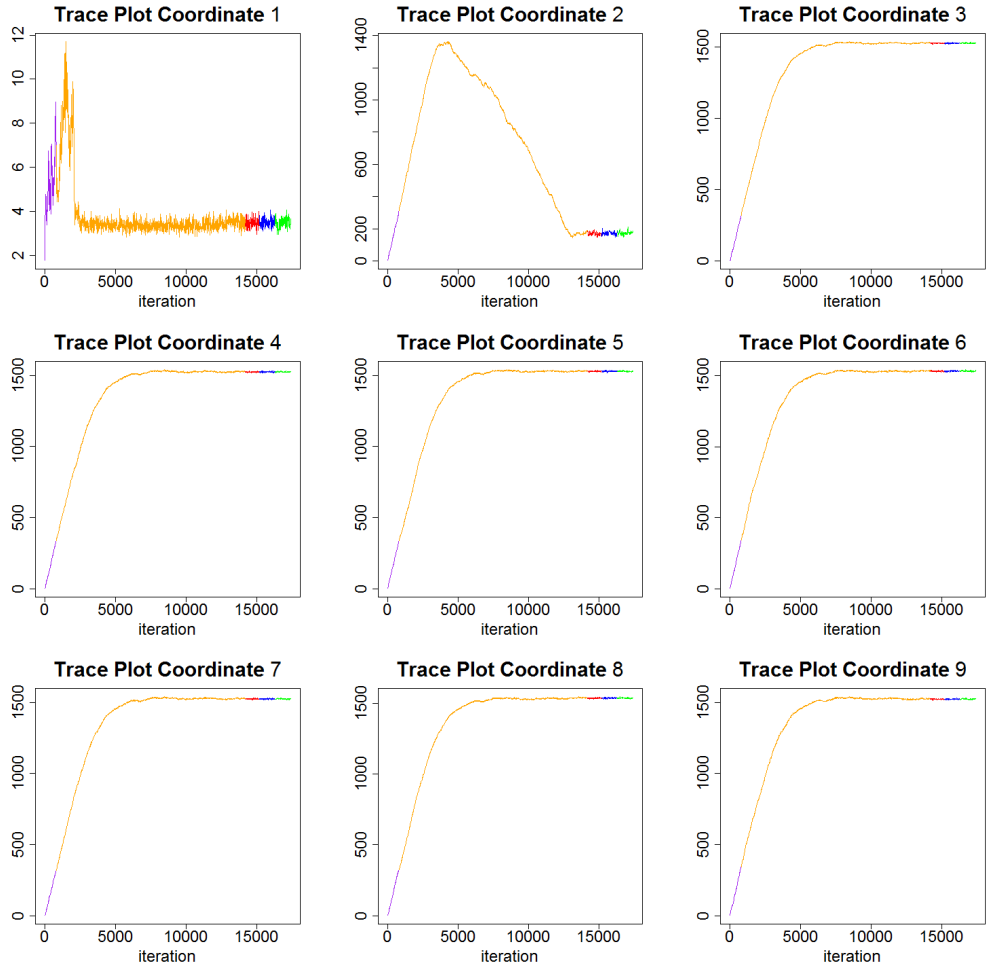

Figure 6: Trace Plots for Variance Components Model. Each coordinate corresponds to the parameter listed in Table 10 as ordered.

5.5 A Strongly Multimodal Example

The 'strongly multimodal' target density function we work with is

$$\frac{1}{3} * N(\mu_1, \Sigma\Sigma^t) + \frac{1}{3} * N(\mu_2, \Sigma\Sigma^t) + \frac{1}{3} * N(\mu_3, \Sigma\Sigma^t)$$

where each component of Σ is randomly drawn from $N(0,1)$. The parameter values are

$$\begin{cases} \mu_1 &= (21.62166, -10.00424, 15.49878)^T \\ \mu_2 &= (9.671977, -28.515220, -12.744802)^T \\ \mu_3 &= (26.0518930, 0.2331812, -0.3433256)^T \end{cases}$$

$$\Sigma\Sigma^t = \begin{pmatrix} 1.2742983 & 0.1801673 & -1.353580 \\ 0.1801673 & 2.6300580 & 1.451527 \\ -1.3535803 & 1.4515267 & 4.861334 \end{pmatrix}.$$

