University of Oxford

Approximate inference for large data sets in a sample Bayesian setting

by

Adrien Bec-Hairault

Green Templeton College

A dissertation submitted in partial fulfilment of the degree of Master of Science in Statistical Science.

Department of Statistics, 24–29 St Giles,

Oxford, OX1 3LB

September 2018

This is my own work (except where otherwise indicated).

Candidate: Adrien Hairault

Signed:.....

Date:.....

Abstract

The use of classic Monte Carlo Markov Chains (MCMC) algorithms in the context of Bayesian inference, which usually involves repeated likelihood evaluations, has become more and more computationally expensive with the advent of Big Data. This dissertation assesses and compares the efficiency of five recent algorithms designed to respond to this challenge. First, we review extensively the theory behind each algorithm, provide a potential pseudo-code implementation, and identify mistakes in two of the original articles, one of which seems substantial. Second, we test the algorithms on different synthetic data sets to reveal their strengths and weaknesses. Among other techniques, the key metrics we use are the Effective Sample Size (ESS) per likelihood evaluation (for efficiency) as well as the quality of the posterior approximation, measured by the Hellinger distance between the desired target and the obtained output.

Acknowledgements

I would like to thank my dissertation supervisor Geoff Nicholls for his patient and helpful guidance. Thank you for having answered so promptly all my questions and queries and for coming up with great ideas.

I would also like to thank my mother, Frédérique Bec, for the moral support and the help she gave me during the proof-reading process.

Thank you Olga Kosno, my dear friend, so far and yet so close.

Finally, completing this dissertation and this year in Oxford would certainly have been harder without the help of the wonderful persons I have met and whom I know I will meet again.

Contents

Introduction

Both frequentist and Bayesian approaches of inference assume that for any parameter θ there exists a true value θ_0. However, Bayesian inference regards parameter θ as a random variable, which is not conceivable in a frequentist framework. The *posterior* distribution of parameter θ, cornerstone of Bayesian analysis, updates thanks to Bayes' formula[1] the prior knowledge of the scientist about θ, embodied by a *prior* distribution, with the information about the parameter carried by the data, given by the *likelihood* distribution. In short,

$$\begin{aligned} \pi(\theta|z) &= \frac{p(z|\theta)\pi(\theta)}{\int_{\theta\in\Theta} p(z|\theta)\pi(\theta)d\theta} \\ &\propto p(z|\theta)\pi(\theta) \end{aligned} \tag{1}$$

where, z, $\pi(.|z)$, $\pi(.)$, $p(.|\theta)$, Θ denote respectively the data, the posterior and the prior distributions of θ, the likelihood function, and the parameter space.

Hence, in a Bayesian inference framework, the posterior distribution can be seen as a measure of uncertainty, or a strength of belief about the true value of parameter θ

However, in practice, this distribution is usually intractable and measures of interest like its mean or mode may not be derived analytically, leading to the dominance of frequentist methods over Bayesian ones in the first half of the 20th century. The emergence of

[1]The Bayes' theorem, derived by Thomas Bayes (1701-1761) and Pierre-Simon Laplace (1749-1827), states that for any events A and B such that $P(B) \neq 0$,

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

Monte Carlo Markov Chains (MCMC) with the Metropolis-Hastings (MH) algorithm by Metropolis et al. (1953) coupled with the advent of powerful computers enabled a strong resurgence of Bayesian analysis. This fairly simple algorithm allows sampling from any distribution provided one can evaluate its probability density function up to a normalising constant. We give below a possible implementation of the random walk Metropolis Hastings algorithm.

Input : data set z, number of samples T, initial state θ_0, proposal distribution q
Output: $\{\theta_t\}_{t=0}^{T}$ asymptotically from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$

Initialization;
$llik_{current} \leftarrow p(z|\theta_0)$;
for t=0,...,T **do**
  $\theta' \sim q(.|\theta_t)$;
  $llik_{proposal} \leftarrow p(z|\theta')$;
  $u \sim \mathcal{U}_{[0,1]}$;
  $\alpha \leftarrow \min(1, \frac{llik_{proposal}\pi(\theta')q(\theta_t|\theta')}{llik_{current}\pi(\theta_t)q(\theta'|\theta_t)})$;
  if $u \leq \alpha$ **then**
    $\theta_{t+1} \leftarrow \theta'$;
    $llik_{current} \leftarrow llik_{proposal}$
  else
    $\theta_{t+1} \leftarrow \theta_t$
  end
end

Algorithm 1: `Metropolis-Hastings`

Among other methods, Monte Carlo Markov Chains (MCMC) algorithms have been used for years in order to approximate any posterior distribution in the context of Bayesian inference. However, repeated evaluations of the likelihood function, which are at the core of the vast majority of MCMC algorithms (cf algorithm 1), have become more and more computationally expensive with the recent and steady increase in size of data sets, also known as Big Data.

This dissertation aims at reviewing five recent MCMC methods designed to bypass likelihood evaluations on the full data set in order to reduce the high computational cost inherent to the implementation of classic MCMC methods. The diverse choice of algorithms reviewed reflects our desire to cover a broad part of the spectrum of current research in this field of statistics. Using Bardenet et al. (2017)'s terminology, we may classify those

algorithms as falling into two categories. Among the *Divide-and-conquer approaches* we present *Consensus Monte Carlo* by Scott et al. (2016) and the *Non-Parametric Density Product Estimator* (NPDPE) by Neiswanger et al. (2013). The other algortihms we review fall into the *subsampling methods* category, namely a *Pseudo-Marginal* (PM) algorithm by Quiroz et al. (2018), a *rejection-free* stochastic-gradient method by Welling and Teh (2011) which builds upon Langevin Dynamics (SGLD), and the *Approximate Metropolis-Hastings test* (ApMHT) by Korattikara et al. (2014).

In Chapter 1 we review extensively each method explaining the mechanisms at work and providing our pseudo-code implementation. We also report the proofs of interest and complete them by filling out the gaps left intentionally by the authors or by showing some properties left unproved. Furthermore, after spotting a mistake in an essential consistency proof of algorithm ApMHT, we provide a corrected alternative that leads to the desired result.

In Chapter 2 we explain in details our methodology and assessment criteria before confronting our algorithms to four synthetic Bayesian models whose posteriors display features such as multi-modality, high variance and high dimensionality. We then sum up our discoveries and the limitations of our analysis.

Chapter 1

Algorithms

In this section, we attempt to make notation consistent across the five articles of interest. Therefore, $q(\theta'|\theta), p(z|\theta), \pi(\theta|z)$ and $\pi(\theta)$ will denote respectively the proposal distribution used in the algorithm, the likelihood function, the posterior and the prior distributions. We also let z be a data set of size N and θ a parameter in $\Theta \subset \mathbb{R}^d$.

1.1 Approximate Metropolis-Hastings Test (ApMHT)

We now present the first of the algorithms we review in this dissertation. Korattikara et al. (2014) propose an interesting alternative to evaluating the likelihood function on the whole data set.

Suppose that the observations are conditionally independent given the parameter θ, i.e that $p(z|\theta) = \prod_{i=1}^{N} p(z_i|\theta)$, then the posterior distribution can be written as follows.

$$\pi(\theta|z) \propto \pi(\theta) \prod_{i=1}^{N} p(z_i|\theta)$$

We prove the following proposition that is used by Korattikara et al. (2014) in order to reformulate the usual MH accept-reject step (cf algorithm 1)

Proposition 1.1.1. *Let* $u \sim \mathcal{U}_{[0,1]}$ *and* $\alpha = \min\left(1, \frac{\pi(\theta'|z)q(\theta_t|\theta')}{\pi(\theta|z)q(\theta'|\theta_t)}\right)$. *Then* $u \leq \alpha \Leftrightarrow \mu_0 \leq \mu$,

where

$$\mu_0 = \frac{1}{N} \log \left(u \frac{\pi(\theta_t) q(\theta' | \theta_t)}{\pi(\theta') q(\theta_t | \theta')} \right) \tag{1.1}$$

$$\mu = \frac{1}{N} \sum_{i=1}^{N} l_i \ \textit{where } l_i = \log p(z_i | \theta') - \log p(z_i | \theta_t) \textit{for } \theta', \theta_t \in \Theta \tag{1.2}$$

And α denotes the standard Metropolis-Hastings acceptance probability (cf algorithm 1)

Proof.

$$\begin{aligned}
\mu_0 \leq \mu &\Leftrightarrow \log \left(u \frac{\pi(\theta_t) q(\theta' | \theta_t)}{\pi(\theta') q(\theta_t | \theta')} \right) - \sum_{i=1}^{N} l_i \leq 0 \\
&\Leftrightarrow \log \left(u \frac{\pi(\theta_t) \prod_{i=1}^{N} p(z_i | \theta_t) q(\theta' | \theta_t)}{\pi(\theta') \prod_{i=1}^{N} p(z_i | \theta') q(\theta_t | \theta')} \right) \leq 0 \\
&\Leftrightarrow u \frac{\pi(\theta_t) \prod_{i=1}^{N} p(z_i | \theta_t) q(\theta' | \theta_t)}{\pi(\theta') \prod_{i=1}^{N} p(z_i | \theta') q(\theta_t | \theta')} \leq 1 \\
&\Leftrightarrow u \leq \alpha \quad \square
\end{aligned}$$

The trick used by Korattikara et al. (2014) is to bypass the full computation of μ by only using a subsample $\mathcal{X} \subset z$ of size $m \leq N$ drawn uniformly at random from $\{1, ..., N\}$ without replacement. A simple Student's t-test is then performed in order to determine whether the estimator $\hat{\mu} = \bar{l} = \frac{1}{|\mathcal{X}|} \sum_{i \in \mathcal{X}} l_i$ of μ is significantly different from μ_0. If the test is conclusive, then Proposition 1.1.1 enables us to make a decision regarding the acceptance or rejection of a candidate θ' by checking the sign of $\hat{\mu} - \mu_0$. On the contrary, if $\hat{\mu}$ is not significantly different from 0, then m new data points are randomly added to $\mathcal{X}$ and this until we do not need any more data to decide whether $\mu_0 < \mu$ or $\mu_0 > \mu$. See algorithm 2 for a more detailed implementation.

Input : z, T, ε, m, θ_0, q
Output: $\{\theta_t\}_{t=0}^{T}$ asymptotically and approximately from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$

Initialization;
$number\_llik\_eval \leftarrow 0$;
for t=0,...,T **do**
  $data \leftarrow z$ $n \leftarrow 0$, $done \leftarrow$ `FALSE`;
  $\theta' \sim q(.|\theta_t)$;
  $u \sim \mathcal{U}_{[0,1]}$;
  $\mu_0 = \frac{1}{N} \times \log\left(u \frac{\pi(\theta_t)q(\theta'|\theta_t)}{\pi(\theta')q(\theta_t|\theta')}\right)$;
  $\mathcal{X} = \emptyset$;
  while $done =$ *FALSE* **do**
    Draw mini-batch $\mathcal{X}'$ of size $\min(m, N-n)$ from $data$ without replacement and do $data \leftarrow data \setminus \mathcal{X}'$ as well as $\mathcal{X} = (\mathcal{X}, \mathcal{X}')$;
    $n = |\mathcal{X}|$
    $\bar{l} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \log p(\mathcal{X}_i;\theta') - \log p(\mathcal{X}_i;\theta_t)$;
    $\bar{l^2} \leftarrow \frac{1}{n}\sum_{i=1}^{n} (\log p(\mathcal{X}_i;\theta') - \log p(\mathcal{X}_i;\theta_t))^2$;
    $\hat{s}_l \leftarrow \sqrt{\frac{n}{n-1}(\bar{l^2} - \bar{l}^2)}$;
    $\hat{s} \leftarrow \frac{\hat{s}_l}{\sqrt{n}}\sqrt{1 - \frac{n-1}{N-1}}$;
    $\delta \leftarrow 1 - \phi(|\bar{l} - \mu_0|/\hat{s}); n-1)$;
    if $\delta < \varepsilon$ **then**
      if $\bar{l} > \mu_0$ **then**
        $\theta_{t+1} = \theta'$
      else
        $\theta_{t+1} = \theta_t$
      end
      $done \leftarrow TRUE$;
      $number\_llik\_eval \leftarrow number\_llik\_eval + n$
    end
  end
end

Algorithm 2: `ApMHT`. $\phi(.;n)$ denotes the CDF of the Student t-distribution with n degrees of freedom

Note that when $m = N$, then ApMHT is equivalent to standard MH. Indeed, if $m = N$ then $\hat{\mu} = \bar{l} = \mu$ and $\hat{s}$ as defined in the algorithm above is equal to 0. Hence $\delta = 0$ which in turn implies that ApMHT becomes the standard MH algorithm.

ApMHT is an approximate method since it does not target the true posterior exactly. However, Korattikara et al. (2014) manage to find a reasonably small upper bound to the total variation distance[1] between the posteriors targeted by ApMHT on the one hand and standard MH, on the other hand.

[1]Let $(\Theta, \mathcal{S}, \mu)$ be a measure space and P_1 and P_2 be two probability measures with Radon-Nikodym derivatives f_{P_1} and f_{P_2} respectively. Then the total variation distance is $d_T \equiv \frac{1}{2}\int_\Theta |f_{P_1}(\theta) - f_{P_2}(\theta)| d\mu(\theta)$

Let $(\mathcal{K}_0, \pi_0, \alpha_0)$ and $(\mathcal{K}_\varepsilon, \pi_\varepsilon, \alpha_\varepsilon)$ denote the transition kernels, the target distributions and the acceptance probability of standard MH and ApMHT respectively. Note that α_ε is analytically intractable since it is an expectation over multiple random variables, namely the batch size m, the batch $\mathcal{X}$ as well as the proposal $\theta' \sim q$.

We first show the following lemma in greater details than the original article.

Lemma 1.1.1. *Let $\mathcal{P}$ be a probability distribution and $\eta \in [0, 1)$.*
If

$$d_T(P\mathcal{K}_0, \pi_0) \leq \eta d_T(P, \pi_0) \qquad \textit{(Contraction condition)} \tag{1.3}$$

and $\exists \Delta > 0$ s.t

$$d_T(P\mathcal{K}_0, P\mathcal{K}_\varepsilon) \leq \Delta \qquad \textit{(Bounded one-step error)} \tag{1.4}$$

Then the total variation distance between π_0 and π_ε is upper-bounded as follows

$$d_T(\pi_0, \pi_\varepsilon) \leq \frac{\Delta}{1 - \eta} \tag{1.5}$$

Proof. Let us consider a Markov chain with transition kernel $\mathcal{K}_\varepsilon$ initialised by a random probability measure P. Denote by $P^{(t)} := P\mathcal{K}_\varepsilon^t$ the distribution after t time steps. Thanks to (1.4) we can provide an upper-bound to the distance between $P^{(t+1)}$ and $P^{(t)}\mathcal{K}_0$:

$$d_T(P^{(t+1)}, P^{(t)}\mathcal{K}_0) = d_T(P^{(t)}\mathcal{K}_\varepsilon, P^{(t)}\mathcal{K}_0) \leq \Delta \tag{1.6}$$

Also, by (1.3), we have

$$d_T(P^{(t)}\mathcal{K}_0, \pi_0) \leq \eta d_T(P^{(t)}, \pi_0) \tag{1.7}$$

We can now derive an upper-bound for the distance between $P^{(t+1)}$ and π_0. By the triangle inequality,

$$\begin{aligned} d_T(P^{(t+1)}, \pi_0) &\leq d_T(P^{(t+1)}, P^{(t)}\mathcal{K}_0) + d_T(P^{(t)}\mathcal{K}_0, \pi_0) \\ &\leq \Delta + \eta d_T(P^{(t)}, \pi_0) \end{aligned} \tag{1.8}$$

Let $0 < r < 1 - \eta$ and let $\mathcal{B}(\pi_0, \frac{\Delta}{r}) \equiv \{P : d_T(P, \pi_0) < \frac{\Delta}{r}\}$ be the ball with centre π_0 and radius $\frac{\Delta}{r}$. Now consider the case when $P^{(t)} \notin \mathcal{B}(\pi_0, \frac{\Delta}{r})$ i.e $d_T(P^{(t)}, \pi_0) \geq \frac{\Delta}{r} \Leftrightarrow d_T(P^{(t)}, \pi_0)r \geq \Delta$. Hence, (1.8) is equivalent to

$$d_T(P^{(t+1)}, \pi_0) \leq r d_T(P^{(t)}, \pi_0) + \eta d_T(P^{(t)}, \pi_0) = (r + \eta) d_T(P^{(t)}, \pi_0) \tag{1.9}$$

with $r + \eta < 1$.

Note that (1.9) only holds for $P^{(t)}$ outside the ball. Hence, it does not imply convergence of $P^{(t)}$ to π_0 as $t \to \infty$. It suggests instead that $(P^{(t)})_t$ will get closer and closer to π_0 until it enters the ball after a final number of steps $t_0 \in \mathbb{N}$, as illustrated in Figure 1.1.

We now need to show that $(P^{(t)})_t$ does not leave the ball once it got inside, i.e that for all $t \geq t_0$, $P^{(t)} \in \mathcal{B}(\pi_0, \frac{\Delta}{r})$. We proceed by induction.

1. At step $t = t_0$, $P^{(t)} \in \mathcal{B}(\pi_0, \frac{\Delta}{r})$ by definition of t_0.

2. Let $t \in \mathbb{N}$, $t > t_0$ and assume $P^{(t)} \in \mathcal{B}(\pi_0, \frac{\Delta}{r}) \Leftrightarrow d_T(P^{(t)}, \pi_0) \leq \frac{\Delta}{r}$. Then

$$\begin{aligned} d_T(P^{(t+1)}, \pi_0) \leq \Delta + \eta d_T(P^{(t)}, \pi_0) &\leq \Delta + \eta \frac{\Delta}{r} \quad \text{by the induction hypothesis} \\ &= \frac{\Delta}{r}(\eta + r) \leq \frac{\Delta}{r} \\ &\Rightarrow P^{(t+1)} \in \mathcal{B}(\pi_0, \frac{\Delta}{r}) \end{aligned}$$

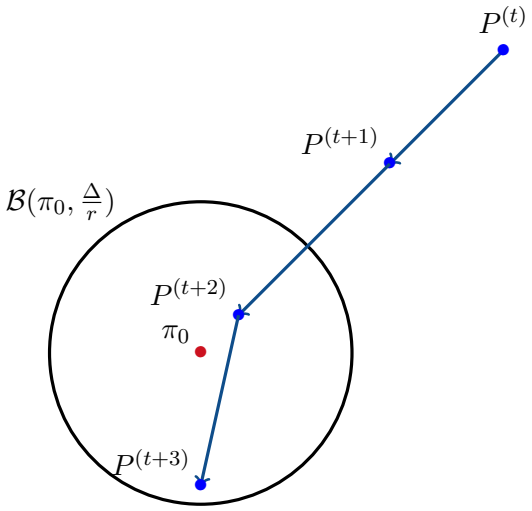
□

Hence, we have proved that $\forall t \geq t_0,\ P^{(t)} \in \mathcal{B}(\pi_0, \frac{\Delta}{r})$. Now, since $\{P^{(t)}\}_{t\in\mathbb{N}}$ converges to π_ε, then

$$\begin{aligned} &\pi_\varepsilon \in \mathcal{B}(\pi_0, \frac{\Delta}{r}) \\ \Leftrightarrow &d_T(\pi_\varepsilon, \pi_0) \leq \frac{\Delta}{r},\ \forall\, 0 < r < 1 - \eta \\ \Rightarrow &d_T(\pi_\varepsilon, \pi_0) \leq \frac{\Delta}{1-\eta} \end{aligned} \tag{1.10}$$

□

Figure 1.1: Illustration of equation (1.9)



Theorem 1.1.1. *Making the same assumptions as in Lemma 1.1.1, the total variation distance between π_0 and π_ε is upper bounded as follows :*

$$d_T(\pi_0, \pi_\varepsilon) \leq \frac{\Delta_{max}}{1-\eta}$$

where $\Delta_{max} \equiv \sup_{\theta,\theta'} |\alpha_\varepsilon(\theta, \theta') - \alpha_0(\theta, \theta')|$ and $\eta \in [0, 1)$

In the proof provided by Korattikara et al. (2014), we have spotted a mistake in the definition of the Metropolis-Hastings transition kernel which in turn leads to a flawed proof.

They indeed write this kernel as

$$\mathcal{K}_0(\theta, \theta') = \alpha_0(\theta, \theta')q(\theta'|\theta) + \delta_\theta(\theta')(1 - \alpha_0(\theta, \theta'))$$

Instead of

$$\mathcal{K}_0(\theta, \theta') = \alpha_0(\theta, \theta')q(\theta'|\theta) + \delta_\theta(\theta')(1 - r_0(\theta))$$

where

$$r_0(\theta) = \int_{\theta'} \alpha_0(\theta, \theta')q(\theta'|\theta)d\theta'$$

as can be seen on Figure A.1 in Appendix where we provide a screenshot of the proof given in the article. This is not simply a typo, as later in the proof it is clear they use α_0 instead or r_0.

However, the corrected proof we provide below does lead to the same result.

Proof. Now that Lemma (1.1.1) is proved, there only remains to show that for any probability measure P,

$$d_T(P\mathcal{K}_0, P\mathcal{K}_\varepsilon) = \frac{1}{2}\int_{\theta' \in \Theta} |(P\mathcal{K}_\varepsilon)(\theta') - (P\mathcal{K}_0)(\theta')|d\theta' \leq \Delta_{max} \tag{1.11}$$

where $\Delta_{max} := \sup_{\theta,\theta'} |\alpha_\varepsilon(\theta, \theta') - \alpha_0(\theta, \theta')|$.