We first ran 10 different chains with the starting values of the 1^{st} adaption phase chosen randomly from $U(-30, 30)$ for every coordinate of each chain. Since we want only one chain for one particular mode, once every chain ran for both 1^{st} adaption phase and transient phase, we reduced the number of chains down, based on the criteria described in Section 4.5, leaving us with 3 different chains. Then, we ran for the 2^{nd} adaption phases for all 3 chains, separately. Once this is done, we once again checked every chain had different modes with the values generated in the 2^{nd} adaption phase. Then we randomly chose 7 starting points as described in Section 4.5. With 10 different starting points in total (7 starting values randomly chosen and the last values from the 3 different chains), we created 10 replicative chains to run for the sampling phase. The plots for the sampling phase in Figure 7 are from one replicative chain which used the last values of the 2^{nd} adaption phase of the chain started with 'mode 1' as the starting point of the sampling phase. We did 10 different runs with the same starting values for the 1^{st} adaption phase, and the end results we got from the runs are shown in Table 12, Table 13 and Figure 7.

Table 12: Results of MCMC: Multimodal Distribution. Results of 10 independent runs of full algorithm in Section 4.5.

	Estimates									
μ	18.764	18.170	20.413	18.965	18.853	19.746	19.690	18.687	19.241	20.200
	-13.440	-14.311	-9.723	-12.907	-13.093	-11.551	-11.769	-13.604	-12.467	-10.825
	0.9235	-0.1113	2.7895	0.3334	-0.0131	1.0996	1.8526	1.1745	0.9141	1.5964
	Acceptance Rates									
	0.3374	0.3489	0.3057	0.3449	0.3464	0.3370	0.3627	0.3472	0.3448	0.3471
	Runtime (in seconds)									
	72.04	71.40	70.26	72.93	70.31	74.74	71.33	72.50	72.20	71.29

Table 13: Summary Statistics for the Estimates in Table 12: Multimodal Distribution. μ is the true mean of target distribution.

	Mean of Estimates	SD of Estimates	Min Estimate	Max Estimate	μ
	19.273	0.719	18.170	20.413	19.115
	-12.369	1.401	-14.311	-9.723	-12.762
	1.0559	0.8827	-0.1113	2.7895	0.804

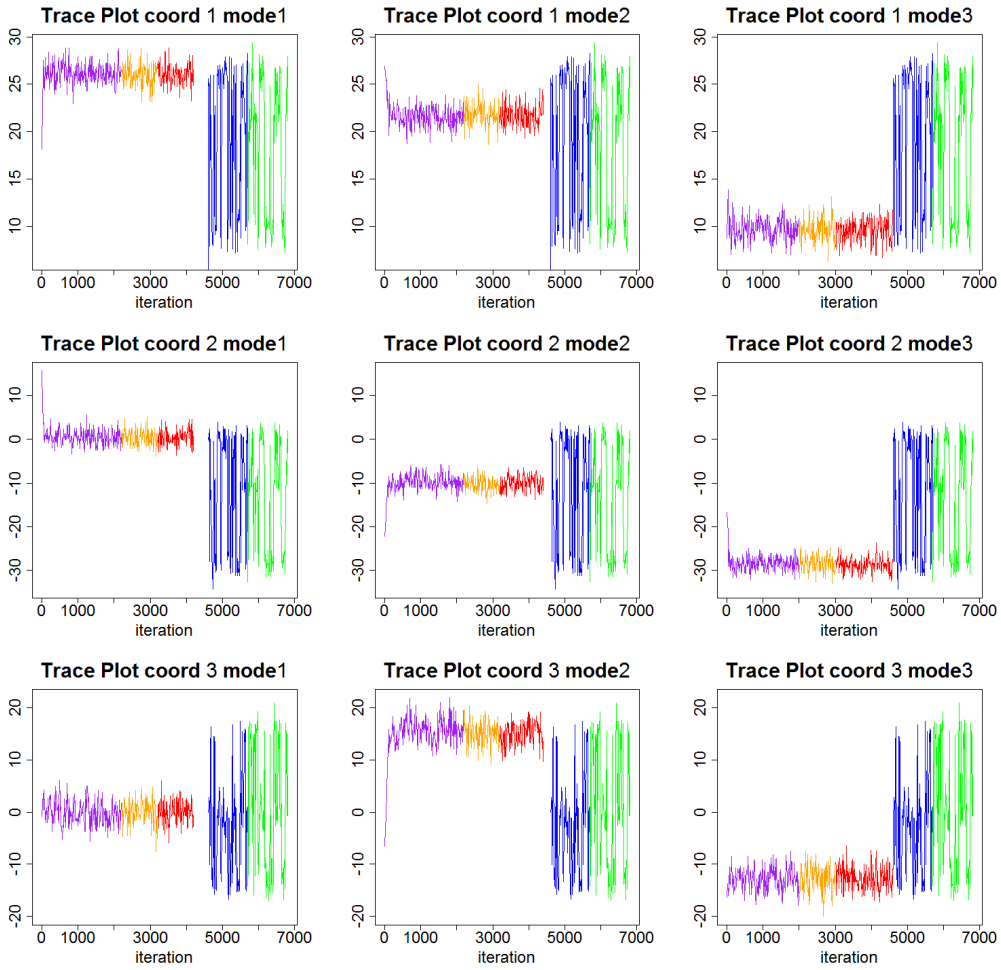

Figure 7: Trace Plots for Mixture of Three 3-Dimensional Multivariate Normals. Each row represents each coordinate and each column represents each chain trapped in different mode until the sampling phase.

As we see from Figure 7, the mixing of the Markov chain for the sampling phase looks to be good and not just occurs within one mode as in the previous

phases.

6 Discussion

This paper has presented a new algorithm, implemented in the R package 'atmcmc' (Yang 2014), to run adaptive MCMC for a finite amount of time, diagnose when the adaption is sufficient, and then run conventional MCMC with standard convergence diagnostics to get good target distribution convergence and estimates. The algorithm was illustrated on a number of examples, and found to perform quite well in each case.

We finish by discussing a number of related issues.

6.1 Acceptance Rate

Our algorithm makes heavy use of the fact (Roberts et al. 1997; Roberts and Rosenthal 2001) that for the symmetric random walk Metropolis algorithm on certain target densities, the optimal asymptotic acceptance rate is 0.234, which can be achieved if we use the proposal distribution $N(X_n, \Sigma_p)$ where $\Sigma_p = (2.38^2/d) * \Sigma$ and Σ is the target covariance matrix. Now, for certain truncated (discontinuous) target densities, the optimal acceptance rate is actually 0.1353 (Neal et al. 2012), and the true optimal value of the multiple to Σ is unknown. We used the multiple $2.38^2/d$ whether the target density is truncated or not. The acceptance rates we found in Table 1, Table 3 and Table 6 are slightly higher than the optimal acceptance rates, which is reasonable as our Markov chains have dimensions far from infinity. The acceptance rates for the Variance Components Model (VCM) look to be more puzzling, as they vary between 0.1353 and 0.234 in Table 8 but are close to 0.234 in Table 10. In VCM some coordinates are truncated and some are not, and there is little known about the optimal asymptotic acceptance rate in this case. Also, note that for the 'strongly multimodal' algorithm extension, the acceptance rates are a lot higher than 0.234 as seen in Table 12, which is due to the fact that direct jumps between the modes are allowed in this case.