In the ApMHT algorithm, the transition kernel $\mathcal{K}_\varepsilon$ is

$$\mathcal{K}_\varepsilon(\theta, \theta') = \alpha_\varepsilon(\theta, \theta')q(\theta'|\theta) + \delta_\theta(\theta')(1 - r_\varepsilon(\theta))$$

where we use the approximate acceptance probability α_ε instead of α_0 and r_ε is defined in a similar way as r_0.

Let $\Delta_\alpha := \alpha_\varepsilon - \alpha_0$ the error in the acceptance probability. Clearly, this quantity is upper-bounded by Δ_{max} as $|\Delta_\alpha| \leq \Delta_{max}$, by definition of Δ_{max}.

Proof of (1.11). For all P probability measure,

$$\begin{aligned}
&\frac{1}{2}\int_{\theta' \in \Theta} |(P\mathcal{K}_\varepsilon)(\theta') - (P\mathcal{K}_0)(\theta')| d\theta' \\
=&\frac{1}{2}\int_{\theta' \in \Theta} \left| \int_{\theta \in \Theta} (P\mathcal{K}_\varepsilon)(\theta, \theta') - (P\mathcal{K}_0)(\theta, \theta') dP(\theta) \right| d\theta' \\
=&\frac{1}{2}\int_{\theta' \in \Theta} \left| \int_{\theta \in \Theta} (\alpha_\varepsilon(\theta, \theta') q(\theta'|\theta) + \delta_\theta(\theta')(1 - r_\varepsilon(\theta)) - \alpha_0(\theta, \theta') q(\theta'|\theta) - \delta_\theta(\theta')(1 - r_0(\theta))) dP(\theta) \right| d\theta' \\
=&\frac{1}{2}\int_{\theta' \in \Theta} \left| \int_{\theta \in \Theta} q(\theta'|\theta) \Delta_\alpha(\theta, \theta') - \delta_\theta(\theta')(r_\varepsilon(\theta) - r_0(\theta)) dP(\theta) \right| d\theta' \\
\leq&\frac{1}{2}\int_{\theta' \in \Theta} \int_{\theta \in \Theta} q(\theta'|\theta) |\Delta_\alpha(\theta, \theta')| + \delta_\theta(\theta') |r_\varepsilon(\theta) - r_0(\theta)| dP(\theta) d\theta'
\end{aligned}$$

Where

$$\begin{aligned}
|r_\varepsilon(\theta) - r_0(\theta)| &= \left| \int_{\theta'} (\alpha_\varepsilon(\theta, \theta') - \alpha_0(\theta, \theta')) q(\theta'|\theta) d\theta' \right| \\
&\leq \int_{\theta'} |\alpha_\varepsilon(\theta, \theta') - \alpha_0(\theta, \theta')| q(\theta'|\theta) d\theta' \\
&\leq \Delta_{max} \int_{\theta'} q(\theta'|\theta) d\theta' \\
&= \Delta_{max}
\end{aligned}$$

Hence

$$\begin{aligned}
&\frac{1}{2}\int_{\theta' \in \Theta} |(P\mathcal{K}_\varepsilon)(\theta') - (P\mathcal{K}_0)(\theta')| d\theta' \\
\leq&\frac{1}{2}\Delta_{max} \int_{\theta' \in \Theta} \int_{\theta \in \Theta} q(\theta'|\theta) + \delta_\theta(\theta') dP(\theta) d\theta' \\
=&\Delta_{max}
\end{aligned}$$

□

One might be worried that Δ_{max} is large. However recall that if $\varepsilon \to 0$, then the batch

size m tends to N which in turn implies that $\alpha_\varepsilon \to \alpha_0$. Hence, provided we choose ε small enough, the upper-bound derived in Theorem 1.1.1 is guaranteed to be small. Under this condition, ApMHT should yield a good approximation of the posterior π_0.

1.2 Stochastic Gradient Langevin Dynamics (SGLD)

We now present the second of the algorithms we consider.

Stochastic Gradient Langevin Dynamics (SGLD, Welling and Teh (2011)) has two major assets when applied to Bayesian posterior sampling in the context of Big Data. Firstly, it only requires a fraction of size $m \ll N$ of the total data set at each iteration. Secondly, it can be tuned to become a rejection-free algorithm, ensuring that every candidate θ' is accepted at each step, which once again reduces the total computational cost as we bypass the evaluation of the acceptance probability α of standard Metropolis-Hastings (MH, Metropolis et al. (1953)).

This class of algorithms combines two major concepts, namely Langevin Monte Carlo theory (LMC, Roberts and Rosenthal (1998)) and stochastic gradient which we review below. LMC relies on the overdamped Langevin Îto diffusion, or Langevin equation.

$$d\theta_t = \frac{1}{2}\nabla_{\theta_t} \log \pi(\theta_t|z) + d\eta_t \tag{1.12}$$

where η_t is a d-dimensional Wiener Process.

However, the use of digital computers requires a discretised version of (1.12). Equation (1.13) can then be used as a proposal distribution in standard MH.

$$\theta_{t+1} = \theta_t + \frac{\varepsilon}{2}\nabla_{\theta_t} \log \pi(\theta_t|z) + \eta_t \tag{1.13}$$

where $\eta_t \sim \mathcal{N}(0, \varepsilon_t)$.

This discretisation process comes at the expense of the rejection-free property of exact LMC. Indeed, an expensive MH accept-reject step is essential to correct the discretisation error in order to target the true posterior. However, as $\varepsilon \to 0$, the discretisation error decreases so that the acceptance probability α tends back to 1, making the accept-reject step unnecessary, if an approximate target is acceptable.

SGLD is, in its original form, an asymptotically exact algorithm which builds upon the latter and introduces the use of mini-batches of data in order to further enhance computational efficiency when evaluating the gradient of the posterior. In fact, the update step of SGLD goes as follows.

Let $\mathcal{I} \subset \{1, ..., N\}$ be an index subset of size $m \ll N$,

$$\theta_{t+1} = \theta_t + \frac{\varepsilon_t}{2} \underbrace{\left(\nabla_{\theta_t} \log \pi(\theta_t) + \frac{N}{n} \sum_{i \in \mathcal{I}} \nabla_{\theta_t} \log p(z_i|\theta_t) \right)}_{\text{stochastic gradient}} + \underbrace{\eta_t}_{\text{injected noise}} \tag{1.14}$$

where $\eta_t \sim \mathcal{N}(0, \varepsilon_t)$ and

$$\sum_{t=1}^{\infty} \varepsilon_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \varepsilon_t^2 < \infty \tag{1.15}$$

In (1.15), the first condition ensures good mixing of the Markov chain whereas the second makes sure that the chain does converge to the model.

Welling and Teh (2011) give an informal intuition as to why SGLD produces samples that asymptotically approach the true posterior $\pi(\theta|z)$, i.e why (1.14) is equivalent to (1.13) as $t \to \infty$.

Stochasticity comes from two factors in (1.14), namely the stochastic gradient and the injected Gaussian noise η_t. It is easy to see that they respectively have a variance of $O(\varepsilon_t^2)$ and $O(\varepsilon_t)$. Hence, as $t \to \infty$, (1.15) implies that $\varepsilon_t \to 0$, which in turn implies that the injected noise dominates the stochastic gradient noise. Therefore, one should expect SGLD to sample from the true posterior approximately as $t \to \infty$. Besides, since $\varepsilon_t \to 0$, the acceptance probability α tends to 1 which makes the accept-reject step unnecessary.

However, in practice, it is common to use a small constant step-size $0 < \varepsilon \ll 1$ tuned in order to be big enough to guarantee good mixing and small enough to bypass the MH

accept-reject step. The algorithm is then no longer asymptotically exact.

Below is given a possible implementation of the algorithm.

Input : data z, ε, T, m, θ_0
Output: $\theta = \{\theta_t\}_{t=0}^{T}$ approximately drawn from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$
Initialization;
for $t = 0, ..., T$ **do**
    Draw minibatch of size m $\mathcal{X} \subset z$ without replacement;
    Draw $\eta \sim \mathcal{N}(0, \varepsilon^2)$;
    $\theta_{t+1} = \theta_t + \frac{\varepsilon}{2}\left(\nabla_{\theta_t} \log \pi(\theta_t) + \frac{N}{m}\nabla_{\theta_t} \log p(\mathcal{X}|\theta_t) + \eta\right)$;
end

Algorithm 3: 'Approximate' SGLD

1.3 Consensus Monte-Carlo

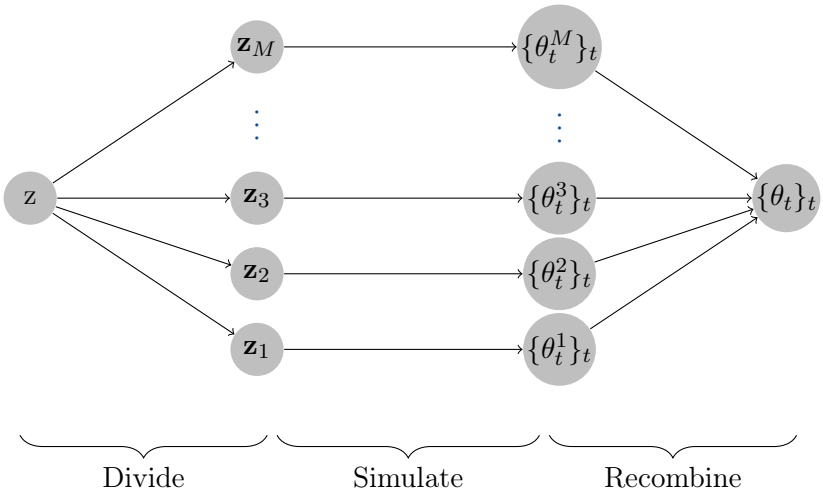
We now present the third algorithm reviewed in this study.

Parallelising MCMC computations is another strategy that can be implemented in order to bypass large likelihood evaluations. This class of methods splits the data into smaller batches and distribute them to several cores or machines. Assume, M machines are available. Making the assumption that the observations are conditionally independent given θ, the posterior distribution can be written as follows :

$$\pi(\theta|z) \propto \prod_{m=1}^{M} \pi(\theta)^{1/M} p(\mathbf{z}_m|\theta) \tag{1.16}$$

where $\{\mathbf{z}_1, ..., \mathbf{z}_M\}$ denote subsets of data set z and $\forall 1 \leq i, j \leq M$, $\mathbf{z}_i \cap \mathbf{z}_j = \emptyset$. Then, the user can run independently and simultaneously standard MCMC methods on batches of size $m \ll N$ if M is large enough, which should decrease the total computation time dramatically. Note that machine $m \in \{1, ..., M\}$ will generate samples $\{\theta_t^m\}_t$ from an algorithm targeting $\pi_m(\theta|z) \propto \pi(\theta)^{1/M} p(\mathbf{z}_m|\theta)$. This first step is a common feature of most algorithms using this so-called *divide-and-conquer* strategy (Bardenet et al. (2017)). They differ however on how to combine the M generated samples in order to obtain samples $\{\theta_t\}_t \sim \pi(\theta|z)$, i.e distributed according to the true posterior.

Figure 1.2: Parallel MCMC strategy



Scott et al. (2016) suggest using the *Consensus Monte-Carlo* algorithm. This method uses intensively the Bernstein-Von Mises theorem and its extension by Le Cam and Yang (2012) which states that under some regularity conditions, the posterior distribution is asymptotically independent from the prior and follows a normal distribution concentrated around the true value of the estimated parameter with variance equal to the inverse of the Fischer information matrix. In particular, one can make the assumption that $\theta|\mathbf{z}_m$, $m = 1, ..., M$ is normally distributed, provided that $|\mathbf{z}_m|$ is large enough.

We prove the following lemma 1.3.1 and property 1.3.1 which are at the core of Scott et al. (2016)'s algorithm and are left unproved in their article as a bit too obvious.

Lemma 1.3.1. *Suppose that $M = 2$. This can be assumed without loss of generality as the following method may be iterated. If*

$$\theta|\boldsymbol{z}_1 \sim \mathcal{N}(\mu_1, \Sigma_1) \quad \textit{and} \quad \theta|\boldsymbol{z}_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$$

then

$$\theta|z \sim \mathcal{N}(\mu, \Sigma) \tag{1.17}$$

where

$$\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \quad \textit{and} \quad \mu = \Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)$$

Proof.

$$\begin{aligned}
\pi(\theta|z) &\propto \pi(\theta|\mathbf{z}_1)\pi(\theta|\mathbf{z}_2) \\
&\propto \exp\left(-\frac{1}{2}\left[(\theta-\mu_1)^T\Sigma_1^{-1}(\theta-\mu_1) + (\theta-\mu_2)^T\Sigma_2^{-1}(\theta-\mu_2)\right]\right) \\
&\propto \exp\left(-\frac{1}{2}\left[\theta^T(\Sigma_1^{-1}+\Sigma_2^{-1})\theta - 2(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)^T\theta\right]\right) \text{ by symmetry of } \Sigma_1 \text{ and } \Sigma_2 \\
&\propto \exp\left(-\frac{1}{2}\left[(\theta - \Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2))\Sigma^{-1}(\theta - \Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2))^T\right]\right) \text{ by completing the square} \\
&\propto \exp\left(-\frac{1}{2}\left[(\theta-\mu))\Sigma^{-1}(\theta-\mu))^T\right]\right)
\end{aligned}$$

□

Property 1.3.1. *Let* $\theta_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ *and* $\theta_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$.
Then $\Sigma(\Sigma_1^{-1}\theta_1 + \Sigma_2^{-1}\theta_2) \sim \mathcal{N}(\mu, \Sigma)$

Proof. It is clear that the quantity of interest is normal, as a linear combination of normally-distributed variables. Now

- $\mathbb{E}[\Sigma(\Sigma_1^{-1}\theta_1 + \Sigma_2^{-1}\theta_2)] = \Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) = \mu$
- $\mathbb{V}\left(\Sigma(\Sigma_1^{-1}\theta_1 + \Sigma_2^{-1}\theta_2)\right) = \mathbb{V}\left(\Sigma\Sigma_1^{-1}\theta_1\right) + \mathbb{V}\left(\Sigma\Sigma_2^{-1}\theta_2\right) = \Sigma(\Sigma_1^{-1} + \Sigma_2^{-1})\Sigma = \Sigma \quad \square$

Hence, assuming the posterior is asymptotically normal, as is the case if Bernstein-Von Mises theorem applies, and the number of data points is large, then by Lemma 1.3.1 and Property 1.3.1, Scott et al. (2016) combine the obtained M samples by applying the following formula in order to target the full posterior :

Suppose machine m generates sample $\theta_{m1}, ..., \theta_{mT}$. Then $\forall t \in \{1, ..., T\}$,

$$\theta_t = \left(\sum_{m=1}^{M} \Sigma_m^{-1} \right)^{-1} \sum_{m=1}^{M} \Sigma_m^{-1} \theta_{mt} \tag{1.18}$$

Note that this is simply a weighted average of the subposterior samples with weights Σ_m^{-1}, $m = 1, ..., M$. In practice, only an estimate of the weights can be computed as seen in the algorithm below.

Input : samples $\theta_m = \{\theta_{mt}\}_{t=1}^{T}$ each from $\pi(\theta|z_s) \propto p(\mathbf{z}_m|\theta)\pi(\theta)^{1/M}$ where $\mathbf{z}_m$ denotes subsets of the data for $m = 1, ..., M$

Output: $\theta = \{\theta_t\}_{t=0}^{T}$ asymptotically from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$

Initialization;
$W_m \leftarrow \left(var(\{\theta_{mt}\}_{t=1}^{T})\right)^{-1}$ for $m = 1, ..., M$;
$W \leftarrow \sum_{m=1}^{M} W_m$;
$\theta \leftarrow 0$;
for $m = 1, ..., M$ **do**
  $\theta = \theta + W_m \theta_m$;
end
$\theta = W^{-1}\theta$

Algorithm 4: Consensus Monte Carlo

One can notice that if the posterior distribution is normal, then this algorithm targets exactly the posterior. However if the posterior is only assymptotically normal by the Bernstein-Von Mises theorem, then Consensus Monte-Carlo is only asymptotically exact.

1.4 Asymptotically exact sampling via non-parametric density product estimation

We now introduce the fourth algorithm of our study.

The *Non-Parametric Density Product Estimator* (NPDPE) algorithm described by Neiswanger et al. (2013) comes as a more flexible alternative to the *Consensus Monte-Carlo* described in Section 1.3. With *Consensus Monte-Carlo* one could expect substantial issues in cases where the Bernstein-Von Mises theorem does not hold. NPDPE however does not make such assumptions on the asymptotic behaviour of the posterior distribution.

As in Section 1.3, we rewrite the posterior distribution as

$$\pi(\theta|z) \propto \prod_{m=1}^{M} \pi(\theta)^{1/M} p(\mathbf{z}_m|\theta) \tag{1.19}$$

where $\{\mathbf{z}_1, ..., \mathbf{z}_M\}$ denote subsets of data set z and $\forall 1 \leq i, j \leq M$, $\mathbf{z}_i \cap \mathbf{z}_j = \emptyset$. Assuming we have obtained thanks to standard MCMC methods T samples $\{\theta_{t_m}^m\}_{t_m=1}^T$ for each subposterior π_m, Neiswanger et al. (2013) propose the following kernel estimator $\hat{\pi}_m$

$$\hat{\pi}_m(\theta) = \frac{1}{T} \sum_{t_m=1}^{T} \frac{1}{h^d} \mathcal{K}\left(\frac{||\theta - \theta_{t_m}^m||}{h}\right) \tag{1.20}$$

where $\mathcal{K}$ denotes any kernel function with bandwidth $h \in \mathbb{R}^+$. We will use from now on a standard Gaussian kernel, i.e

$$\hat{\pi}_m(\theta) = \frac{1}{T} \sum_{t_m=1}^{T} \mathcal{N}_d(\theta|\theta_{t_m}^m, h^2 I_d) \tag{1.21}$$

Now, the estimator of the posterior distribution suggested by Neiswanger et al. (2013) is simply the product of the subposterior estimators defined in (1.21), hence the name *Non-Parametric Density Product Estimator* :

$$
\begin{aligned}
\widehat{\pi_1 \ldots \pi_M}(\theta) &= \hat{\pi}_1(\theta) \ldots \hat{\pi}_M(\theta) \\
&= \frac{1}{T^M} \prod_{m=1}^{M} \sum_{t_m=1}^{T} \mathcal{N}_d(\theta | \theta_{t_m}^m, h^2 I_d) \\
&\propto \sum_{t_1=1}^{T} \ldots \sum_{t_M=1}^{T} \underbrace{\left[\prod_{m=1}^{M} \mathcal{N}_d \left(\theta_{t_m}^m \middle| \frac{1}{M} \sum_{m=1}^{M} \theta_{t_m}^m, h^2 I_d \right) \right]}_{\text{weight}} \mathcal{N}_d \left(\theta \middle| \frac{1}{M} \sum_{m=1}^{M} \theta_{t_m}^m, \frac{h^2}{M} I_d \right)
\end{aligned}
\tag{1.22}
$$

which can be derived through a straightforward but tedious computation.

Although (1.22) is a large mixture of T^M normal distributions, one can sample rather easily from it. One should first sample a mixture weight by using a Metropolis within Gibbs sampler and then sampling from the normal distribution associated with the sampled weight. A complete description of this procedure is given in Algorithm 5.

In practice, we find that tuning the bandwidth h to a constant small number yields more efficient mixing. Also, in order to prevent computational stability issues, we find that the accept-reject step performed in line `23` is much more reliable when the ratio of weights is log-transformed. Hence, we recommend that line `23` be written as follows :

$$
\texttt{if } \log(u) < \log(w_c) - log(w_{\tilde{t}}) \texttt{ then}
$$

Input : samples $\theta_m = \{\theta_{mt}\}_{t=1}^{T}$ each from $\pi(\theta|z_s) \propto p(\mathbf{z}_m|\theta)\pi(\theta)^{1/M}$ where $\mathbf{z}_m$ denotes subsets of the data for $m = 1, ..., M$

Output: $\theta = \{\theta_t\}_{t=0}^{T}$ asymptotically from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$

Initialization;
$\tilde{t} = \{t_1, ..., t_M\} \overset{i.i.d}{\sim} \mathcal{U}_{\{1,...,T\}}$;
for $i = 1, ..., T$ **do**
  $h \leftarrow i^{-1/(4+d)}$;
  for $m = 1, .., M$ **do**
    $c \leftarrow \tilde{t}$;
    $c_m \sim \mathcal{U}_{\{1,...,T\}}$;
    $\bar{\theta}_{\tilde{t}} \leftarrow 0$;
    $\bar{\theta}_c \leftarrow 0$;
    $w_{\tilde{t}} \leftarrow 1$;
    $w_c \leftarrow 1$;
    for $j = 1, ..., m$ **do**
      $\bar{\theta}_{\tilde{t}} \leftarrow \bar{\theta}_{\tilde{t}} + \theta^j_{\tilde{t}_j}$;
      $\bar{\theta}_c \leftarrow \bar{\theta}_c + \theta^c_{c_j}$;
    end
    $\bar{\theta}_{\tilde{t}} \leftarrow \frac{1}{M}\bar{\theta}_{\tilde{t}}$;
    $\bar{\theta}_c \leftarrow \frac{1}{M}\bar{\theta}_c$;
    for $j = 1, ..., m$ **do**
      $w_{\tilde{t}} \leftarrow w_t \times \mathcal{N}(\theta^j_{\tilde{t}_j}|\bar{\theta}_{\tilde{t}}, h^2 I_d)$;
      $w_c \leftarrow w_c \times \mathcal{N}(\theta^j_{c_j}|\bar{\theta}_c, h^2 I_d)$;
    end
    $u \sim \mathcal{U}_{[0,1]}$;
    if $u < w_c/w_{\tilde{t}}$ **then**
      $\tilde{t} \leftarrow c$;
      $\bar{\theta}_{\tilde{t}} \leftarrow \bar{\theta}_c$
    end
  end
  $\theta_i \sim \mathcal{N}_d(\bar{\theta}_{\tilde{t}}, \frac{h^2}{M} I_d)$
end
$\theta = W^{-1}\theta$

Algorithm 5: NPDPE

It can be shown that the density product estimator $\widehat{\pi_1...\pi_M}$ of $\pi_1...\pi_M$ is consistent in Mean Squared error i.e that $MSE(\widehat{\pi_1...\pi_M}) \longrightarrow 0$ as $T \longrightarrow \infty$. Define the Hölder class $\Sigma(\beta, C)$, for $\beta \geq 1$ and $C > 0$ to be the set of functions $f : \mathcal{X} \longmapsto \mathbb{R}$ l-times differentiable for $l = \lfloor \beta \rfloor$ such that

$$|f^{(l)}(\theta) - f^{(l)}(\theta')| \leq C||\theta - \theta'||^{\beta - l}, \quad \forall \theta, \theta' \in \mathcal{X}$$

Let $\mathcal{P}(\beta, C) = \left\{p \in \Sigma(\beta, C) | p \geq 0,\ \int p(\theta)d\theta = 1\right\}$

Assume that for all $1 \leq m \leq M$, subposterior $\pi_m \in \mathcal{P}(\beta, C)$ and $\exists b > 0$ s.t $\pi_m(\theta) \leq b$, $\forall \theta \in \mathcal{X}$. Neiswanger et al. (2013) first derive an upper bound for the bias of their estimator $\widehat{\pi_1 ... \pi_M}$ of $\pi_1 ... \pi_M$.