6.2 Significance of Transient and 2^{nd} Adaption Phase

One question is whether our 2^{nd} adaption phase helps in terms of convergence speed, or whether our 1^{st} adaptive phase alone would be sufficient. To check this, we removed the 2^{nd} adaption phase from the full algorithm and ran each unimodal example three times, with starting values for Gelman-Rubin

diagnostics picked based on the flat part of the transient phase. The runtimes we got for 3 different runs of the 9-dimensional multivariate normal example are 121.07, 79.89, and 76.13 seconds, respectively. These runtimes are all larger than any of runtimes we got for our full MCMC model in Table 1. Estimates from all three runs came out reasonable. For the logistic regression example, we got 8.78, 6.84 and 9.43 seconds for 3 runs; for pump failure data, we got 5330.87, 5007.72 and 1490.13 seconds; for VCM with flat inverse gamma priors, we got 72.65, 149.96 and 80.34 seconds, and for VCM with concentrated inverse gamma priors, we got 202.66, 89.36 and 153.84 seconds. Estimates from all these runs came out reasonably good except σ_θ^2 for VCM with flat inverse gamma priors was even more underestimated (about 2700 - 3200) than results in Table 9 from full algorithm. We conclude that runtimes were larger (most times by a lot) without a 2^{nd} adaption phase than with our full algorithm.

As another test, we tried removing both the transient phase and the 2^{nd} adaption phase, and again ran each unimodal example three times. (Here the starting values for the Gelman-Rubin diagnostic were chosen based on the values from the 1^{st} adaption phase, since we had neither a transient phase nor a 2^{nd} adaption phase; that is, we had to pick starting values for the sampling phase without having a rough idea on the range of target distribution.) For the 9-dimensional multivariate normal example, we got runtimes of 101.70, 80.22 and 69.03 seconds; for logistic regression, we got 8.47, 8.19 and 7.53 seconds; for pump failure data, we got 10756.04, 2593.72 and 2866.11 seconds; for VCM with flat inverse gamma priors, we got 6227.88, 744.50 and 72.83 seconds, and for VCM with concentrated inverse gamma priors, we got 918.97, 2774.98 and 2551.30 seconds. The estimates again came out reasonable. But once again, all runtimes were larger than the corresponding ones for our full algorithm.

6.3 Comparison with a Full-dimensional Metropolis

We next consider how our algorithm fares against a full-dimensional Metropolis algorithm. Notice that we are not comparing our algorithm to the full-dimensional Metropolis with a really good proposal distribution. Finding a good proposal distribution for the full-dimensional Metropolis algorithm is the goal of the first three phases of our algorithm, and if we know a good proposal distribution from the start, there is no need to adapt the chain. However, in most cases of MCMC examples, we have little to no idea on the target distribution, and picking a good proposal distribution out of all possible choices is virtually impossible without adaption.

Since our algorithm runs replicative chains in the sampling phase to check

for the convergence, it wouldn't be fair to compare our algorithm to one full-dimensional Metropolis chain in terms of runtime. Also, if we only run one full-dimensional Metropolis chain, we have to find a way to figure out when the full-dimensional Metropolis chain achieved the same level of convergence as the chain by our algorithm. Thus, we compare our algorithm with the full-dimensional Metropolis algorithm in terms of number of iterations they run for, and we run the same number of replicative chains for the full-dimensional Metropolis algorithm to check for the convergence through Gelman-Rubin diagnostics, using the same $\hat{R}_c$ and $\hat{R}_{interval}$ cutoffs with our algorithm.