The authors state and prove the following lemma. Our contribution is the explicit derivation of the upper-bound for $|\mathbb{E}[\hat{\pi}_m(\theta)] - \pi_m(\theta)|$.

Lemma 1.4.1.

$$\sup_{\pi_1,...,\pi_M \in \mathcal{P}(\beta,C)} |\mathbb{E}[\widehat{\pi_1 ... \pi_M}(\theta)] - \pi_1 ... \pi_M(\theta)| \leq \sum_{m=1}^{M} c_m h^{m\beta}$$

for some $c_1, ..., c_M > 0$

Proof. Let $\pi_1, ..., \pi_m \in \mathcal{P}(\beta, C)$,

$$\begin{aligned}
|\mathbb{E}[\widehat{\pi_1 ... \pi_M}(\theta)] - \pi_1 ... \pi_M(\theta)| &= |\mathbb{E}[\hat{\pi}_1(\theta) ... \hat{\pi}_M(\theta)] - \pi_1 ... \pi_M(\theta)| \\
&= |\mathbb{E}[\hat{\pi}_1(\theta)] ... \mathbb{E}[\hat{\pi}_M(\theta)] - \pi_1 ... \pi_M(\theta)|
\end{aligned}$$

Now,

$$\begin{aligned}
|\mathbb{E}[\hat{\pi}_m(\theta)] - \pi_m(\theta)| &= \left| \frac{1}{T} \sum_{t_m=1}^{T} \int \mathcal{N}(\theta | \theta_{t_m}^m, h^2 I_d) \pi_m(\theta_{t_m}^m) d\theta_{t_m}^m - \pi_m(\theta) \right| \\
&= \left| \frac{1}{T} \sum_{t_m=1}^{T} \int \mathcal{N}(\theta | \theta_{t_m}^m, h^2 I_d) (\pi_m(\theta_{t_m}^m) - \pi_m(\theta)) d\theta_{t_m}^m \right| \\
&= c \left| \sum_{t_m=1}^{T} \int \exp\left\{ \frac{1}{2} \left(\frac{||\theta_{t_m}^m - \theta||}{h} \right)^2 \right\} (\pi_m(\theta_{t_m}^m) - \pi_m(\theta)) d\theta_{t_m}^m \right|, \ c > 0
\end{aligned}$$

Let $y_m = \frac{\theta^m_{t_m} - \theta}{h}$. Then,

$$\begin{aligned}
|\mathbb{E}[\hat{\pi}_m(\theta)] - \pi_m(\theta)| &= \tilde{c}\left|\sum_{m=1}^{T}\int \exp\left\{\frac{1}{2}||y_m||^2\right\}(\pi_m(\theta + hy_m) - \pi_m(\theta))dy_m\right|,\ \tilde{c} > 0 \\
&\leq \tilde{c}\sum_{m=1}^{T}\int \exp\left\{\frac{1}{2}||y_m||^2\right\}|\pi_m(\theta + hy_m) - \pi_m(\theta)|\, dy_m \\
&\leq \tilde{c}Ch^{\beta}\sum_{m=1}^{T}\int \exp\left\{\frac{1}{2}||y_m||^2\right\}|y_m| dy_m,\ \text{since } \pi_m \in \mathcal{P}(\beta, C) \\
&\leq c'_m h^{\beta},\ c'_m > 0, \text{since the moments of a Gaussian are bounded.}
\end{aligned}$$

Hence,

$$\begin{aligned}
|\mathbb{E}[\widehat{\pi_1...\pi_M}(\theta)] - \pi_1...\pi_M(\theta)| &= |\mathbb{E}[\hat{\pi}_1(\theta)]...\mathbb{E}[\hat{\pi}_M(\theta)] - \pi_1...\pi_M(\theta)| \\
&\leq |(\pi_1(\theta) + c'_1h^{\beta})...(\pi_M(\theta) + c'_Mh^{\beta}) - \pi_1...\pi_M(\theta)| \\
&\leq |c_1h^{\beta} + ... + c_Mh^{M\beta}| \text{ where } c_1, ..., c_M{>}0 \\
&\leq |c_1h^{\beta}| + ... + |c_Mh^{M\beta}| \\
&= \sum_{m=1}^{M} c_m h^{m\beta}
\end{aligned}$$

□

The authors then derive an upper bound for the variance of their density product estimator. Our contribution to this proof is the explicit derivation of an upper-bound for $\mathbb{V}[\hat{\pi}_m(\theta)]$, $\forall\ 1 \leq m \leq M$.

Lemma 1.4.2.

$$\sup_{\pi_1,...,\pi_M \in \mathcal{P}(\beta,C)} \mathbb{V}[\widehat{\pi_1...\pi_M}(\theta)] \leq \sum_{m=1}^{M}\binom{M}{m}\frac{c_m}{T^m}h^{dm}$$

for some $c_1, ..., c_M > 0$

Proof. Let us first derive an upper-bound for $\mathbb{V}[\hat{\pi}_m(\theta)]$, $\theta \in \mathcal{X}$, $1 \leq m \leq M$.

$$\mathbb{V}[\hat{\pi}_m(\theta)] \leq \mathbb{E}[\hat{\pi}_m(\theta)^2] = c\frac{1}{Th^{2d}} \sum_{t_m=1}^{T} \int \exp\left\{ \left(\frac{||\theta_{t_m}^m - \theta||}{h} \right)^2 \right\} \pi_m(\theta_{t_m}^m) d\theta_{t_m}^m, \ c > 0$$

Let $y_m = \frac{\theta_{t_m}^m - \theta}{h}$. Then,

$$\begin{aligned}
\mathbb{V}[\hat{\pi}_m(\theta)] &= c\frac{h^d}{Th^{2d}} \sum_{m=1}^{T} \int \exp\left\{||y_m||^2\right\} \pi_m(hy_m + \theta) dy_m \\
&\leq c\frac{\sup_\theta \pi_m(\theta)}{Th^d} \sum_{m=1}^{T} \int \exp\left\{||y_m||^2\right\} dy_m \\
&\leq \frac{c'_m}{Th^d}
\end{aligned}$$

Now,

$$\begin{aligned}
\mathbb{V}[\widehat{\pi_1 ... \pi_m}(\theta)] &= \mathbb{E}[\hat{\pi}_1(\theta)^2] ... \mathbb{E}[\hat{\pi}_M(\theta)^2] - \mathbb{E}[\hat{\pi}_1(\theta)]^2 ... \mathbb{E}[\hat{\pi}_M(\theta)]^2 \\
&= \left(\prod_{m=1}^{M} \mathbb{V}[\hat{\pi}_m] + \mathbb{E}[\hat{\pi}_m]^2 \right) - \left(\prod_{m=1}^{M} \mathbb{E}[\hat{\pi}_m]^2 \right) \\
&\leq \sum_{m=0}^{M} \binom{M}{m} \frac{\tilde{c}^m c^{M-m}}{T^{M-m} h^{d(M-m)}} - \tilde{c}^M
\end{aligned}$$

where $\tilde{c} = \max\{c_1, ..., c_M\}$, $\{c_1, ..., c_M\}$ being the upper-bounds derived in lemma 1.4.1 and $c = \max\{c'_1, ..., c'_M\}$

$$\begin{aligned}
&= \sum_{m=0}^{M-1} \binom{M}{m} \frac{\tilde{c}^m c^{M-m}}{T^{M-m} h^{d(M-m)}} \\
&\leq \sum_{m=1}^{M} \binom{M}{m} \frac{c_m}{T^m h^{dm}}, \text{where } c_m > 0 \ \forall m
\end{aligned}$$

□

Neiswanger et al. (2013) then use results from lemma 1.4.1 and 1.4.2 to derive an upper bound on the Mean-Squared Error (MSE)

Proposition 1.4.1. *If* $h \asymp T^{-1/(2\beta+d)}$ *then*

$$\sup_{\pi_1,...,\pi_M \in \mathcal{P}(\beta,C)} MSE(\widehat{\pi_1...\pi_M}(\theta)) = \sup_{\pi_1,...,\pi_M \in \mathcal{P}(\beta,C)} \mathbb{E}\left[\int (\widehat{\pi_1...\pi_M}(\theta) - \pi_1...\pi_M(\theta))^2 d\theta\right]$$
$$\leq \frac{c}{T^{2\beta/(2\beta+d)}} \tag{1.23}$$

for $c > 0$ *and* $0 < h \leq 1$

Proof. For $\pi_1, ..., \pi_M \in \mathcal{P}(\beta, C)$, using the bias-variance decomposition of the MSE,

$$\mathbb{E}\left[\int (\widehat{\pi_1...\pi_M}(\theta) - \pi_1...\pi_M(\theta))^2 d\theta\right] \leq \left(\sum_{m=1}^{M} c_m h^{mb}\right)^2 + \sum_{m=1}^{M} \binom{M}{m} \frac{\tilde{c}_m}{T^m h^{dm}} \text{ by lemma 1.4.1 and 1.4.2}$$
$$\leq kT^{-2\beta/(2\beta+d)} + \frac{\tilde{k}}{T^{1-d(2\beta+d)}}, \text{ for } k, \tilde{k} > 0$$
$$\leq \frac{c}{T^{2\beta/(2\beta+d)}}$$

Hence the estimator $\widehat{\pi_1...\pi_M}$ is consistent since its MSE tends to 0 as T, the number of samples, increases.

1.5 Pseudo-Marginal MCMC

We now introduce the last algorithm of this dissertation.

1.5.1 General setting

In this section, we will review a large class of algorithms called *pseudo-marginal MCMC*. These methods are called *pseudo* since they use an estimator $\hat{L}(z|\theta, u)$ of the likelihood. The auxiliary random variable u is used in order to derive the likelihood estimator but is then integrated out of the posterior. Hence the name *pseudo-marginal*.
One natural way to derive such an estimator is to recall that the log-likelihood of conditionally independent observations can be written as

$$l(\theta) = \sum_{i=1}^{N} l_i(\theta), \text{ where } l_i(\theta) := \log p(z_i|\theta) \tag{1.24}$$

and sample uniformly at random m indices $u_1, ..., u_m$ in $\{1, ..., N\}$ where $m \ll N$ to build the unbiased estimator

$$\hat{l}(\theta, u) = \frac{N}{m} \sum_{i=1}^{m} l_{u_i}(\theta) \tag{1.25}$$

However, this simple technique generally yields a large variance. Besides, some of the l_i's should bring a significantly larger contribution to (1.24) than others. Hence, they should be attributed a higher probability of being drawn. However, this would require evaluating N likelihood weights at each iteration which would cancel the computational gains of using $\hat{l}(\theta, u)$ instead of $l(\theta)$. The idea of Quiroz et al. (2018) is to use some control variates $q_i(\theta)$ in order to make the variances of the l_i's comparable so that drawing them uniformly at random yields a good estimator. The derivation of the q_i's will be explained in the next subsection.

Hence, the estimator proposed by Quiroz et al. (2018) is

$$\hat{l}(\theta,u) := q(\theta) + N\hat{\mu}(\theta,u) \text{ where } q(\theta) := \sum_{i=1}^{N} q_i(\theta) \text{ and } \hat{\mu}(\theta,u) := \frac{1}{m}\sum_{i=1}^{m} l_{u_i}(\theta) - q_{u_i}(\theta) \tag{1.26}$$

We give the details (which are brief) for the proofs ofLemmas 1.5.1 and 1.5.2 as well as Properties 1.5.1 and 1.5.2.

Lemma 1.5.1. *$\hat{l}(\theta,u)$ is an unbiased estimator of $l(\theta)$*

Proof.

- $\mathbb{E}_u[l_{u_i}(\theta)] = \sum_{u=1}^{N} \mathbb{P}(u_i = u) l_u(\theta) = \frac{1}{N}\sum_{u=1}^{N} l_u(\theta) = \frac{1}{N} l(\theta)$
- Similarly $\mathbb{E}_u[q_{u_i}(\theta)] = \frac{1}{N} q(\theta)$ by (1.26)
- Hence $\mathbb{E}_u[\hat{l}(\theta,u)] = q(\theta) + \frac{N}{m}\sum_{i=1}^{m} \frac{1}{N}(l(\theta) - q(\theta)) = l(\theta)$ □

Lemma 1.5.2. $\sigma^2 := Var(\hat{l}(\theta,u)) = \frac{N^2\sigma_\mu^2}{m}$
where $\sigma_\mu^2 = Var(l_{u_i}(\theta) - q_{u_i}(\theta)) = \frac{1}{N}\sum_{i=1}^{N}(l_i(\theta) - q_i(\theta) - \frac{1}{N}(l(\theta) - q(\theta)))^2$

Proof.

$$Var(\hat{l}(\theta,u)) = N^2 Var(\hat{\mu}(\theta,u)) = \frac{N^2\sigma_\mu^2}{m}$$

□

Lemma 1.5.3. *Asymptotically, when $m \to \infty$ and N is fixed, $\hat{l}(\theta,u) \sim \mathcal{N}\left(l(\theta), \frac{N^2\sigma_\mu^2}{m}\right)$, assuming $\sigma_\mu^2 < \infty$.*

Proof. Standard application of the Central Limit Theorem (CLT) as the u_i's are i.i.d and $\sigma_\mu^2 < \infty$ □

Thus Quiroz et al. (2018) have found an unbiased estimator $\hat{l}(\theta, u)$ for the log-likehood $l(\theta)$, with a variance they can attempt to control using q. Now, deriving an estimator of the likelihood function $L(\theta) = \exp\{l(\theta)\}$ is not straightforward since $\mathbb{E}[\exp\{\hat{l}(\theta, u)\}] \neq \exp\{\mathbb{E}[\hat{l}(\theta, u)]\}$ in general. They propose the following estimator following Ceperley and Dewing (1999) and Nicholls et al. (2012) :

$$\hat{L}(\theta, u) := \exp\left(\hat{l}(\theta, u) - \frac{N^2}{2m}\sigma_\mu^2\right) \tag{1.27}$$

Proposition 1.5.1. *The estimator of the likelihood $\hat{L}(\theta, u)$ defined in (1.27) is unbiased.*

Proof.

$$\mathbb{E}\left[\hat{L}(\theta, u)\right] = \frac{\mathbb{E}\left[\exp\left(\hat{l}(\theta, u)\right)\right]}{\exp\left(\frac{N^2}{2m}\sigma_\mu^2\right)}$$

Now, recall that if $\log(X) \sim \mathcal{N}(\mu, \sigma^2)$ then $X \sim \log\mathcal{N}(\mu, \sigma^2)$ and $\mathbb{E}[X] = \exp(\mu + \frac{\sigma^2}{2})$. Hence,

$$\mathbb{E}\left[\hat{L}(\theta, u)\right] = \frac{\exp\left(l(\theta) + \frac{N^2}{2m}\sigma_\mu^2\right)}{\exp\left(\frac{N^2}{2m}\sigma_\mu^2\right)} = L(\theta)$$

□

In practice, computing exactly σ_μ^2 is not feasible as it must be computed at every observation z_i, which is what we want to bypass. Thus, Quiroz et al. (2018) use the following estimate instead.

$$\hat{\sigma}_\mu^2 = \frac{1}{m}\sum_{i=1}^{m}\left(l_{u_i}(\theta) - q_{u_i}(\theta) - \frac{1}{m}\sum_{i=1}^{m}\left(l_{u_i}(\theta) - q_{u_i}(\theta)\right)\right)^2$$

Hence, the estimate derived in (1.27) where we plug the estimate $\hat{\sigma}_\mu^2$ is only expected to be "nearly unbiased".

We now show why *pseudo-marginal* MCMC with an unbiased estimator of the likelihood targets the true posterior distribution $\pi(\theta|z)$.

Proposition 1.5.2. *Let $\pi_u(.)$ and $\hat{\pi}(\theta, u|z)$ denote the prior distribution on u and the target posterior distribution when using the likelihood estimate (1.27). Then the marginal distribution of the sampled θ's, $\int_u \hat{\pi}(\theta, u|z)du$ is equal to the true posterior distribution $\pi(\theta|z)$.*

Proof.

$$\hat{\pi}(\theta, u|z) = \frac{\hat{L}(\theta, u)\pi(\theta)\pi_u(u)}{m(z)} \tag{1.28}$$

where

$$m(z) = \int_\theta \int_u \hat{L}(\theta, u)\pi(\theta)\pi_u(u)dud\theta = \int_\theta \pi(\theta) \int_u \hat{L}(\theta, u)\pi_u(u)dud\theta = \int_\theta \pi(\theta)L(\theta)d\theta$$

which is simply the usual marginal distribution of z, also called *evidence*, thanks to the unbiasedness property of the likelihood estimator.

Hence we can write

$$m(z) = \frac{L(\theta)\pi(\theta)}{\pi(\theta|z)}, \text{ where } \pi(\theta|z) \text{ denotes the true posterior.} \tag{1.29}$$

Plugging (1.29) into (1.28) we get

$$\begin{aligned} \hat{\pi}(\theta, u|z) &= \frac{\hat{L}(\theta, u)\pi(\theta)\pi_u(u)\pi(\theta|z)}{L(\theta)\pi(\theta)} \\ \hat{\pi}(\theta, u|z) &= \frac{\hat{L}(\theta, u)\pi_u(u)\pi(\theta|z)}{L(\theta)} \end{aligned} \tag{1.30}$$

Now, integrating out u from (1.30),

$$\int_u \hat{\pi}(\theta, u|z)du = \frac{\pi(\theta|z)}{L(\theta)} \int_u \hat{L}(\theta, u)\pi_u(u)du = \pi(\theta|z) \tag{1.31}$$

by the unbiasedness of $\hat{L}(\theta, u)$.

□

1.5.2 Choice of control variates $q_i(\theta)$'s

Assume we divide the dataset into K clusters C_k for $k = 1, ..., K$ and derive the centroids $z_k^c \in \mathbb{R}^{d\times 1}$ where d denotes the dimension of parameter θ. Quiroz et al. (2018) suggest constructing q_i as a second order Taylor approximation of l_i with respect to z_i around the centroids $\{z_k^c\}_k$. This gives

$$q_i(\theta) = q(z_i; \theta) = l(z_k^c; \theta) + \nabla_z l(z_k^c; \theta)(z_i - z_k^c) + \frac{1}{2}(z_i - z_k^c)^T H(z_k^c : \theta)(z_i - z_k^c), \tag{1.32}$$

for z_i belonging to cluster k, where $l(z_i; \theta) := l_i(\theta) = \log p(z_i|\theta)$ and H denotes the Hessian of the log-likelihood function at z_k^c.

Quiroz et al. (2018) provide an efficient way to compute $q(\theta) = \sum_{i=1}^N q_i(\theta)$, which is necessary in order to estimate $\hat{l}(\theta, u)$. First of all, $\sum_{i=1}^N q_i(\theta)$ is equal to

$$\sum_{k=1}^K \sum_{i\in C_k} l(z_k^c; \theta) + \sum_{k=1}^K \sum_{i\in C_k} \nabla_z l(z_k^c; \theta)(z_i - z_k^c) + \frac{1}{2}\sum_{k=1}^K \sum_{i\in C_k} (z_i - z_k^c)^T H(z_k^c; \theta)(z_i - z_k^c) \tag{1.33}$$

The authors show that each term can be simplified to a simple sum which does not require going through the whole data set.

- First term :

$$\sum_{k=1}^{K} \sum_{i \in C_k} l(z_k^c; \theta) = \sum_{k=1}^{K} |C_k| l(z_k^c; \theta)$$

- Second term :

$$\sum_{k=1}^{K} \sum_{i \in C_k} \nabla_z l(z_k^c; \theta)(z_i - z_k^c) = \sum_{k=1}^{K} \nabla_z l(z_k^c; \theta) \sum_{i \in C_k} (z_i - z_k^c) = 0$$

- Third term : let $b_i = (z_i - z_k^c) \in \mathbb{R}^{d \times 1}$ and $H^{(k)} = H(z_k^c; \theta)$. Then

$$b_i^T H^{(k)} b_i = \sum_{s,t} H_{st}^{(k)} b_{is} b_{it}, \text{ by definition of a quadratic form.}$$

  Hence,

$$\sum_{k=1}^{K} \sum_{i \in C_k} b_i^T H^{(k)} b_i = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{s,t} H_{st}^{(k)} b_{is} b_{it} = \sum_{s,t} \left(\sum_{k=1}^{K} H_{st}^{(k)} \sum_{i \in C_k} b_{is} b_{it} \right)$$

  Define $B^{(k)} \in \mathbb{R}^{d \times d}$ to be a matrix with elements $\{\sum_{i \in C_k} b_{ij} b_{ik}\}_{jk}$. Then,

$$\sum_{k=1}^{K} \sum_{i \in C_k} b_i^T H^{(k)} b_i = \sum \text{vec} \left(\sum_{k=1}^{K} H^{(k)} \odot B^{(k)} \right)$$

  where $\odot$ denotes the element-wise multiplication or Hadamard product and vec(A) denotes the sum of the elements of any matrix A.

Note that since the $b^{(k)}$ do not depend on θ, they can be computed before the first iteration only once. Thanks to this simplification, Quiroz et al. (2018) prevent heavy computations by limiting the evaluation of the control variates to sums of K elements where $K \ll N$ in general.

The authors also give a complete derivation of the gradient and Hessian matrix of the likelihood evaluated at the data for GLMs. In these computations, we found a typo that leads to calculation errors. The interested reader can refer to the Appendix.

Input : data z, C_k and c_k clusters and centroids of data for $k = 1, ..., K$, T, m, θ_0
Output: $\theta = \{\theta_t\}_{t=0}^{T}$ asymptotically from $\pi(\theta|z) \propto p(z|\theta)\pi(\theta)$

Initialization;
for $k = 1, ..., K$ **do**
  $B_k = 0_{d,d}$, null square matrix of dimension d;
  for $i \in C_k$ **do**
    $b \leftarrow z_i - c_k$;
    $B_k = B_k + bb^t$;
  end
end
for $t = 0, ..., T$ **do**
  Draw without replacement minibatch $\mathcal{X}$ of size m;
  Draw θ_p from symmetric proposal distribution;
  for $i = 1, ..., m$ **do**
    $k \leftarrow \{k : \mathcal{X}_i \in C_k\}$;
    $d_{\theta_{t_i}} = \log p(\mathcal{X}_i|\theta_t) - \log p(c_k|\theta_t) - \nabla_z \log p(c_k|\theta_t)(z_i - c_k) -$
    $\frac{1}{2}(\mathcal{X}_i - c_k)^t H_z(c_k|\theta_t)(\mathcal{X}_i - c_k)$;
    $d_{\theta_{p_i}} = \log p(\mathcal{X}_i|\theta_p) - \log p(c_k|\theta_p) - \nabla_z \log p(c_k|\theta_p)(z_i - c_k) -$
    $\frac{1}{2}(\mathcal{X}_i - c_k)^t H_z(c_k|\theta_p)(\mathcal{X}_i - c_k)$;
  end
  $\bar{\theta}_t \leftarrow \frac{1}{m}\sum d_{\theta_{t_i}}$;
  $\sigma_t^2 \leftarrow \frac{1}{m}\sum (d_{\theta_{t_i}} - \bar{\theta}_t)^2$;
  $\bar{\theta}_p \leftarrow \frac{1}{m}\sum d_{\theta_{p_i}}$;
  $\sigma_p^2 \leftarrow \frac{1}{m}\sum (d_{\theta_{p_i}} - \bar{\theta}_p)^2$;
  $l_{\theta_t} \leftarrow 0$;
  $l_{\theta_p} \leftarrow 0$;
  $h_{\theta_t} \leftarrow 0$;
  $h_{\theta_p} \leftarrow 0$;
  for $k = 1, ..., K$ **do**
    $N_k = |C_k|$;
    $l_{\theta_t} \leftarrow l_{\theta_t} + N_k \log p(c_k|\theta_t)$;
    $l_{\theta_p} \leftarrow l_{\theta_p} + N_k \log p(c_k|\theta_p)$;
    $h_{\theta_t} \leftarrow h_{\theta_t} + H_z(c_k|\theta_t) \odot B_k$;
    $h_{\theta_p} \leftarrow h_{\theta_p} + H_z(c_k|\theta_p) \odot B_k$;
  end
  $q_{\theta_t} \leftarrow l_{\theta_t} + \frac{1}{2}\sum_i \sum_j h_{\theta_{t_{i,j}}}$;
  $q_{\theta_p} \leftarrow l_{\theta_p} + \frac{1}{2}\sum_i \sum_j h_{\theta_{p_{i,j}}}$;
  $\hat{l}_{\theta_t} \leftarrow q_{\theta_t} + N\bar{\theta}_t - \frac{N^2}{2m}\sigma_t^2$;
  $\hat{l}_{\theta_p} \leftarrow q_{\theta_p} + N\bar{\theta}_p - \frac{N^2}{2m}\sigma_p^2$;
  $\alpha \leftarrow \hat{l}_{\theta_p} + \log \pi(\theta_p) - \hat{l}_{\theta_t} - \log \pi(\theta_t)$;
  $u \sim U_{[0,1]}$;
  if $\alpha \geq \log(u)$ **then**
    $\theta_{t+1} = \theta_p$;
  else
    $\theta_{t+1} = \theta_t$;
  end
end

Algorithm 6: Pseudo-Marginal MCMC

Chapter 2

Experiments

In this section, we will attempt to assess the algorithms discussed in Chapter 1, namely three approximate sub-sampling algorithms (ApMHT, Pseudo-Marginal MCMC and SGLD) and two parallel MCMC algorithms (Consensus-Monte Carlo and NPDPE). Our assessment method is described in the next section.

2.1 Methodology

For each synthetic model we consider, our main concerns are to know whether the algorithms do target the right posterior distribution, how accurate is their approximation of the latter, and finally how efficient they are.

2.1.1 Accuracy of the algorithms

Since it would not be sensible to compare on an equal footing two algorithms whose qualities of approximation of the posterior distribution are significantly different, we have to measure how accurate each algorithm is.

Therefore, we will generate a *reference* posterior distribution that we will abusively call *true posterior* from a very large number of exact Metropolis-Hastings samples. This will

allow us to estimate how close our algorithms' outputs are to the *true posterior* thanks to a discrepancy measure, which we choose to be the Hellinger distance (Hellinger (1909)) defined as follows.

Let P and Q be two probability measures absolutely continuous w.r.t the Lebesgue measure μ. Then the Hellinger distance between P and Q is

$$H(P,Q) = \frac{1}{2}\int(\sqrt{p}-\sqrt{q})^2 d\mu$$

Where p and q are the Radon-Nikodym derivatives (or probability density functions) of P and Q respectively.

We could have used other popular metrics such as the K-L divergence. However, the Hellinger distance, as a distance, is symmetric and easier to interpret.

2.1.2 Efficiency of the algorithms

We will use the Effective Sample Size (ESS) as a measure of efficiency. It is defined as

$$ESS = \frac{T}{1+2\sum_{k=1}^{T-1}\rho(k)}$$

where $\rho(k)$ denotes the correlation at lag k and T is the length of the Markov chain.

Hence the ESS is the number of independent samples that would give the same variance reduction as our T correlated MCMC samples: the larger the ESS for a given T, the more efficient is the algorithm.

However, we also need a measure of the computational cost of each algorithm. It is tempting to choose the running time for the sake of clarity and ease of comprehension. Yet, it might not be fair to do so as our implementation of the algorithms is probably not optimal. Hence we use the number of full likelihood evaluations required to perform each algorithm.

We call a "likelihood evaluation" an evaluation of the likelihood function (or log-likelihood) on a single data point. We then divide the total number of likelihood evaluations by N, the data set size, so that 1 *standardised likelihood evaluation*, or full likelihood evaluation, is equivalent to the evaluation of the likelihood function on the whole data set. For example, when $T = 20000$, our measure of computational cost for Metropolis-Hastings will be T runs * N single-point likelihood evaluations$/N = 20000$ (cf Algorithm 1).

Hence, we set our efficiency assessment criterion to be the ESS per standardised likelihood evaluation, as it will penalise methods that require many likelihood evaluations and reward those which yield a large number of independent samples.

On deriving the number of standardised likelihood evaluations.

We provide a summary table giving the number of standardised likelihood evaluations necessary to perform T iterations of each algorithm on a data set of size N.

	Number of standardised likelihood evaluations
ApMHT	$random$
Pseudo-Marginal	$T(4K + 8m)/N$
SGLD	$2Tm/N$[1] or Tm/N[2]
Consensus MC	T/M
NPDPE	T/M

Table 2.1: Algorithms' number of standardised likelihood evaluations for a data set of size N and a number of iteration T

These values can easily be derived when looking at the pseudo-code of each algorithms. Note that the numerical evaluation of the gradient function of the likelihood requires two likelihood evaluations.

In practice, it is easier to derive analytically the gradient of the likelihood function with respect to z that to θ since the normalising constant in θ is often intractable.

Therefore, in order to mimic real scientific scenarios, we compute analytically the gradient function and the Hessian matrix of the likelihood with respect to z for the Pseudo-Marginal

[1] In the case of a numerical approximation of the gradient

[2] In the case of an analytic evaluation of the gradient

algorithm, as required.

However, for SGLD, we compute it numerically, as the gradient of the likelihood must be computed with respect to θ. Besides, we were not able to formally compute the gradient function at θ analytically for the models introduced in Sections 2.3 and 2.4 as the likelihood functions for these Data Generating Processes (DGP) involve the absolute value of θ.

For algorithm ApMHT, it is easy to see that the number of required likelihood evaluations is random. In order to compute this value, we use the variable *number_llik_eval* (cf algorithm 2) which counts the number of data points necessary to pass the test "$\delta < \varepsilon$". We then multibply this variable by 2 to account for the fact that the likelihood is evaluated at both the candidate θ' and the current state θ_t in this algorithm. Note that the algorithm in the form we gave it requires more likelihood evaluations - this is for simplicity of presentation. It is straightforward to store and re-use single-point likelihood evaluations on earlier batches. Hence it is more fair to use variable *number_llik_eval* instead of the actual number of likelihood evaluations performed by our personal implementation of the algorithm.

2.1.3 Convergence of the algorithms

Our last concern will be to determine whether the output of our algorithms is distributed according to the posterior distribution. To do so, we will check posterior consistency by applying the idea of Geweke (2004) which we describe below. This idea was then developed and extended to an actual test by Talts et al. (2018).

Theorem 2.1.1. *Let $\pi(\theta)$, $p(z|\theta)$ and $\pi(\theta|z)$ denote the prior, the likelihood and the posterior distributions of θ respectively.*

Let

$$\phi \sim \pi(.)$$

$$z' \sim p(.|\phi) \text{ and}$$

$$\theta \sim \pi(.|z')$$

Then the marginal distribution $p(\theta)$ *of* θ *is equal to the prior distribution of* θ, $\pi(\theta)$.

Proof.

$$\begin{aligned} p(\theta) &= \int_{\Theta}\int_{\mathcal{Z}} \pi(\phi)p(z'|\phi)\pi(\theta|z')dz'd\phi \\ &= \pi(\theta)\int_{\Theta}\int_{\mathcal{Z}} \frac{\pi(\phi)p(z'|\phi)p(z'|\theta)}{p(z')}dz'd\phi \\ &= \pi(\theta) \end{aligned} \tag{2.1}$$

Hence, if we simulate a sample $\{\theta_i\}_{i=1}^{T}$ from the above procedure, we should expect it to be distributed according to the prior distribution $\pi(.)$, which can easily be checked with a non-parametric test like the Kolmogorov-Smirnov test (Kolmogorov (1933)).

Note that in addition to $\pi(\theta|z)$, it is easy to see that $\pi(\theta)$ is also a solution to the integral equation

$$\int_{\Theta}\int_{\mathcal{Z}} \pi(\phi)p(z'|\phi)f(\theta)dz'd\phi = \pi(\theta) \tag{2.2}$$

which is to be solved for $f(\theta)$.

However, we were not able to prove formally that those were the only two solutions of (2.2). Yet, if we assume that they are, then the only way Geweke (2004)'s test could go wrong is if an algorithm happened to produce samples distributed according to the prior, which seems very unlikely.

Our comparison methodology is given below. The procedure implemented in order to choose the algorithms' parameter values was designed to mimic as best as possible how

these parameters might be tuned in real applications.

For a fixed number of iterations T,

1. When possible, we tune the algorithm parameters in order to achieve a reasonable Hellinger distance to the true posterior, using the first column of table 2.2. Note that some algorithms do not produce a good approximation when confronted to particular data sets no matter what parameter value is chosen.

2. For the algorithms whose outputs do yield a good enough approximation, we tune the parameters of the second column of table 2.2 in order to maximise the ESS per standardised likelihood evaluation, trying not to increase the approximation error significantly.

3. We then perform Geweke (2004)'s test in order to check if posterior consistency holds for a fixed number T of generated samples.

4. Finally, we report for each algorithm the ESS per standardised likelihood evaluation and the Hellinger distance to the true posterior.

	Parameter responsible mainly for...	
	Approximation error	Effective Sample size
ApMHT	ε	RW jump size
Pseudo-Marginal	K, m	RW jump size
SGLD	ε, m	ε
Consensus MC	-	RW jump size
NPDPE	h	RW jump size

Table 2.2: Summary of the role of each tuning parameter

Our choice of random walk (RW) proposals may be criticised as just comparing the algorithms efficiency for one proposal scheme. This is true. However, it is a very common choice and its sumplicity allows us to focus on the algorithms rather than the details of the updating process.

2.2 Test data set - Normal posterior

We start our analysis by what could be called a "trivial" model in order to check that all our algorithms were correctly implemented and that they produce a good approximation of the posterior distribution in the most simple cases.

2.2.1 Data Generating Process

Here we consider a model in which the data follows a normal distribution whose mean parameter θ is to be inferred. The model considered is constructed as follows.

$$p(z_i|\theta) = \mathcal{N}(z_i; \theta, \sigma^2), \text{ for } i = 1, ..., N \qquad \text{Likelihood} \qquad (2.3)$$

$$\pi(\theta) = \mathcal{N}(\theta; 1, s^2) \qquad \text{Prior distribution} \qquad (2.4)$$

Where σ^2=1 and $s^2 = 1$. We choose 1 to be the true value of θ

This model trivially leads to a normal posterior as the normal distribution is its own conjugate. For this model we set the number of iteration T to 20000 and the data set size N to 10000.

2.2.2 Results

The format of Figure 2.2 (which is repeated in the experiments below) is as follows. In each row of three graphs, the upper-left plot represents the true target in red and the approximation in blue. The MCMC trace of θ is given below. The right-hand-side plot shows a simulation from the prior, and the marginal of θ as obtained from the Geweke (2004)'s test described earlier. Recall that those two curves should be similar in case of success. Figure 2.2 does not reveal any issue. All five algorithms have produced a good approximation of the true posterior and have successfully passed Geweke (2004)'s test, which is what

Table 2.3: Summary

	Parameter tuning	Geweke (2004)'s test : Fail/Pass	Hellinger distance from *true posterior*	$ESS/$ std. likelihood evaluation
MH	-	-	0.00073	$\frac{2982.69}{20000} = 0.15$
ApMHT	$\varepsilon = 0.05$	Pass	0.0020	$\frac{3447.73}{22791.6} = 0.151$
Pseudo-marginal	m=300 K=53	Pass	0.0019	$\frac{3425.30}{5224} = 0.66$
SGLD	$\varepsilon = 0.0002$ m=5000	Pass	0.00062	$\frac{19584.3}{20000} = 0.98$
Consensus MC	M=10	Pass	0.0011	$\frac{3308.06}{2000} = 1.65$
NPDPE	M=10 h=0.01	Pass	0.0055	$\frac{125.97}{2000} = 0.063$

we expected. Plot (i) shows a relatively poorer mixing of NPDPE which is reflected by a lower ESS in Table 2.3. Overall this table and Figure 2.1 reveal good performances with a clear dominance of Consensus Monte Carlo in terms ESS per standardised likelihood evaluations, followed closely by SGLD. In terms of quality of approximation, SGLD seems to perform even better than MH itself.

When compared to its counterparts, NPDPE shows rather disappointing results. In particular, it is the only algorithm which does not beat MH in terms of ESS per std. likelihood evaluation.

Figure 2.1: Log of Hellinger distance vs log of ESS per std. likelihood evaluation

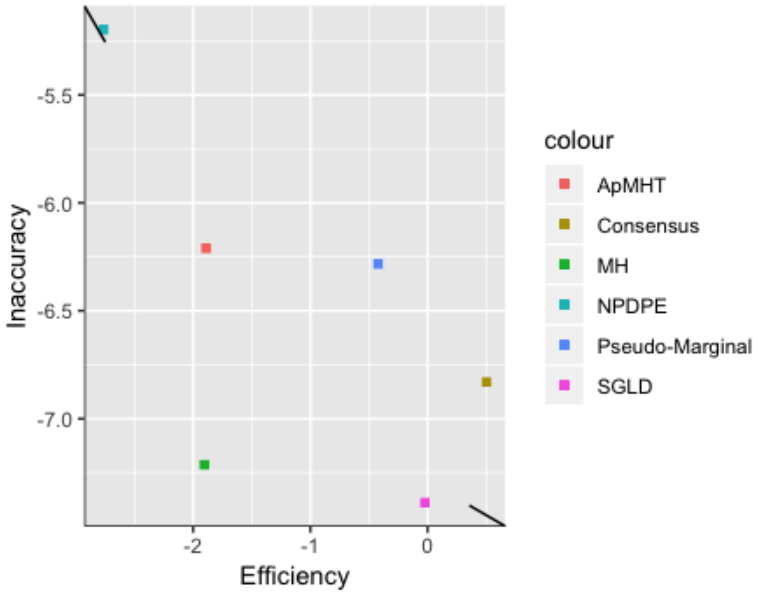
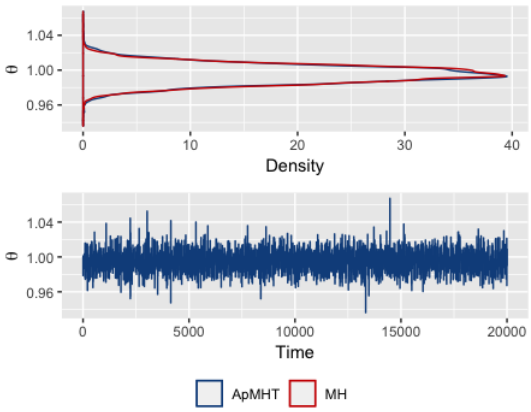
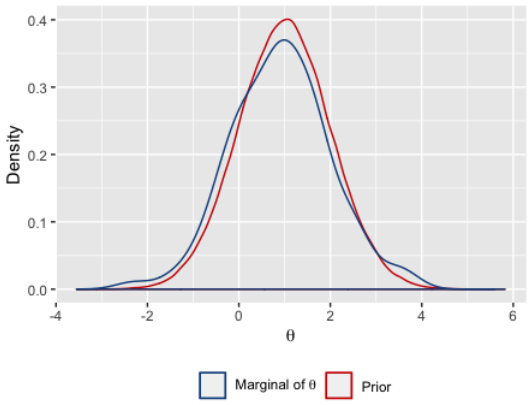


Figure 2.2: Summary plots

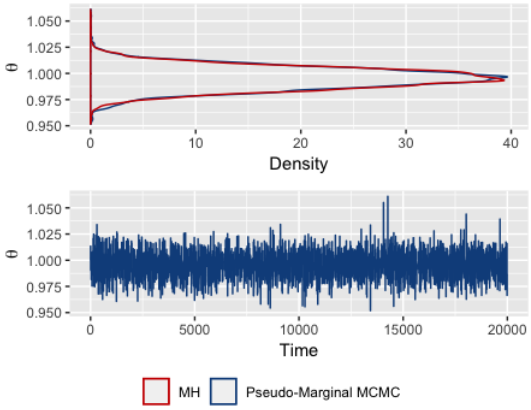
(a) ApMHT diagnostics



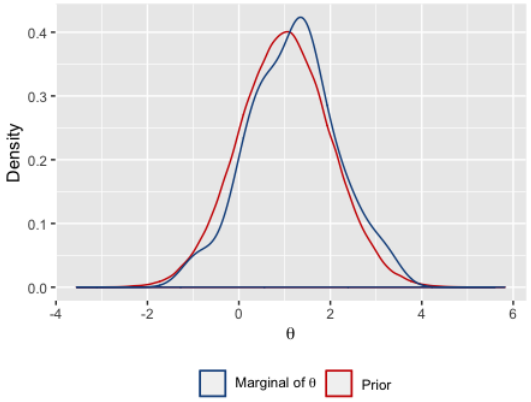
(b) ApMHT Geweke (2004)'s test, p-value=0.63



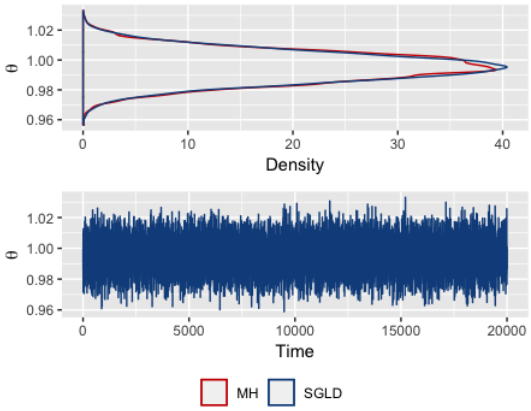
(c) Pseudo-Marginal diagnostics



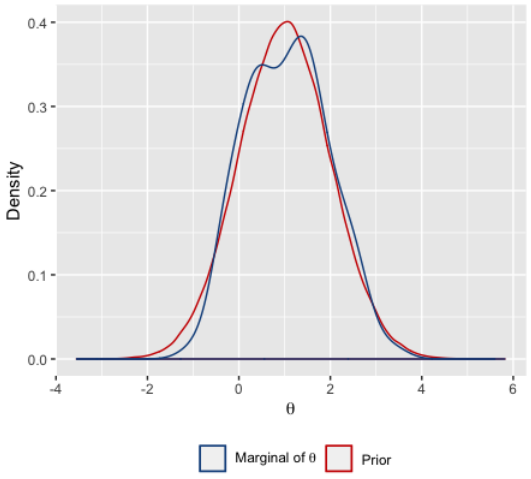
(d) PM Geweke (2004)'s test, p-value=0.13