We ran the full-dimensional Metropolis on the pump failure data from Section 5.3 and the VCM from Section 5.4. The proposal distributions for the Metropolis algorithm are $N(X_n, I)$ for the pump failure data and the VCM with flat inverse gamma priors and $N(X_n, 10I)$ for the VCM with concentrated inverse gamma priors, where X_n is the current state value and I is the identity matrix. We took the initial value $0.1 * \mathbf{1}$ for the Metropolis algorithm, which is the initial value of the runs of our algorithm for all unimodal examples in Section 5. We also ran the same number of replicative chains, 10, as what we used in the runs of our algorithm in Section 5 for the Gelman-Rubin diagnostics, with the same $\hat{R}_c$ and $\hat{R}_{interval}$ cutoffs. With pump failure data, we ran the full-dimensional Metropolis for 2 million iterations, but the algorithm failed to achieve the convergence by the Gelman-Rubin diagnostics. Our algorithm was proved to be significantly more efficient in this case in terms of number of iterations since all of the runs in Section 5.3 converged in between 81200 and 126200 iterations. For VCM with flat inverse gamma priors, the Metropolis algorithm again didn't achieve the convergence for 2 million iterations by the Gelman-Rubin diagnostics. All the runs of our algorithm for the same example, shown in Section 5.4, achieved the convergence in between 156800 and 299600 iterations. For VCM with concentrated inverse gamma priors, the Metropolis algorithm achieved the convergence in 1775200 iterations, and the comparative runs in Section 5.4 converged somewhere in between 77200 and 210200 iterations. Thus, we conclude that our algorithm performed notably better in these particular cases.

Even if the proposal distribution is not as bad as what we just described, our algorithm still beat the full-dimensional Metropolis algorithm with a decent proposal distribution. When we removed both transient and 2^{nd} adaption phase from our algorithm in Section 6.2, we practically ran the full-dimensional Metropolis algorithm with a roughly-tuned diagonal proposal covariance matrix. So, we took the run with the fewest iterations for the 1^{st} adaption phase out of three runs from Section 6.2 to compare the efficiency of our algorithm to the full-dimensional Metropolis with a roughly tuned proposal distribution. This comparison is really not fair to our algorithm

since we counted the number of iterations for the 1^{st} adaption phase against our algorithm but gave the tuned proposal distribution obtained from the 1^{st} adaption phase of our algorithm to the full-dimensional Metropolis from the start. However, our algorithm still did better.

For pump failure data example, two chains with the fewest number of iterations for the 1^{st} adaption phase from Section 6.2 took 664200 and 333800 iterations, respectively, counting only for the sampling phase, or in other words the standard full-dimensional Metropolis phase. Thus, the full-dimensional Metropolis algorithm took 2.65 to 8.18 times more iterations to reach the same level of the convergence compared to the runs in Section 5.3. For VCM with flat inverse gamma priors, the number of iterations for the run with the fewest 1^{st} adaption period from Section 6.2 was 547400. Our algorithm ran 1.83 to 3.49 times faster in terms of number of iterations. For VCM with the concentrated inverse gamma priors, our algorithm was 1.58 to 4.57 times better than the full-dimensional Metropolis run from Section 6.2.

6.4 Initial Value for a Markov Chain

One important issue for improving Markov chain convergence speed is getting a good starting value. This problem is solved naturally for unimodal target distributions in our algorithm since we go through all the different phases, including the mode-finding transient phase, before starting the final sampling phase. The information we obtain in the first 3 phases are used to get a rough idea about the target distribution, and hence to pick good starting values for the sampling phase. The situation is a bit more problematic for a 'strongly multimodal' target distribution. There it is important to have well-dispersed starting points to find all the modes in the target distribution, right from the beginning. If we fail to find some modes, our resulting estimates will be compromised. Of course, similar concerns apply to virtually all MCMC algorithms and diagnostic approaches.

Implementing MCMC in practice does have some difficulties depending on the target distribution one is dealing with. We hope that the algorithm presented in this paper, and implemented in the R package 'atmcmc' (Yang 2014), will provide a simple and efficient approach that can be applied to most target distributions which arise in practice.

References

Andrieu, C. and Atchadé, Y. F. (2007). On the efficiency of adaptive MCMC algorithms. *Electronic Communications in Probability*, 12(33):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(3):1462–1505.

Atchadé, Y. F. and Rosenthal., J. S. (2005). On adaptive Markov Chain Monte Carlo algorithms. *Bernoulli*, 11(5):815–828.

Brooks, S., Gelman, A., Jones, G. L., and Meng, X., editors (2011). *Handbook of Markov Chain Monte Carlo*. Taylor & Francis.