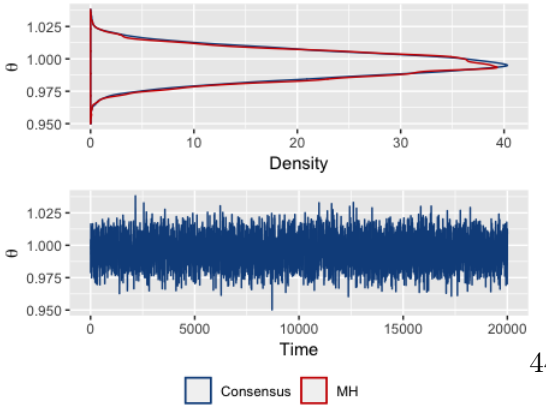
(e) SGLD diagnostics



(f) SGLD Geweke (2004)'s test, p-value=0.77



(g) Consensus diagnostics



(h) Consensus Geweke (2004)'s test, p-value=0.20

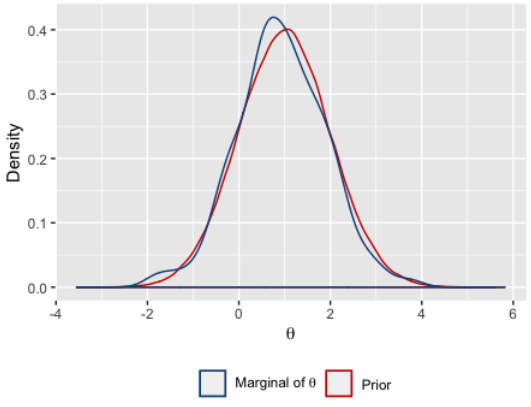
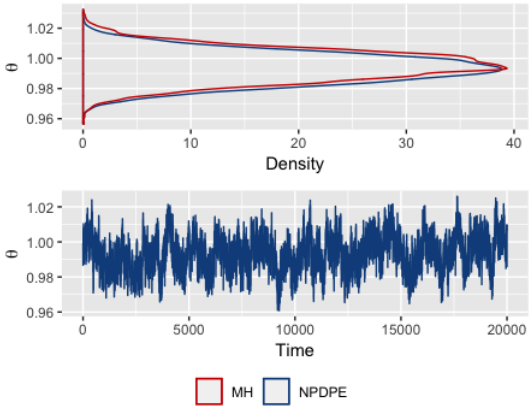
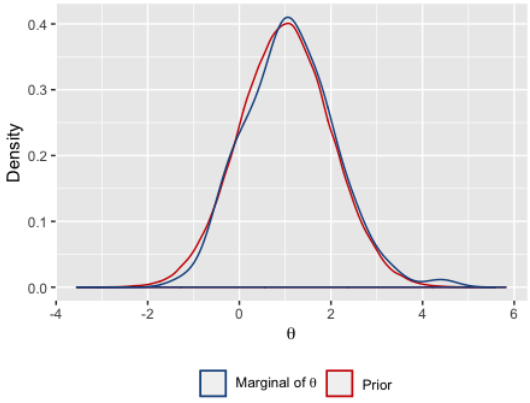


Figure 2.2: continued

(i) NPDPE diagnostics



(j) NPDPE Geweke (2004)'s test, p-value=0.72



2.3 Bimodal - High variance

In this section, we wish to design a model whose posterior distribution is bimodal such that the probability density between the two modes is not too low. Since normality does not hold in this case, we can test the adaptability of Consensus Monte-Carlo which relies heavily on this assumption. It will also be interesting to see whether gradient methods like SGLD are able to mix properly or if, on the contrary, they will get stuck in one of the modes. More generally, this model should be a challenge to the mixing abilities of our algorithms.

2.3.1 Data Generating Process

The model considered is constructed as follows

$$p(z_i|\theta) = \mathcal{N}(z_i; |\theta|, \sigma^2), \text{ for } i = 1, ..., N \qquad \text{Likelihood} \qquad (2.5)$$

$$\pi(\theta) = \mathcal{N}(\theta; 0, s^2) \qquad \text{Prior distribution} \qquad (2.6)$$

Where σ^2=20 and $s^2 = 0.3^2$.

Hence, the posterior given one observation z_i is of the form

$$\begin{aligned}\pi(\theta|z_i) &\propto \exp\left\{-\frac{1}{\sigma^2}(z_i - |\theta|)^2\right\} \times \exp\left\{-\frac{|\theta|^2}{2s^2}\right\} \\ &\propto \exp\left\{-\frac{(1+\frac{\sigma^2}{s^2})}{2}(|\theta| - (1+\frac{\sigma^2}{s^2})^{-1}z_i)^2\right\}\end{aligned}$$

which is bimodal in θ with the two modes being symmetric around 0. We choose $\theta = 0.1$ so that the area of low posterior probability between the two modes is not too large to prevent good mixing. For this model we set $T = 20000$ and the data set size $N = 10000$.

2.3.2 Results

We attempt to tune each algorithm's parameters so that their Hellinger distance from the true posterior is below 0.01. As can be seen in Table 2.4 we did not manage to tune SGLD parameter ε in order to achieve this goal but the Hellinger distance for this algorithm can be considered close enough. On the contrary, some algorithms like Pseudo-marginal MCMC are quite far below the chosen threshold.

As can be seen on Figure 2.4, all algorithms have successfully passed Geweke (2004)'s test except Consensus Monte-Carlo. This could be expected as the technique derived by Scott et al. (2016) relies heavily on the assumption that normality holds, which is clearly not the case here. Conversely, its parallel counterpart NPDPE seems to have worked in a better way probably thanks to its non-parametric property. Surprisingly, SGLD does seem to have made a good approximation of the true posterior, although we expected it to get stuck in one of the modes.

The trace plots indicate rather good mixing even though NPDPE's performance in that matter seems weaker.

When we look at the number of standardised likelihood evaluations we see that Pseudo-

Marginal MCMC is undeniably more efficient. When it comes to ESS, we can spot no real outlier although we should highlight the rather poor performance of NPDPE, which we could already infer from the trace plot.

Overall, when looking at the ESS per standardised likelihood evaluation, it is clear that Pseudo-Marginal MCMC outperformed the other algorithms on this synthetic data set.

Table 2.4: Summary

	Parameter tuning	Geweke (2004)'s test : Fail/Pass	Hellinger distance from *true posterior*	ESS/ std. likelihood evaluation
MH	-	-	0.0012	$\frac{2627.75}{20000} = 0.13$
ApMHT	$\varepsilon = 0.05$	Pass	0.0018	$\frac{2589.55}{19130.2} = 0.14$
Pseudo-marginal	m=100 K=190	Pass	0.00083	$\frac{1614.1}{3120} = 0.52$
SGLD	$\varepsilon = 0.05$ m=5000	Pass	0.012	$\frac{1361.1}{20000} = 0.07$
Consensus MC	M=10	Fail	0.086*	$\frac{2418.326}{2000} = 1.21$*
NPDPE	M=10 h=0.1	Pass	0.0012	$\frac{306.97}{2000} = 0.15$

*These values are not relevant as the algorithm did not pass Geweke (2004)'s test.

Figure 2.3: Log of Hellinger distance vs log of ESS per std. likelihood evaluation

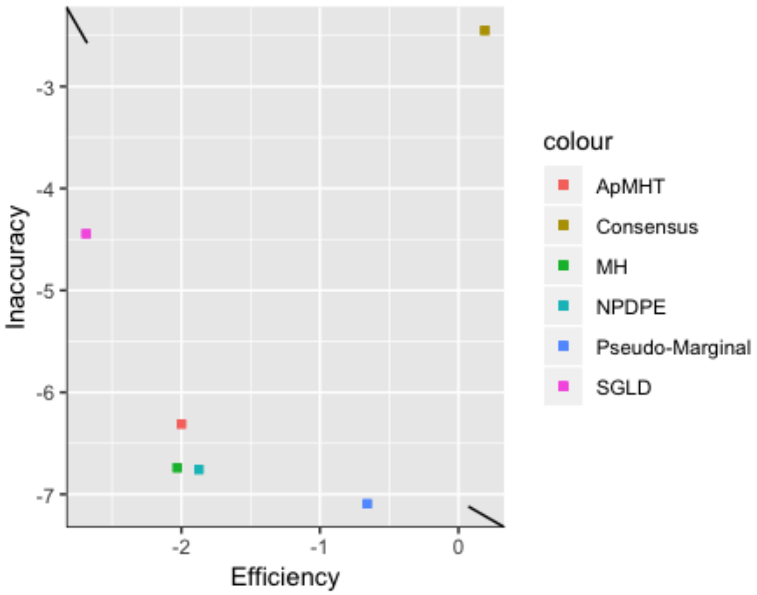
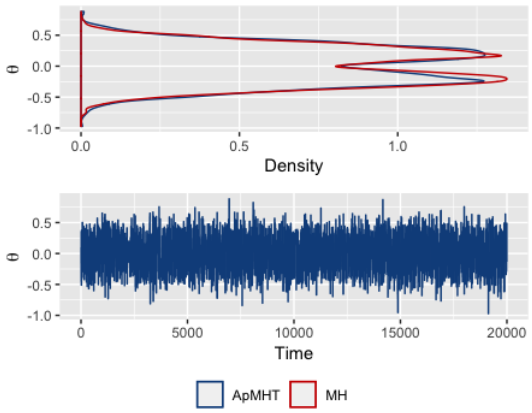
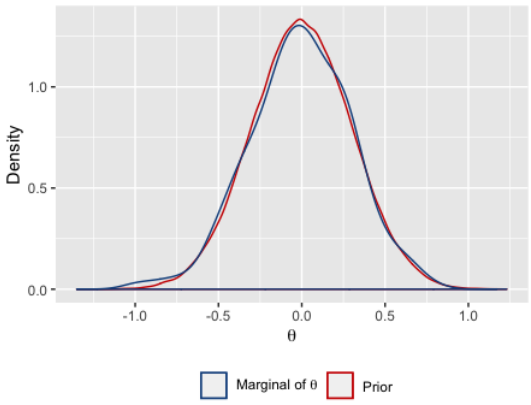


Figure 2.4: Summary plots

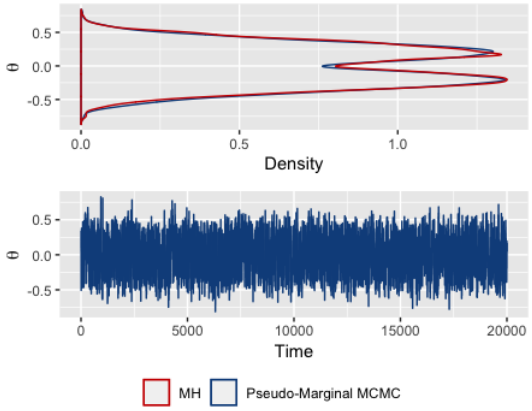
(a) ApMHT diagnostics



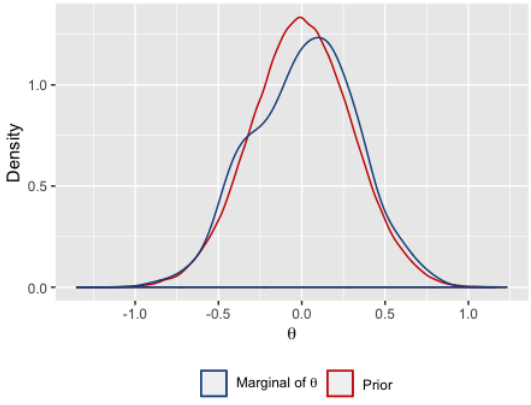
(b) ApMHT Geweke (2004)'s test, p-value=0.97



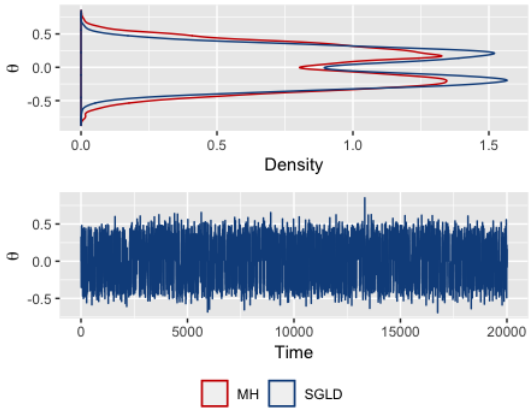
(c) Pseudo-Marginal diagnostics



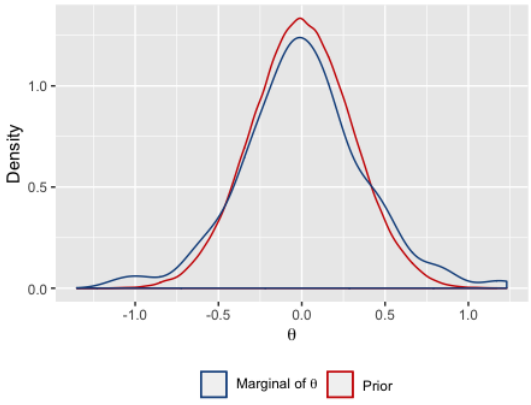
(d) PM Geweke (2004)'s test, p-value=0.17



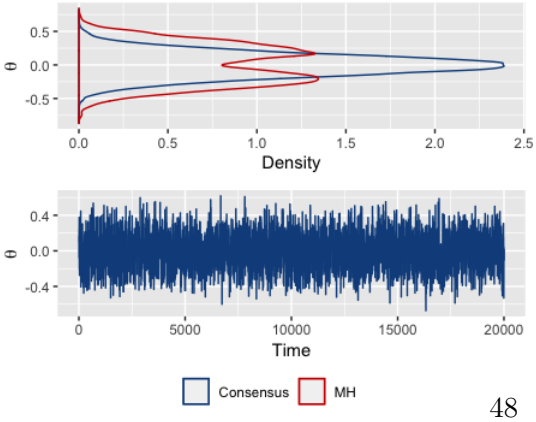
(e) SGLD diagnostics



(f) SGLD Geweke (2004)'s test, p-value=0.89



(g) Consensus diagnostics



(h) Consensus Geweke (2004)'s test, p-value=0

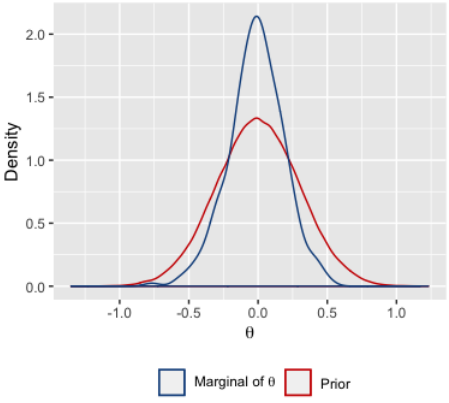
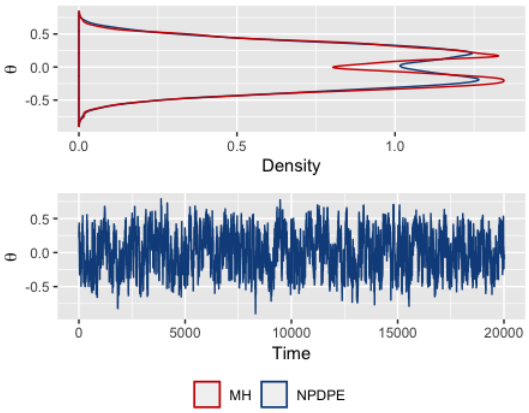
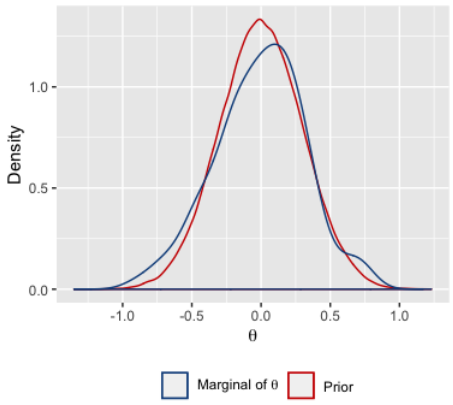


Figure 2.4: continued

(i) NPDPE diagnostics



(j) NPDPE Geweke (2004)'s test, p-value=0.92



2.4 Bimodal - Low variance

In this section, we repeat a similar experiment of bimodality. However, we truly separate the modes by an area of low probability. This should pose a true challenge for MCMC algorithms that do not propose new candidates according to a random walk as they may not be able to cross this artificial low porobability "bridge". In particular, it will be interesting to see how SGLD, which performed surprinsingly well in the previous experiment, cope with this situation.

2.4.1 Data Generating Process

We repeat the scheme of the previous section setting this time $\sigma^2 = 1$. Thanks to this trick, the posterior distribution should be much more concentrated around the modes.

$$p(z_i|\theta) = \mathcal{N}(z_i; |\theta|, \sigma^2), \text{ for } i = 1, ..., N \qquad \text{Likelihood} \qquad (2.7)$$

$$\pi(\theta) = \mathcal{N}(\theta; 0, s^2) \qquad \text{Prior distribution} \qquad (2.8)$$

Table 2.5: Summary

	Parameter tuning	Geweke (2004)'s test : Fail/Pass	Hellinger distance from *true posterior*	$ESS/$ std. likelihood evaluation
MH	-	-	0.013	$\frac{20.72}{20000} = 0.001$
ApMHT	$\varepsilon = 0.05$	Pass	0.016	$\frac{125.56}{26537} = 0.005$
Pseudo-marginal	m=200 K=54	Pass	0.003	$\frac{168.08}{3632} = 0.046$
SGLD	$\varepsilon = 2.25 * 10^{-4}$ m=5000	Fail	0.31	$\frac{25588.15}{20000} = 1.28$*
Consensus MC	M=3	Fail	0.83	$\frac{2031.54}{6666} = 0.30$*
NPDPE	M=3 h=0.07	Fail	0.25	$\frac{1609.33}{6666} = 0.24$*

*These values are not relevant as the algorithm did not pass Geweke (2004)'s test.

Where σ^2=1 and $s^2 = 0.3^2$. For this model we set $T = 20000$ and the data set size $N = 10000$

2.4.2 Results

Figure 2.6 indicates that ApMHT and Pseudo-marginal MCMC seem to give the best approximations of the true posterior. They are also the only two methods that successfully pass Geweke (2004)'s test.

As to the other algorithms, we can see that, as feared, SGLD got stuck in one mode whereas consensus Monte-Carlo unsurprisingly produced an erroneous approximation based on the Bernstein-von Mises theorem. However, we can see that NPDPE almost succeeded in approximating the two mode-scheme but clearly overestimated the variance of the posterior. When ignoring the algorithms which did not converge to the posterior, Table 2.5 and Figure 2.5 show a clear dominance of Pseudo-Marginal MCMC both in terms of quality of approximation and efficiency, followed by ApMHT. Both algorithms yield an equal or better Hellinger distance to the posterior than MH and a substantially higher ESS per std. likelihood evaluation.