Brooks, S. P. and Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of computational and graphical statistics*, 7(4):434–455.

Fort, G., Moulines, E., and Priouret, P. (2011). Convergence of adaptive and interacting Markov chain Monte Carlo algorithms. *The Annals of Statistics*, 39(6):3262–3289.

Gaver, D. P. and O'Muircheartaigh, I. G. (1987). Robust empirical Bayes analyses of event rates. *Technometrics*, 29(1):1–15.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409.

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

Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472.

George, E. I., Makov, U. E., and Smith, A. F. M. (1993). Conjugate likelihood distributions. *Scandinavian Journal of Statistics*, 20(2):147–156.

Geyer, C. J. and Johnson, L. T. (2014). *MCMC: Markov Chain Monte Carlo*. R package version 0.9-3. `http://CRAN.R-project.org/package=mcmc`.

Giordani, P. and Kohn, R. (2010). Adaptive independent Metropolis–Hastings by fast estimation of mixtures of normals. *Journal of Computational and Graphical Statistics*, 19(2):243–259.

Haario, H., Laine, M., Mira, A., and Saksman, E. (2006). DRAM: efficient adaptive MCMC. *Statistics and Computing*, 16(4):339–354.

Haario, H., Saksman, E., and Tamminen, J. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109.

Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The BUGS project: Evolution, critique, and future directions. *Statistics in Medicine*, 28(25):3049–3067. `http://www.openbugs.net`.

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(6):1087–1092.

Neal, P. J., Roberts, G. O., and Yuen, W. K. (2012). Optimal scaling of random walk Metropolis algorithms with discontinuous target densities. *The Annals of Applied Probability*, 22(5):1880–1927.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. `http://www.R-project.org`.

Roberts, G. O., Gelman, A., and Gilks, W. R. (1997). Weak convergence and optimal scaling of random walk Metropolis algorithms. *The annals of applied probability*, 7(1):110–120.

Roberts, G. O. and Rosenthal, J. S. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical science*, 16(4):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(2):458–475.

Roberts, G. O. and Rosenthal, J. S. (2009). Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367.

Rosenthal, J. S. (2004). Adaptive MCMC Java applet. `http://probability.ca/jeff/java/adapt.html`.

Rosenthal, J. S. (2007a). AMCMC: An R interface for adaptive MCMC. *Computational Statistics & Data Analysis*, 51(12):5467–5470.

Rosenthal, J. S. (2007b). The AMCMC package. `http://probability.ca/amcmc`.

Rosenthal, J. S. (2011). Optimal proposal distributions and adaptive MCMC. In Brooks, S., Gelman, A., Jones, G. L., and Meng, X., editors, *Handbook of Markov Chain Monte Carlo*, pages 93–112. Taylor & Francis.

Scheidegger, A. (2012). *adaptMCMC: Implementation of a generic adaptive Monte Carlo Markov Chain sampler*. R package version 1.1. `http://CRAN.R-project.org/package=adaptMCMC`.

Soetaert, K. and Petzoldt, T. (2010). Inverse modelling, sensitivity and Monte Carlo analysis in R using package FME. *Journal of Statistical Software*, 33(3):1–28.

Soetaert, K. and Petzoldt, T. (2014). *FME: A Flexible Modelling Environment for Inverse Modelling, Sensitivity, Identifiability, Monte Carlo Analysis*. R package version 1.3.1. `http://CRAN.R-project.org/package=FME`.

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(1):439–448.

Vihola, M. (2010a). Grapham: graphical models with adaptive random walk Metropolis algorithms. *Computational Statistics and Data Analysis*, 54(1):49–54.

Vihola, M. (2010b). The Grapham package. `http://www.stats.ox.ac.uk/~mvihola/grapham/`.

Vihola, M. (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22(5):997–1008.

Yang, J. (2014). *atmcmc: Automatically Tuned Markov Chain Monte Carlo*. R package version 1.0. `http://CRAN.R-project.org/package=atmcmc`.