Figure 2.5: Log of Hellinger distance vs log of ESS per std. likelihood evaluation

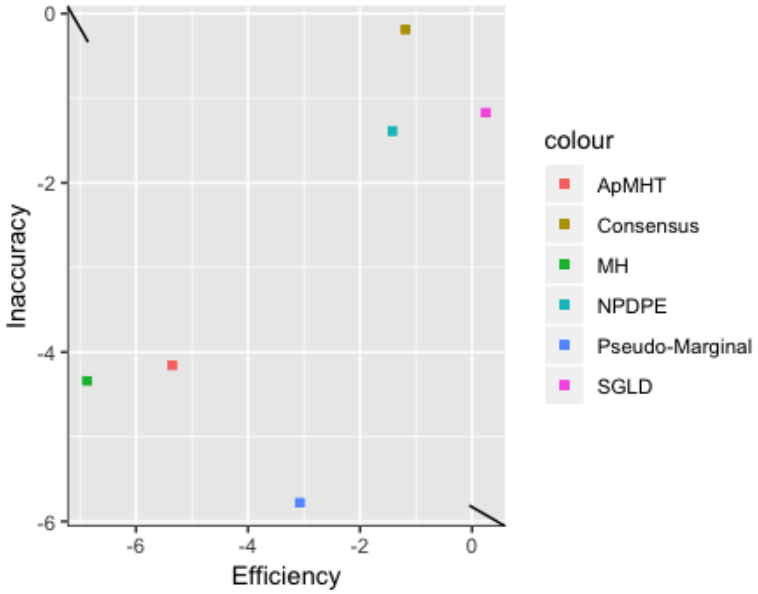
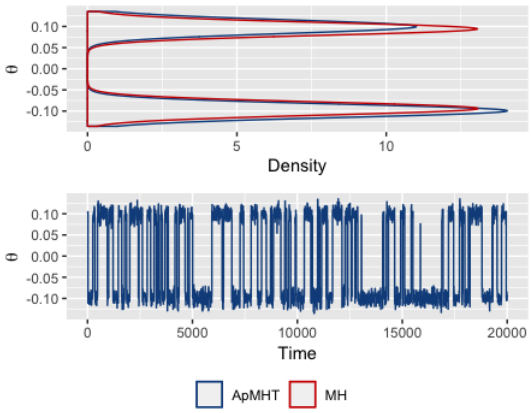
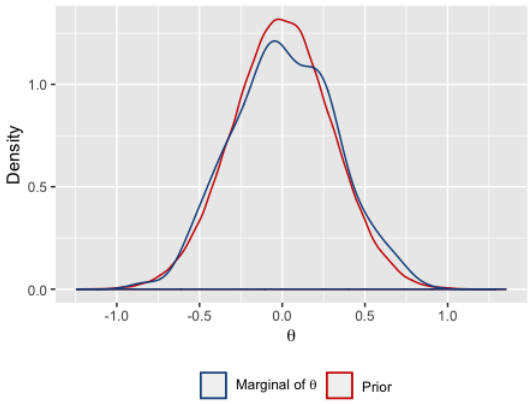


Figure 2.6: Summary plots

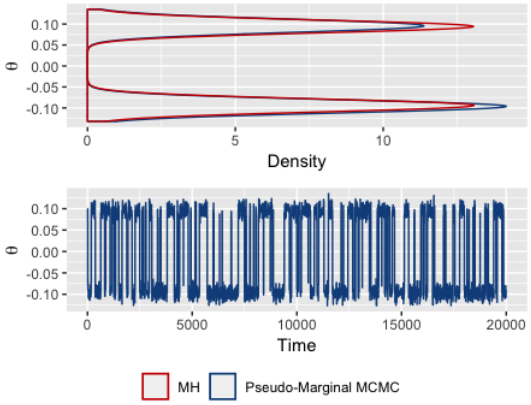
(a) ApMHT diagnostics



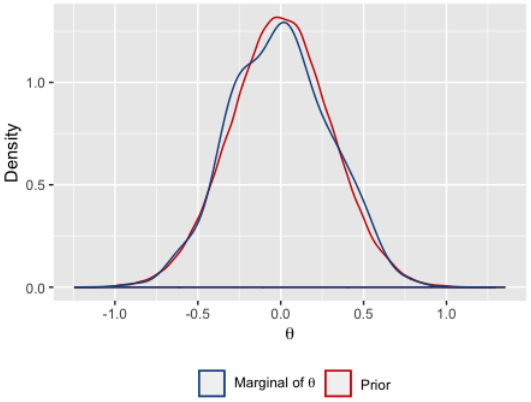
(b) ApMHT Geweke (2004)'s test, p-value=0.37



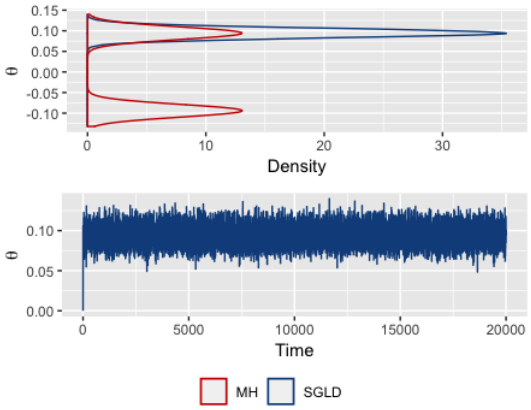
(c) Pseudo-Marginal diagnostics



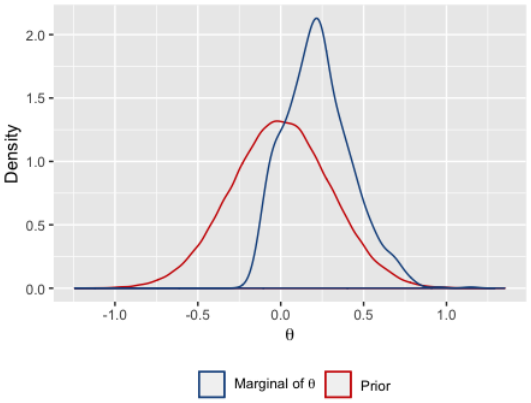
(d) PM Geweke (2004)'s test, p-value=0.88



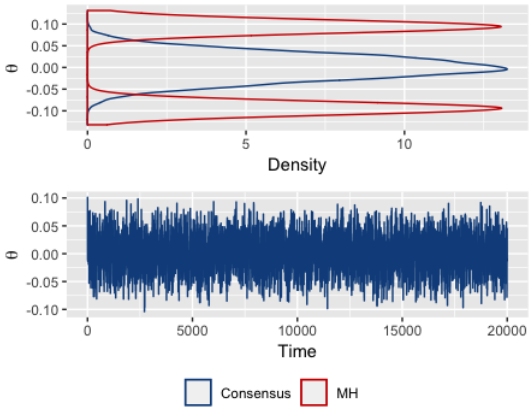
(e) SGLD diagnostics



(f) SGLD Geweke (2004)'s test, p-value=0



(g) Consensus diagnostics



(h) Consensus Geweke (2004)'s test, p-value=0

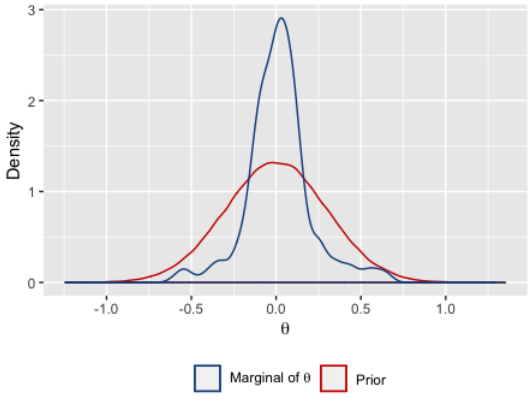
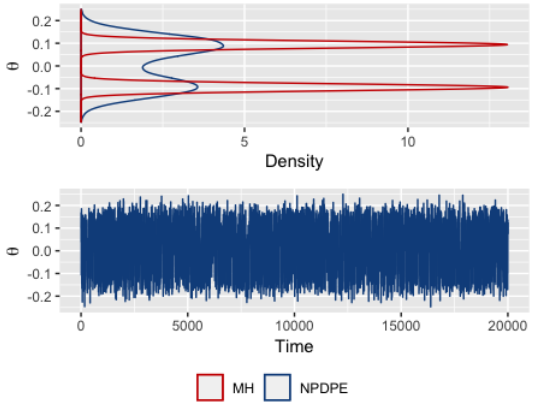
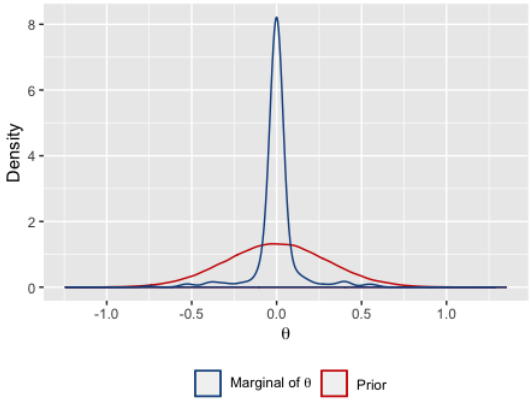


Figure 2.6: continued

(i) NPDPE diagnostics



(j) NPDPE Geweke (2004)'s test, p-value=0



2.5 High-dimensional logistic regression

This final experiment will give us a more concrete insight into real world problems as we will now consider a fairly high dimensional logistic regression with a relatively low amount of data. It might seem surprising to set $N = 1000$ when our main interest are tall data sets but we wished to reflect the common situation when the number of parameters to estimate, in our case 20, combined with a 'small' amount of data makes accurate inference difficult to achieve. However, the conclusions we will draw from this experiment can easily be extended to situations where say $N = 100000$ and the number of parameters is $d = 1000$. This setting also enables us to confront our algorithms with rather fat-tailed posterior distribution, which has not been done previously. In such situations we expect mixing to be difficult for the algorithms which propose updates according to a random walk, even with a large step-size.

2.5.1 Data Generating Process

In this section, we artificially create a logistic regression setting. To do so, we first sample our true parameters as follows,

$$\theta \sim \mathcal{N}_d(0, 1)$$

where $d = 20$.

We then generate the covariates $x_i = (x_{1i}, ..., x_{20i})$ for $i = 1, ..., 1000$ each independently from a standard Normal distribution.

Finally,

$$z_i \sim \mathcal{B}(\pi_i)$$

where

$$\pi_i = \frac{e^{\theta_1 x_{1i} + ... + \theta_{20} x_{20i}}}{1 + e^{\theta_1 x_{1i} + ... + \theta_2 0 x_{20i}}}$$

and $\mathcal{B}(\pi_i)$ denotes the Bernoulli distribution of mean π_i.

For this experiment, we choose to set $T = 20000$ and simulate our *true posterior* from 2,000,000 runs of standard random-walk MH. Moreover, we choose the prior distribution π to be standard multivariate normal.

2.5.2 Results

So as to provide clear and meaningful plots, Figure 2.8 shows contour lines of the marginal distributions of θ_2 and θ_4 only, and all other measures of assessment including Geweke (2004)'s test are carried out on the marginal distribution of θ_2. As can be seen on the summary plots (Figure 2.8) most algorithms seem to have produced a good enough approximation of the true posterior, even though the marginals sampled by NPDPE look somewhat shifted. Nevertheless, all Geweke (2004)'s tests suggest that samples were indeed drawn from the posterior distribution. Overall, Table 2.6 and Figure 2.7 show very strong performances in term of approximation and efficiency for SGLD when compared to its counterparts. The facts that the mixing of all other algorithms is poorer tends to indicate that another proposal distribution should be used instead of a random walk. In particular, a Langevin proposal might be more efficient, as suggested by the achievements of SGLD. Therefore, we cannot discard these algorithms as their performances might be enhanced by a more suitable choice of proposal distribution.

Table 2.6: Summary

	Parameter tuning	Geweke (2004)'s test : Fail/Pass	Hellinger distance from *true posterior*	$ESS/$ std. likelihood evaluation
MH	-	-	0.0024	$\frac{273.50}{20000} = 0.014$
ApMHT	$\varepsilon = 0.05$	Pass	0.0054	$\frac{273.70}{25840.8} = 0.011$
Pseudo-marginal	m=100 K=153	Pass	0.0027	$\frac{185.10}{28240} = 0.0066$
SGLD	$\varepsilon = 0.022$ m=500	Pass	0.00042	$\frac{20685.87}{20000} = 1.03$
Consensus MC	M=3	Pass	0.015	$\frac{375.23}{6660} = 0.056$
NPDPE	M=3 h=0.07	Pass	0.025	$\frac{510.05}{6660} = 0.077$

Figure 2.7: Log of Hellinger distance vs log of ESS per std. likelihood evaluation

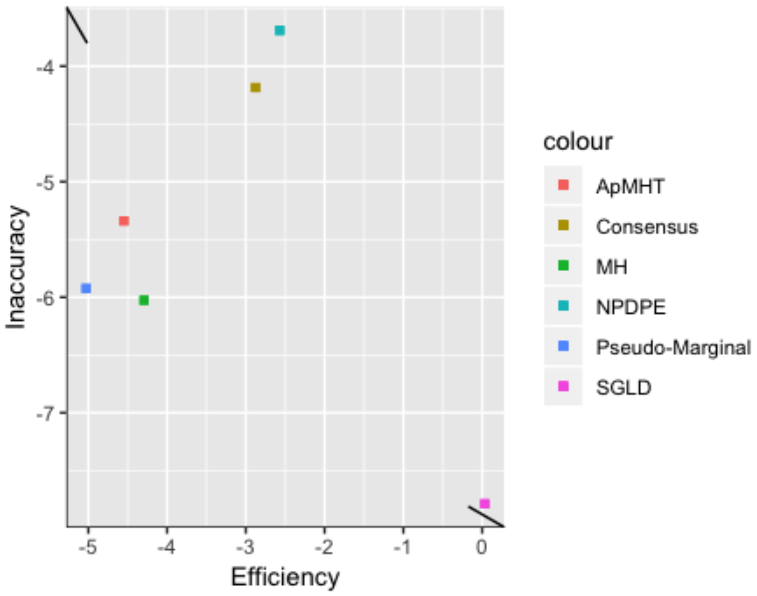
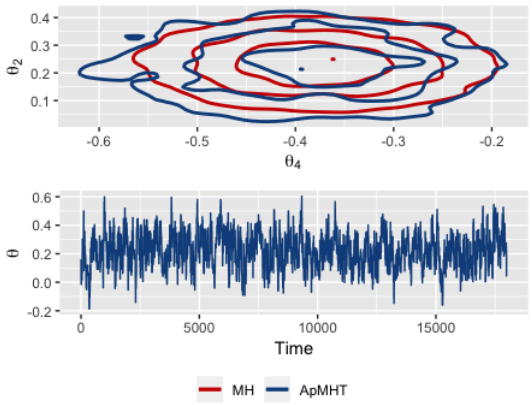
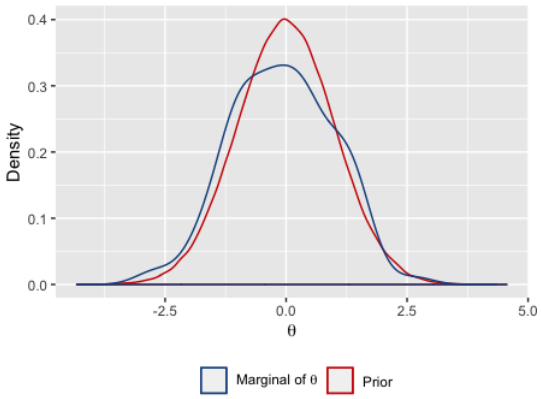


Figure 2.8: Summary plots

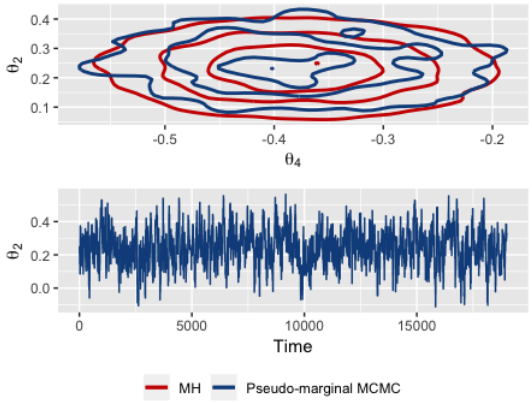
(a) ApMHT diagnostics



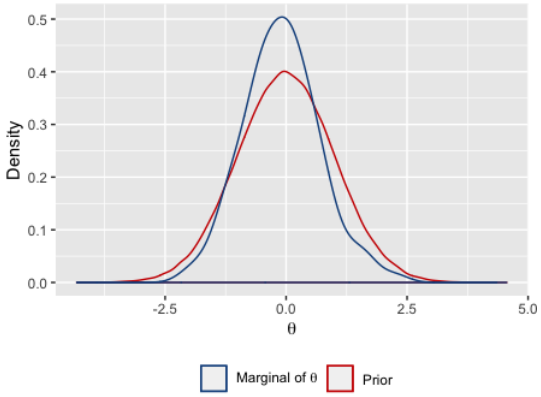
(b) ApMHT Geweke (2004)'s test, p-value=0.50



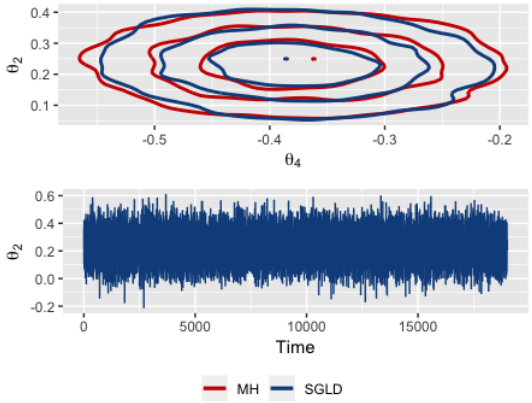
(c) Pseudo-Marginal diagnostics



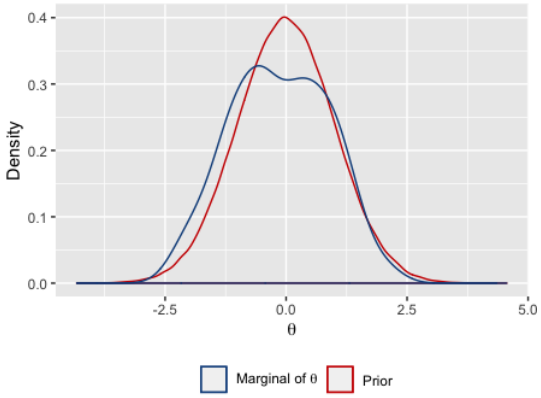
(d) PM Geweke (2004)'s test, p-value=0.23



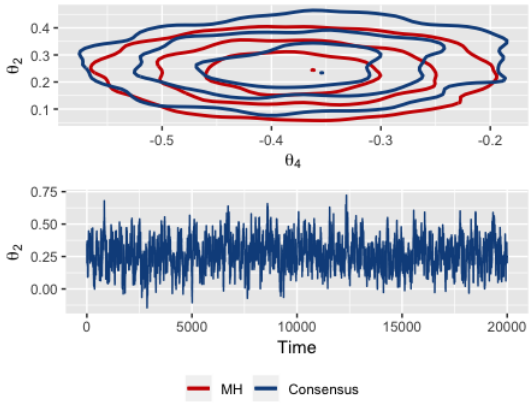
(e) SGLD diagnostics



(f) SGLD Geweke (2004)'s test, p-value=0.26



(g) Consensus diagnostics



(h) Consensus Geweke (2004)'s test, p-value=0.72

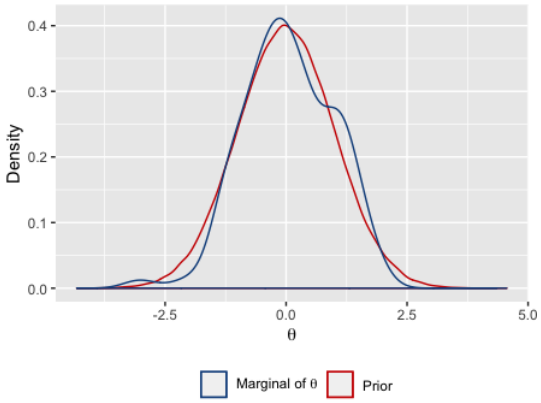
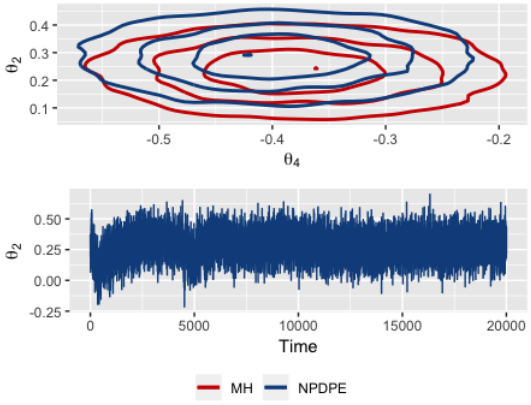
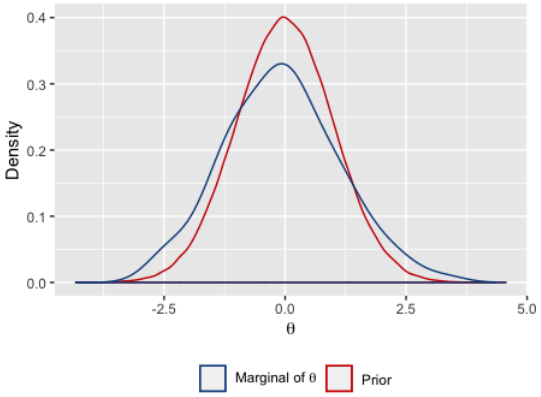


Figure 2.8: continued

(i) NPDPE diagnostics



(j) NPDPE Geweke (2004)'s test, p-value=0.17



Conclusions and future work

Conclusions

During this study, we have been able to implement several recent algorithms designed to improve traditional MCMC methods when confronted to tall data sets in a sample Bayesian setting. The different experiments we carried out led us to the following conclusions.

Consensus Monte Carlo is a very easy algorithm to implement which performs surprisingly well, providing some conditions are met. It indeed turned out to be very dependent on the assumption that the Bernstein-Von Mises theorem holds. When this condition is satisfied, it generally outperforms most of its counterparts in terms of ESS per std likelihood evaluation, even though its approximation error is usually slightly higher. On the other hand, more complex frameworks such as bimodality really hindered a correct posterior sampling. Therefore, we highly recommend its use for simple models especially if the user has access to a high number of computational units. In our examples, we set $M = 3$ as typical laptops will have either 2 or 4 cores (assuming they cannot run more than 1 thread at a time). However, one can easily imagine implementing the algorithm with large values of M, which would improve efficiency even more.

We picked our second *divide and conquer* algorithm, NPDPE, as a more flexible alternative to Consensus Monte Carlo. Despite its tedious implementation, this method did

prove itself able to adapt to more scenarios than its other parallel counterpart and only failed the Geweke (2004)'s test once in the context of an extreme experimental design. Although it has suffered from poor mixing and relatively high approximation errors, the last experiment showed its ability to perform well where the other methods experienced difficulties. As for consensus Monte Carlo, its number of required likelihood evaluations makes it a mighty tool when confronted to large data sets.

SGLD, a rejection-free algorithm, was our only algorithm whose proposal distribution moved along the gradient of the posterior distribution. Thus, we had concerns as to its ability to mix properly when confronted to multimodality or posteriors with a complex geometry. However, this simple algorithm performed remarkably well on our first bimodal example as it succeeded not to get trapped in one of the modes, although its efficiency was not among the best. Moreover, it outperformed by far the other methods on the logistic regression experiment, both in terms of approximation error (beating Metropolis-Hastings itself) and of ESS per std likelihood evaluations. Its performances on our test data set were also remarkable.

Pseudo-marginal MCMC is probably the most consistent and reliable algorithm we have implemented. It passed the Geweke (2004)'s test for all experiments and yielded the best approximation error and ESS per std likelihood evaluation for both our bimodal experimental settings. Its poorer mixing on the last experiment would probably be improved with a more suitable choice of proposal distribution. Its downside however is that it requires to derive the gradient and Hessian matrix of the likelihood function evaluated at the data analytically. If those quantities had to be computed numerically, one should expect a high increase in the number of required likelihood evaluations.

Finally, ApMHT, which is a rather intuitive algorithm, showed it had a great adapt-

ability to most situations. Like for Pseudo-marginal MCMC, no test suggested that it did not sample from the posterior distribution. Although it has never been the top-performing algorithm, it showed great consistency across the different synthetic models we designed. However, MH sometimes proved itself more efficient than ApMHT.

We sum up all our conclusions in Table 2.7 where we mention the principal features of each algorithm and also provide a personal feeling regarding the level of difficulty of implementation for each method.

Future work

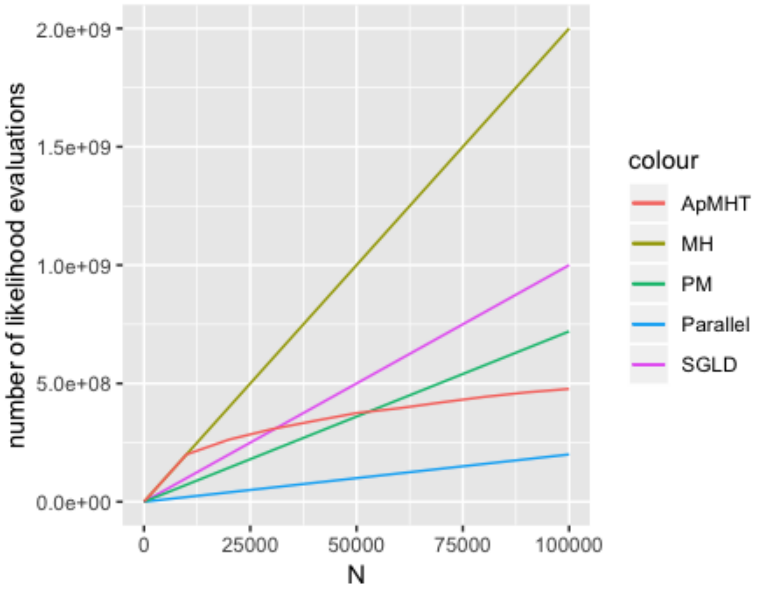
Throughout our analysis, we tried to give a useful insight into how modern MCMC techniques could tackle the challenges Bayesian inference has been facing since the advent of Big Data.

Some additional content might have been interesting to provide. In particular, it would be interesting to know whether the number of required likelihood evaluations is a linear or sub-linear function of the data set size N. To give a simple overview of the situation, we provide Figure 2.9. These curves were derived using our first model (test data set) by using the number of likelihood evaluations indicated in Table 2.7 below. For Consensus Monte Carlo and NPDPE we assumed the user had M=10 machines available. By looking at our previous experiments' tuning parameters values we make the assumptions that $m = 0.5N$ for SGLD and $K = 0.05N$, $m = 0.02N$ for Pseudo-Marginal MCMC. For ApMHT we had to run simulations for different values of N as the number of likelihood evaluations is random.

Making these strong assumptions, we can see that only ApMHT's curve seems to be a sub-linear function of N. However the curves of SGLD and Pseudo-Marginal represent here a worst-case scenario as the values m and K should be tuned for each data set size N. Hence, some further work should be done in order to produce similar plots on different

data sets, making sure that the parameters of each algorithm are properly tuned for each value of N.

Figure 2.9: Log of Hellinger distance vs log of ESS per std. likelihood evaluation



One should also consider using different ways of proposing a candidate θ' for all the algorithms except SGLD. We have indeed seen in our last experiment that the choice of proposal distribution is essential for good mixing and satisfactory performances.

A lot of additional work out of the scope of this study could be carried out in order produce a complete overview of the current state of research in this domain. For instance, other recent approaches like the rejection-free Bouncy Particle Sampler by Bouchard-Côté et al. (2017) samples exactly from the posterior distribution and shows really promising results.

One should also consider comparing the MCMC methods we implemented to other popular tools designed to perform approximate Bayesian inference like Variational inference (e.g Jain et al. (2018)) or Expectation propagation (see Minka (2001)).

Table 2.7: Summary table

	ApMHT (RW)	Pseudo-Marginal (RW)	SGLD	Consensus (RW)	NPDPE (RW)
Exact posterior sampling ?	Approximate Asym. exact as $\varepsilon \to 0$	Approximate Asym. exact as $m \to N$	Approximate Asym. exact as $m \to N$	Asym. exact if B-vM theorem holds	Approximate
Number of likelihood evaluations	random	$T(4K + 8m)$*	Tm*	TN/M	TN/M
Difficulty of implementation	Easy	Difficult*	Medium*	Easy	Medium
Normal model	AE** = 4^{th} ESS/llik*** = 4^{th}	AE=3^{rd} ESS/llik=3^{rd}	AE =1^{st} ESS/llik =2^{nd}	AE =2^{nd} ESS/llik =1^{st}	AE =5^{th} ESS/llik =5^{th}
Bimodal model high variance	AE = 3^{rd} ESS/llik = 3^{rd}	AE=1^{st} ESS/llik=1^{st}	AE =4^{th} ESS/llik =4^{th}	Fail	AE =2^{nd} ESS/llik =2^{rd}
Bimodal model low variance	AE = 2^{nd} ESS/llik = 2^{nd}	AE=1^{st} ESS/llik=1^{st}	Fail	Fail	Fail
High dimensional logistic reg. model	AE = 3^{rd} ESS/llik = 4^{th}	AE=2^{nd} ESS/llik=5^{th}	AE =1^{st} ESS/llik =1^{st}	AE =4^{nd} ESS/llik =3^{rd}	AE =5^{th} ESS/llik =2^{nd}

*Assuming the gradient and the Hessian matrix are computed analytically

**Ranking of the algorithms according to their ability to minimize the approximation error (AE)

***Ranking of the algorithms according to their ability to maximise the ESS per std likelihood evaluation (ESS/llik)

References

Bardenet, R., Doucet, A., and Holmes, C. (2017). On markov chain monte carlo methods for tall data. *The Journal of Machine Learning Research*, 18(1):1515–1557.

Bouchard-Côté, A., Vollmer, S. J., and Doucet, A. (2017). The bouncy particle sampler: A non-reversible rejection-free markov chain monte carlo method. *Journal of the American Statistical Association*, (just-accepted).

Ceperley, D. and Dewing, M. (1999). The penalty method for random walks with uncertain energies. *The Journal of chemical physics*, 110(20):9812–9820.

Geweke, J. (2004). Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association*, 99(467):799–804.

Hellinger, E. (1909). Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271.

Jain, A., Srijith, P., and Desai, S. (2018). Variational inference as an alternative to mcmc for parameter estimation and model selection. *arXiv preprint arXiv:1803.06473*.

Kolmogorov, A. (1933). Sulla determinazione empirica di una lgge di distribuzione. *Inst. Ital. Attuari, Giorn.*, 4:83–91.

Korattikara, A., Chen, Y., and Welling, M. (2014). Austerity in mcmc land: Cutting the metropolis-hastings budget. In *International Conference on Machine Learning*, pages 181–189.

Le Cam, L. and Yang, G. L. (2012). *Asymptotics in statistics: some basic concepts*. Springer Science & Business Media.

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.

Minka, T. P. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc.

Neiswanger, W., Wang, C., and Xing, E. (2013). Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*.

Nicholls, G. K., Fox, C., and Watt, A. M. (2012). Coupled mcmc with a randomized acceptance probability. *arXiv preprint arXiv:1205.6857*.

Quiroz, M., Kohn, R., Villani, M., and Tran, M.-N. (2018). Speeding up mcmc by efficient data subsampling. *Journal of the American Statistical Association*, (just-accepted):1–35.

Roberts, G. O. and Rosenthal, J. S. (1998). Optimal scaling of discrete approximations to langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268.

Scott, S. L., Blocker, A. W., Bonassi, F. V., Chipman, H. A., George, E. I., and McCulloch, R. E. (2016). Bayes and big data: The consensus monte carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88.

Talts, S., Betancourt, M., Simpson, D., Vehtari, A., and Gelman, A. (2018). Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin

dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688.

Appendix A

Miscellaneous

A.1 Proof of theorem 1.1.1 by Korattikara et al. (2014)

Following is a screenshot of the proof provided by Korattikara et al. (2014) in their article. The Markov transition kernel they define seems to be wrong as it should be equal to

$$P_a(\theta, \theta')q(\theta'|\theta) + \left(1 - \int_{\theta'} P_a(\theta, \theta')q(\theta'|\theta)d\theta'\right)\delta_\theta(\theta')$$

following the notation of the authors.

This mistake changes the proof dramatically but the obtained result is the same, as we have shown.

C.2 Proof of Theorem 1

We first derive an upper bound for the one step error of the approximate Metropolis-Hastings algorithm, and then use Lemma 3 to prove Theorem 1. The transition kernel of the exact Metropolis-Hastings algorithm can be written as

$$\mathcal{T}_0(\theta,\theta') = P_a(\theta,\theta')q(\theta'|\theta) + (1 - P_a(\theta,\theta'))\delta_D(\theta' - \theta) \tag{33}$$

where δ_D is the Dirac delta function. For the approximate algorithm proposed in this paper, we use an approximate MH test with acceptance probability $\tilde{P}_{a,\epsilon}(\theta,\theta')$ where the error, $\Delta P_a \stackrel{\text{def}}{=} P_{a,\epsilon} - P_a$, is upper bounded as $|\Delta P_a| \leq \Delta_{\max}$. Now let us look at the distance between the distributions generated by one step of the exact kernel $\mathcal{T}_0$ and the approximate kernel $\mathcal{T}_\epsilon$. For any P,

$$\begin{aligned}
&\int_{\theta'} d\Omega(\theta')|(P\mathcal{T}_\epsilon)(\theta') - (P\mathcal{T}_0)(\theta')| \\
&= \int_{\theta'} d\Omega(\theta') \left| \int_\theta dP(\theta)\Delta P_a(\theta,\theta') \left(q(\theta'|\theta) - \delta_D(\theta' - \theta)\right) \right| \\
&\leq \Delta_{\max} \int_{\theta'} d\Omega(\theta') \left| \int_\theta dP(\theta)(q(\theta'|\theta) + \delta_D(\theta' - \theta)) \right| \\
&= \Delta_{\max} \int_{\theta'} d\Omega(\theta') \left(g_Q(\theta') + g_P(\theta')\right) = 2\Delta_{\max}
\end{aligned} \tag{34}$$

where $g_Q(\theta') \stackrel{\text{def}}{=} \int_\theta dP(\theta)q(\theta'|\theta)$ is the density that would be obtained by applying one step of Metropolis-Hastings without rejection. So we get an upper bound for the total variation distance as

$$d_v(P\mathcal{T}_\epsilon, P\mathcal{T}_0) = \frac{1}{2}\int_{\theta'} d\Omega(\theta')|P\mathcal{T}_\epsilon - P\mathcal{T}_0| \leq \Delta_{\max} \tag{35}$$

Apply Lemma 3 with $\Delta = \Delta_{\max}$ and we prove Theorem 1.

Figure A.1: Screenshot of the proof of Theorem 1.1.1 as given by Korattikara et al. (2014)

A.2 Derivation of the Gradient and Hessian matrix for GLM by Quiroz et al. (2018)

In their appendix, Quiroz et al. (2018) give an indication on how to compute the control variates for a Generalised Linear Model (GLM). However, it seems like their derivation of the gradient function is wrong.

Using the notation of the original article (cf Figure A.2) we indeed find that

$$\frac{\partial l}{\partial x} = \frac{\theta k'^{-1}(x^T\theta)g'(\Psi)}{g(\Psi)} + T(y)\theta k'^{-1}(x^T\theta)b'(\Psi) \tag{A.1}$$

$$= \left(\frac{g'(\Psi)}{g(\Psi)}k'^{-1}(x^T\theta) + T(y)k'^{-1}(x^T\theta)b'(\Psi)\right)\theta \tag{A.2}$$

by applying the chain rule.

S1.4. **Computing the data expanded control variates for the GLM class.** We now derive the control variates based on expansion around z for the class of Generalized Linear Models (GLM, Nelder and Wedderburn 1972). We emphasize that our method applies much more widely: the only requirement is that $\ell(z;\theta)$ is twice differentiable with respect to z. We note that categorical variables, either response or covariates, are considered as continuous in the differentiation.

Consider a univariate GLM

$$p(y|x,\theta) := h(y)g(\Psi)\exp\left(b(\Psi)T(y)\right),$$

with $\mathrm{E}[y|x] := \Psi$, with $k(\Psi) = x^T\theta$ for an invertible link function k. The log-density as a function of data $z = (y,x)^T \in \mathbb{R}^{(p+1)\times 1}$ is

$$\begin{aligned}\ell(z;\theta) &= \log(h(y)) + \log(g(\Psi)) + b(\Psi)T(y)\\ \Psi &= k^{-1}(x^T\theta).\end{aligned}$$

To save space, define

$$k^{-1'} := \frac{d}{da}k^{-1}(a)\bigg|_{a=x^T\theta} \quad \text{and} \quad k^{-1''} := \frac{d^2}{da^2}k^{-1}(a)\bigg|_{a=x^T\theta}.$$

The gradient $\nabla_z\ell(z;\theta)$ is the $\mathbb{R}^{(p+1)\times 1}$ vector

$$\begin{bmatrix}\frac{\partial\ell}{\partial y}\\ \frac{\partial\ell}{\partial x}\end{bmatrix} = \begin{bmatrix}\frac{h'(y)}{h(y)} + b(\Psi)T'(y)\\ \left(\frac{g'(\Psi)}{g(\Psi)}k^{-1'} + b'(\Psi)T(y)\right)\theta\end{bmatrix}$$

Figure A.2: Screenshot of the appendix of Quiroz et al. (2018)

Appendix B

R code

B.1 Libraries

```
library(mvtnorm)
library(coda)
library(ggpubr)
library(coda)
library(rootSolve)
library(ggplot2)
library(plyr)
```

B.2 DGP 1 : Test data - Normal Model

B.2.1 Parameters

```
n=1e4
theta=1
y=rnorm(n,theta)
z=as.matrix(y)
```

B.2.2 DGP and model functions

```
llik<-function (i,z,theta) {
  return(dnorm(z[i],mean=theta, log=TRUE))
}
#likelihood for the Pseudo-marginal algorithn
llik<-function (z,theta) {
  return(dnorm(z,mean=theta, log=TRUE))
}
log_prior<-function(theta) {
```

```
  return(dnorm(theta,mean=1, sd=0.3,log=TRUE))
}
gradient_llik<-function(z,theta) {
  return(-(z-theta))
}
hessian_llik<-function(z,theta) {
  return(rep(-1,length(z)))
}
```

B.3 DGP 2 : Bimodal - High Variance

B.3.1 Parameters

```
theta=0.1
sigma=20
n=1e4
```

B.3.2 DGP and model functions

```
y=rnorm(n,mean=abs(theta),sigma)
z=as.matrix(y)
llik<-function (i,z,theta) {
  return(dnorm(z[i],mean=abs(theta),sigma, log=TRUE))
}
#Likelihood for the Pseudo-Marginal algorithm
llik<-function (z,theta) {
  return(dnorm(z,mean=abs(theta), sd=sigma, log=TRUE))
}
log_prior<-function(theta) {
```

```
  return(dnorm(theta,mean=0,0.3,log=T))
}
gradient_llik<-function(z,theta) {
  return(-(1/sigma^2)*(z-abs(theta)))
}
hessian_llik<-function(z,theta) {
  return(rep(-(1/sigma^2),length(z)))
}
```

B.4 DGP 3 : Bimodal - Low variance

B.4.1 Parameters

```
theta=0.1
sigma=1
n=1e4
```

B.4.2 DGP and model functions

```
y=rnorm(n,mean=abs(theta),sigma)
z=as.matrix(y)
llik<-function (i,z,theta) {
  return(dnorm(z[i],mean=abs(theta),sigma, log=TRUE))
}
#Likelihood for Pseudo-Marginal algorithm
llik<-function (z,theta) {
  return(dnorm(z,mean=abs(theta), sd=sigma, log=TRUE))
}
log_prior<-function(theta) {
```

```
  return(dnorm(theta,mean=0,0.3,log=T))
}
gradient_llik<-function(z,theta) {
  return(-(1/sigma^2)*(z-abs(theta)))
}
hessian_llik<-function(z,theta) {
  return(rep(-(1/sigma^2),length(z)))
}
```

B.5 DGP 4 : High-dimensional logistic regression

B.5.1 Parameters

```
n=1000
d=20
theta=rnorm(d)
```

B.5.2 DGP and model functions

```
x=matrix(NA,n,d)
for (i in (1:d))
  x[,i]=rnorm(n)
pr=1/(1+exp(-x%*%theta))
y=rbinom(n,1,pr)
z=as.matrix(y)
llik<-function (i,z,theta) {
  l=1/(1+exp(-x[i,]%*%theta))
  out=l^(z[i,])*(1-l)^(1-z[i,])
  return(log(out))
```

```
}

log_prior<-function(theta) {

  return(dmvnorm(theta,log=TRUE))

}

sigmoid<-function(x){

  return(1/(1+exp(-x)))

}

firstDerivSigmoid<-function(x){

  return(sigmoid(x)*(1-sigmoid(x)))

}

secondDerivSigmoid<-function(x){

  return(sigmoid(x)*(1-sigmoid(x))*(1-2*sigmoid(x)))

}

inv_logistic<-function(x) {

  return(log(x/(1-x)))

}

firstDerivinv_logistic<-function(x) {

  return(1/(x*(1-x)))

}

secondDerivinv_logistic<-function(x) {

  return((2*x-1)/(x*x*(1-x)*(1-x)))

}

#likelihood function for Pseudo-marginal algorithm

llik <- function(z,theta) {

  return(log(1-sigmoid(z[(1:d)]%*%theta))+z[d+1]*(z[(1:d)]%*%theta))

}

gradient_llik<-function(z,theta) {

  return(c(z[(1:d)]%*%theta,

         as.vector(-firstDerivSigmoid(z[(1:d)]%*%theta)/

         (1-sigmoid(z[(1:d)]%*%theta))+
```

```
          firstDerivSigmoid(z[(1:d)]%*%theta)*
          firstDerivinv_logistic(sigmoid(z[(1:d)]%*%theta))*
          z[d+1])%*%theta))
}
hessian_llik<-function(z,theta) {
  H=matrix(NA, d+1, d+1)
  H[1,1]=0
  H[1,(2:(d+1))]=c(firstDerivinv_logistic(sigmoid(z[(1:d)]%*%theta))*
                     firstDerivSigmoid(z[(1:d)]%*%theta))%*%theta
  H[(2:(d+1)),1]=c(firstDerivinv_logistic(sigmoid(z[(1:d)]%*%theta))*
                     firstDerivSigmoid(z[(1:d)]%*%theta))%*%theta
  H[(2:(d+1)),(2:(d+1))]=c(-firstDerivSigmoid(z[(1:d)]%*%theta)^2/
                         (1-sigmoid(z[(1:d)]%*%theta))^2-
                         secondDerivSigmoid(z[(1:d)]%*%theta)/
                         (1-sigmoid(z[(1:d)]%*%theta))+
                         (secondDerivinv_logistic(sigmoid(z[(1:d)]%*%theta))*
                         firstDerivSigmoid(z[(1:d)]%*%theta)^2+
                         secondDerivSigmoid(z[(1:d)]%*%theta)*
                         firstDerivinv_logistic(sigmoid(z[(1:d)]%*%theta)))*
                         z[d+1])*(theta%*%t(theta))
  return(H)
  }
data=cbind(x,z)
```

In the next part we give the functions used to run each algorithm. We also give the code used to produce our plots and carry out our tests such as Geweke (2004)'s test. We provide the code for our first DGP only as it is straight forward to extend it to any other data set.

B.6 Algorithm APMHT

```
korattikara<-function(z, Time, e=0.05, m,theta_ini){
  z<-as.matrix(z)
  N=dim(z)[1]
  d=length(theta_ini)
  theta=matrix(NA,d,Time+1)
  theta[,1]=theta_ini
  number_llik_eval=0
  ptm <- proc.time()
  for (t in (1:Time)) {
    tp=rnorm(d,theta[,t],1.5) #RW proposal to be tuned
    u=runif(1)
    lbar=0
    lsqbar=0
    n=0
    done=FALSE
    mu0=(1/N)*(log(u)+log_prior(theta[,t])-log_prior(tp))
    batch=0
    index=(1:N)
    while(done==FALSE) {
      draw=sample(length(index),min(m,N-n))
      batch=base::c(index[draw],batch)
      n=n+min(m,N-n)
      index=index[-draw]
      l=llik(batch,z,tp)-llik(batch,z,theta[,t])
      lbar=mean(l)
      lsqbar=mean(l^2)
      sd_batch=sqrt((n/(n-1))*(lsqbar-lbar^2))
      sd_hat=sd_batch/sqrt(n)*sqrt(1-(n-1)/(N-1))
      delta=1-pt(abs((lbar-mu0)/sd_hat), n-1)
      if(delta<e) {
```

```
        if(lbar>mu0) {
          theta[,t+1]=tp
        }
        else {
          theta[,t+1]=theta[,t]
        }
        done=TRUE
        number_llik_eval=number_llik_eval+n
      }
    }
  }
  print(paste("Completed in ", proc.time()[3]-ptm[3]," sec", sep=""))
  return(list(theta=theta,number_llik_eval=number_llik_eval))
}


#Measures of approximation and efficiency
DGP0_korattikara<-korattikara(z,2e4,0.05,3000,theta)
ESS_korattikara<-effectiveSize(as.mcmc(DGP0_korattikara$theta[1,]))/
                (DGP0_korattikara$number_llik_eval/n)
statip::hellinger(DGP0_korattikara$theta[1,],DGP0_MH[1,])


#diagnostic plots
p1<-ggplot(data=as.data.frame(cbind(DGP0_korattikara$theta[1,],
                                    DGP0_MH[1,])))+
  geom_density(aes(DGP0_korattikara$theta[1,],color="ApMHT"))+
  geom_density(aes(DGP0_MH[1,],color="MH"))+
  ylab("Density")+
  xlab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("ApMHT"="dodgerblue4",
```

```
                                "MH"="red3"),
                      labels=c("ApMHT","MH"))+
  coord_flip()
p2<-ggplot(data=as.data.frame(DGP0_korattikara$theta[1,]))+
  geom_line(aes(y=DGP0_korattikara$theta[1,], x=(1:20001),
                color="ApMHT"))+
  xlab("Time")+
  ylab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("ApMHT"="dodgerblue4"))
ggarrange(p1, p2, nrow=2, common.legend = TRUE, legend="bottom")


#Geweke's test
DGP0_GITkorattikara=numeric(100)
for (i in (0:100)) {
  phi=rnorm(1) #Simulate from prior
  y_sim=rnorm(n,mean = abs(phi),sigma)
  DGP0_GITkorattikara[i]=korattikara(y_sim,5000,0.05,1000,0)$
    theta[1,sample((500:5000),1)]
  print(i)
}
ks.test(DGP0_GITkorattikara,pnorm,1,1) #Kolmogorov-Smirnov test
prior_sim=rnorm(100)
ggplot(data=as.data.frame(cbind(prior_sim,DGP0_GITkorattikara)))+
  geom_density(aes(prior_sim, color="Prior"))+
  geom_density(aes(DGP0_GITkorattikara,color="Marginal"))+
  scale_colour_manual(name="",
                      values=c("Prior"="red3",
                               "Marginal"="dodgerblue4"),
                      labels=c(expression(paste("Marginal of", " ", theta))
```

```
                                ,"Prior"))+
  theme(legend.position="bottom")+
  xlab(expression(theta))+
  ylab("Density")
```

B.7 Pseudo-marginal algorithm

B.7.1 Pre-conditioning of the data set

```
#Euclidean distance function
dist_eucl<-function(x1,x2) {
  out=0
  if (is.vector(x1)) {
    out=sum((x1-x2)^2)
  }
  else {
    out=sweep(x1,2,x2)
    out=apply(out^2,1,sum)
  }
  return(sqrt(out))
}


#Clustering Function
clustering <- function(z,eps) {
  # y<-scale(y) SCALE DATA SET
  # x<-scale(x)
  r<-as.matrix(z)
  n=dim(r)[1]
  I=rep(0,n) #Indicator variable, 1 if obs i has been given a cluster
  z=0 #centroids
```

```
  C=0 #indices inside each clusters
  k=0
  index=seq(1,n,1)
  for (j in (1:n)) {
    if(I[j]==0) {
      p<-which(dist_eucl(r[as.logical(1-I),],r[j,])<=eps, arr.ind = TRUE)
      C_current=index[p]
      index=index[-p]
      if (length(C_current)==1)
        z=cbind(r[C_current,],z)
      else
        z=cbind((1/length(C_current))*
                  apply(as.matrix(r[C_current,]),2,sum),z)
      I[C_current]=1
      C=rbind.fill(as.data.frame(t(C_current)),as.data.frame(C))
      k=k+1
    }
  }
  return(list(K=k,z=z,C=t(C)))
}


#Conputation of matrices B
 B=list()
  for (k in (1:(out$K))) {
    B[[k]]=matrix(0,dim(data)[2],dim(data)[2])
    for (i in na.omit(out$C[,k])) {
      b=data[i,]-out$z[,k]
      B[[k]]=B[[k]]+b%*%t(b)
    }
  }
```

```
out<-clustering(z,0.05)
data=z
```

B.7.2 Function

```
quiroz<-function(data,z,K,C,m, Time, theta_ini) {
  N=dim(data)[1]
  d=length(theta_ini)
  d_b=rep(0,m)
  d_bp=rep(0,m)
  theta=matrix(NA,nrow=length(theta_ini), ncol=Time+1)
  theta[,1]=theta_ini
  for(t in (1:Time)) {
    bp=rnorm(d,theta[,t],0.2) #RW proposal to be tuned
    u=sample((1:N),m,replace=TRUE)
    for (i in (1:m)) {
      coord=which(out$C==u[i],arr.ind = TRUE)[2]
      d_b[i]=llik(data[u[i],],theta[,t])-llik(z[,coord],theta[,t])-
        gradient_llik(z=z[,coord],theta=theta[,t]) %*%
        (data[u[i],]-z[,coord])-
        0.5*t(data[u[i],]-z[,coord])%*%hessian_llik(z=z[,coord],
                                                    theta=theta[,t])%*%
        (data[u[i],]-z[,coord])
      d_bp[i]=llik(data[u[i],],bp)-llik(z[,coord],bp)-
        gradient_llik(z=z[,coord],theta=bp)%*%(data[u[i],]-z[,coord])-
        0.5*t(data[u[i],]-z[,coord])%*%hessian_llik(z=z[,coord],
                                                    theta=bp)%*%
        (data[u[i],]-z[,coord])
    }
```

```
  mu_b=mean(d_b)
  sigma_b=var(d_b)
  mu_bp=mean(d_bp)
  sigma_bp=var(d_bp)
  first_term_b=0
  first_term_bp=0
  third_term_b=0
  third_term_bp=0
  for (k in (1:K)) {
    Nk=length(na.omit(out$C[,k]))
    first_term_b=first_term_b+Nk*llik(z[,k],theta[,t])
    first_term_bp=first_term_bp+Nk*llik(z[,k],bp)
    H_bp=hessian_llik(z=z[,k],theta=bp)
    H_b=hessian_llik(z=z[,k],theta=theta[,t])
    third_term_b=third_term_b+H_b*B[[k]]
    third_term_bp=third_term_bp+H_bp*B[[k]]
  }
  q_b=first_term_b+0.5*sum(rowSums(third_term_b))
  q_bp=first_term_bp+0.5*sum(rowSums(third_term_bp))
  l_hat_b=q_b+N*mu_b-(N^2)/(2*m)*sigma_b
  l_hat_bp=q_bp+N*mu_bp-(N^2)/(2*m)*sigma_bp
  prob=(l_hat_bp+log_prior(bp))-(l_hat_b+log_prior(theta[,t]))
  if(log(runif(1))<=prob) {
    theta[,t+1]=bp
  }
  else {
    theta[,t+1]=theta[,t]
  }
  print(t)
}
```

```
  return(theta)
}
 DGP0_quiroz<-quiroz(data,out$z,out$K,out$C,300, 20000, theta)
#Measures of approximation and efficiency
ESS_quiroz=effectiveSize(as.mcmc(DGP0_quiroz[1,]))/(20000*(8*300+4*out$K)/n)
plot(DGP0_quiroz[1,],type="l")
statip::hellinger(DGP0_quiroz[1,],DGP0_MH[1,])


#Diagnostic plots
p1<-ggplot(data=as.data.frame(cbind(DGP0_quiroz[1,],DGP0_MH[1,])))+
  geom_density(aes(DGP0_quiroz[1,],color="PS"))+
  geom_density(aes(DGP0_MH[1,],color="MH"))+
  ylab("Density")+
  xlab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("PS"="dodgerblue4",
                               "MH"="red3"),
                      labels=c("MH","Pseudo-Marginal MCMC"))+
  coord_flip()
p2<-ggplot(data=as.data.frame(DGP0_quiroz[1,]))+
  geom_line(aes(y=DGP0_quiroz[1,], x=(1:20001),color="PS"))+
  xlab("Time")+
  ylab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("PS"="dodgerblue4"))
ggarrange(p1, p2, nrow=2, common.legend = TRUE, legend="bottom")


#Geweke's test
DGP0_GITquiroz=numeric(300)
```

```
for (a in (1:300)) {
  phi=rnorm(1,mean=1,sd=1)
  y_sim=rnorm(n,mean = phi,1)
  y_sim=as.matrix(y_sim)
  out<-clustering(y_sim,0.2)
  data=y_sim
  B=list()
  for (k in (1:(out$K))) {
    B[[k]]=matrix(0,dim(data)[2],dim(data)[2])
    for (i in na.omit(out$C[,k])) {
      b=data[i,]-out$z[,k]
      B[[k]]=B[[k]]+b%*%t(b)
    }
  }
  DGP0_GITquiroz[a]=quiroz(data,out$z,out$K,out$C,300, 1000, 0)$
    theta[1,sample((100:1000),1)]
  print(a)
}
#Kolmogorov-Smirnov test
ks.test(DGP0_GITquiroz,pnorm,1,1)
prior_sim=rnorm(100,1)
ggplot(data=as.data.frame(cbind(prior_sim,DGP0_GITquiroz)))+
  geom_density(aes(prior_sim, color="Prior"))+
  geom_density(aes(DGP0_GITquiroz,color="Marginal"))+
  scale_colour_manual(name="",
                      values=c("Prior"="red3",
                               "Marginal"="dodgerblue4"),
                      labels=c(expression(paste("Marginal of", " ", theta)),
                               "Prior"))+
  theme(legend.position="bottom")+
```

```
  xlab(expression(theta))+
  ylab("Density")
```

B.8 SGLD

```
SGLD<-function(z, eps=0.0001, Time=1000, m=1000, theta_ini=0) {
 z<-as.matrix(z)
 N=dim(z)[1]
 d=length(theta_ini)
 theta=matrix(NA,nrow=d, ncol=Time+1)
 theta[,1]=theta_ini
 for(i in (2:(Time+1))) {
   batch=sample((1:N),m)
   theta[,i]=theta[,i-1] +
     (eps/2)*(gradient(log_prior,theta[,i-1])
                                    +(N/m)*apply(
                                      gradient(llik,x=theta[,i-1],i=batch,z=z),
                                      2,
                                      sum)
                                    +rnorm(d,mean=0,eps))
 }
 return(theta)
}


DGP0_SGLD<-SGLD(z, eps=2*1e-4,Time=20000,m=5000,theta)
#Measures of approximation and efficiency
ESS_SGLD<-effectiveSize(as.mcmc(DGP0_SGLD[1,]))/((Time*2*2000)/n)
statip::hellinger(DGP0_SGLD[1,],DGP0_MH[1,])
#Diagnostic plots
```

```
p1<-ggplot(data=as.data.frame(cbind(DGP0_SGLD[1,],DGP0_MH[1,])))+
  geom_density(aes(DGP0_MH[1,],color="MH"))+
  geom_density(aes(DGP0_SGLD[1,],color="SGLD"))+
  ylab("Density")+
  xlab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("SGLD"="dodgerblue4",
                               "MH"="red3"),
                      labels=c("MH","SGLD"))+
  coord_flip()


p2<-ggplot(data=as.data.frame(DGP0_SGLD[1,]))+
  geom_line(aes(y=DGP0_SGLD[1,], x=(1:20001),color="SGLD"))+
  xlab("Time")+
  ylab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("SGLD"="dodgerblue4"))
ggarrange(p1, p2, nrow=2, common.legend = TRUE, legend="bottom")


#Geweke's test
DGP0_GITSGLD=numeric(100)
for (i in (1:100)) {
  phi=rnorm(1,mean=1,sd=1)
  y_sim=rnorm(n,mean = phi,1)
  DGP0_GITSGLD[i]=SGLD(y_sim, eps=2*1e-4,
                       Time=2000,m=5000,0)[1,sample((500:2000),1)]
  print(i)
}
#Kolmogorov-Smirnov test
ks.test(DGP0_GITSGLD,pnorm,1,1)
```

```r
prior_sim=rnorm(1e2,1,1)
ggplot(data=as.data.frame(cbind(prior_sim,DGP0_GITSGLD)))+
  geom_density(aes(prior_sim, color="Prior"))+
  geom_density(aes(DGP0_GITSGLD,color="Marginal"))+
  scale_colour_manual(name="",
                      values=c("Prior"="red3",
                               "Marginal"="dodgerblue4"),
                      labels=c(expression(paste("Marginal of", " ", theta)),
                               "Prior"))+
  theme(legend.position="bottom")+
  xlab(expression(theta))+
  ylab("Density")
```

B.9 Consensus Monte-Carlo

```r
#Function running MCMC on separate batches of the data set
consensus_batch<-function(z,s,Time,theta_ini){
 z<-as.matrix(z)
 N=dim(z)[1]
 shuffle=sample((1:N),N,replace=FALSE)
 d=length(theta_ini)
 theta=list()
 S=list()
 W=list()
 ptm <- proc.time()
 for (i in (1:s)) {
   S[[i]]=shuffle[(((i-1)*(N-N%%s)/s+1):(i*(N-N%%s)/s))]#set of indices
   theta[[i]]=matrix(NA,nrow=d,ncol=Time+1)
   theta[[i]][,1]=theta_ini
```

```
  llk_current=sum(llik(S[[i]],z,theta_ini))
  for (t in (1:Time)) {
    tp=rnorm(d,theta[[i]][,t],0.16)
    llk_proposal=sum(llik(S[[i]],z,tp))
    ratio=(llk_proposal+(1/s)*log_prior(tp))-
      (llk_current+(1/s)*log_prior(theta[[i]][,t]))
    if(log(runif(1))<=ratio)  {
      theta[[i]][,t+1]=tp
      llk_current=llk_proposal
    }
    else
      theta[[i]][,t+1]=theta[[i]][,t]
  }
  W[[i]]=solve(var(t(theta[[i]])))
 }
 print(paste("Each parallel computing was completed in ",
             (proc.time()[3]-ptm[3])/s," sec", sep=""))
 return(list(theta=theta,W=W))
}
#Function to combine the samples
consensus<-function(theta,W){
 ptm<-proc.time()
 Time=dim(theta[[1]])[2]
 d=dim(theta[[1]])[1]
 s=length(theta)
 norm_const=Reduce('+',W)
 norm_const=solve(norm_const)
 out=0
 for (i in (1:s)) {
   out=out+W[[i]]%*%theta[[i]]
```

```
 }
 out=norm_const%*%out
 print(paste("Combining part completed in ", proc.time()[3]-ptm[3]," sec", sep=""))
 return(out)
}
Time=20000
s=10
DGP0_consensus=consensus_batch(z,s,Time,theta)
plot(DGP0_consensus$theta[[1]][1,],type="l")
ggplot(data=as.data.frame(DGP0_consensus$theta[[1]][1,]),
       aes(DGP0_consensus$theta[[1]][1,]))+
 geom_density()
DGP0_consensus=consensus(DGP0_consensus$theta,DGP0_consensus$W)
#Measures of approximation and efficiency
ESS_consensus=effectiveSize(as.mcmc(DGP0_consensus[1,]))/(Time*((n-n%%s)/s)/n)
statip::hellinger(DGP0_consensus[1,],DGP0_MH[1,])
#Diagnostic plots
p1<-ggplot(data=as.data.frame(cbind(DGP0_consensus[1,],DGP0_MH[1,])))+
 geom_density(aes(DGP0_consensus[1,],color="Consensus"))+
 geom_density(aes(DGP0_MH[1,],color="MH"))+
 ylab("Density")+
 xlab(expression(theta))+
 scale_colour_manual(name="",
                     values=c("Consensus"="dodgerblue4",
                              "MH"="red3"),
                     labels=c("Consensus","MH"))+
 coord_flip()
p2<-ggplot(data=as.data.frame(DGP0_consensus[1,]))+
 geom_line(aes(y=DGP0_consensus[1,], x=(1:20001),color="Consensus"))+
 xlab("Time")+
```

```
 ylab(expression(theta))+
 scale_colour_manual(name="",
                     values=c("Consensus"="dodgerblue4"))
ggarrange(p1, p2, nrow=2, common.legend = TRUE, legend="bottom")


#Geweke's test
DGP0_GITconsensus=numeric(100)
for (i in (1:100)) {
 phi=rnorm(1,mean=1,sd=1)
 y_sim=rnorm(n,phi)
 temp=consensus_batch(y_sim,10,3000,theta)
 DGP0_GITconsensus[i]=consensus(temp$theta,temp$W)[1,sample((500:3000),1)]
 print(i)
}
#Kolmogorov-Smirnov test
ks.test(DGP0_GITconsensus,pnorm,1,1)
prior_sim=rnorm(100,1,1)
ggplot(data=as.data.frame(cbind(prior_sim,DGP0_GITconsensus)))+
 geom_density(aes(prior_sim, color="Prior"))+
 geom_density(aes(DGP0_GITconsensus,color="Marginal"))+
 scale_colour_manual(name="",
                     values=c("Prior"="red3",
                              "Marginal"="dodgerblue4"),
                     labels=c(expression(paste("Marginal of", " ", theta)),
                              "Prior"))+
 theme(legend.position="bottom")+
 xlab(expression(theta))+
 ylab("Density")
```

B.10 NPDPE

```
#Function to run MH on several small batches
batch_MCMC<-function(z,s,Time,theta_ini) {
  z<-as.matrix(z)
  N=dim(z)[1]
  shuffle=sample((1:N),N,replace=FALSE)
  d=length(theta_ini)
  theta=list()
  S=list()
  ptm <- proc.time()
  for (i in (1:s)) {
    S[[i]]=shuffle[(((i-1)*(N-N%%s)/s+1):(i*(N-N%%s)/s))]#set of indices
    theta[[i]]=matrix(NA,nrow=d,ncol=Time+1)
    theta[[i]][,1]=theta_ini
    llk_current=sum(llik(S[[i]],z,theta_ini))
    for (t in (1:Time)) {
      tp=rnorm(d,theta[[i]][,t],0.16)
      llk_proposal=sum(llik(S[[i]],z,tp))
      ratio=(llk_proposal+(1/s)*log_prior(tp))-
        (llk_current+(1/s)*log_prior(theta[[i]][,t]))
      if(log(runif(1))<=ratio)  {
        theta[[i]][,t+1]=tp
        llk_current=llk_proposal
      }
      else
        theta[[i]][,t+1]=theta[[i]][,t]
    }
  }
```

```
  print(paste("Each parallel computing was completed in ",
              (proc.time()[3]-ptm[3])/s," sec", sep=""))
  return(theta)
}
#Function to combine the samples
gaussian_kernel<-function(theta) {
  ptm<-proc.time()
  Time=dim(as.matrix(theta[[1]]))[2]
  d=dim(as.matrix(theta[[1]]))[1]
  s=length(theta)
  t=sample((1:Time),s,replace = TRUE)
  DGP0_neiswanger=matrix(NA,nrow=d,ncol=Time)
  for(i in (1:Time)) {
    h=1e-2
    for(k in (1:s)) {
      c<-t
      c[k]=sample((1:Time),1,replace = TRUE)
      theta_bar_t=0
      theta_bar_c=0
      w_t=0
      w_c=0
      for (j in (1:s)) {
        theta_bar_t=theta_bar_t+theta[[j]][,t[j]]
        theta_bar_c=theta_bar_c+theta[[j]][,c[j]]
      }
      theta_bar_t=(1/s)*theta_bar_t
      theta_bar_c=(1/s)*theta_bar_c
      for (j in (1:s)) {
        w_t=w_t + dmvnorm(theta[[j]][,t[j]],mean=theta_bar_t,sigma=h^2*diag(d),
                          log=TRUE)
```

```
        w_c=w_c + dmvnorm(theta[[j]][,c[j]],mean=theta_bar_c,sigma=h^2*diag(d),
                          log=TRUE)
      }
      if(log(runif(1))<w_c-w_t) {
        t<-c
        theta_bar_t<-theta_bar_c
      }
    }
    DGP0_neiswanger[,i]=rmvnorm(1,mean=theta_bar_t,sigma=(h^2/s)*diag(d))
  }
  print(paste("Combining part completed in ", proc.time()[3]-ptm[3]," sec", sep=""))
  return(DGP0_neiswanger)
}


Time=20000
s=10
DGP0_neiswanger_batch=batch_MCMC(z,10,20000,theta)
plot(DGP0_neiswanger_batch[[3]][1,],type="l")
1 - rejectionRate(as.mcmc(DGP0_neiswanger_batch[[1]][1,]))
DGP0_neiswanger=gaussian_kernel(DGP0_neiswanger_batch)
#Measures of approximation and efficiency
ESS_neiswanger=effectiveSize(as.mcmc(DGP0_neiswanger[1,]))/(Time*((n-n%%s)/s)/n)
statip::hellinger(DGP0_neiswanger[1,],DGP0_MH[1,])
#Diagnostic plots
p1<-ggplot(data=as.data.frame(cbind(DGP0_neiswanger[1,],DGP0_MH[1,])))+
  geom_density(aes(DGP0_neiswanger[1,],color="NPDPE"))+
  geom_density(aes(DGP0_MH[1,],color="MH"))+
  ylab("Density")+
  xlab(expression(theta))+
  scale_color_manual(name="",
```

```
                      values=c("NPDPE"="dodgerblue4",
                               "MH"="red3"
                      ),
                      labels=c("MH","NPDPE"))+
  coord_flip()
p2<-ggplot(data=as.data.frame(DGP0_neiswanger[1,]))+
  geom_line(aes(y=DGP0_neiswanger[1,], x=(1:20001),color="NPDPE"))+
  xlab("Time")+
  ylab(expression(theta))+
  scale_colour_manual(name="",
                      values=c("NPDPE"="dodgerblue4"))
ggarrange(p1, p2, nrow=2, common.legend = TRUE, legend="bottom")


#Geweke's test
DGP0_GITneiswanger=numeric(100)
for (i in (1:100)) {
  phi=rnorm(1,mean=1,sd=1)
  y_sim=rnorm(n,mean = phi,1)
  temp=batch_MCMC(y_sim,10,3000,0)
  DGP0_GITneiswanger[i]=gaussian_kernel(temp)[1,sample((500:3000),1)]
  print(i)
}
#Kolmogorov-Smirnov test
ks.test(DGP0_GITneiswanger[(1:100)],pnorm,1,1)
prior_sim=rnorm(100,1,1)
ggplot(data=as.data.frame(cbind(prior_sim,DGP0_GITneiswanger)))+
  geom_density(aes(prior_sim[(1:100)], color="Prior"))+
  geom_density(aes(DGP0_GITneiswanger,color="Marginal"))+
  scale_colour_manual(name="",
                      values=c("Prior"="red3",
```

```
                              "Marginal"="dodgerblue4"),
                    labels=c(expression(paste("Marginal of", " ", theta)),"Prior"))+
theme(legend.position="bottom")+
xlab(expression(theta))+
ylab("Density")
```