TECHNIQUES OF PARALLELIZATION
IN MARKOV CHAIN MONTE CARLO METHODS

By

VIKNESWARAN GOPAL

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
UNIVERSITY OF FLORIDA
2011
To my loving wife and parents
ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my committee members for their guidance. In particular to my advisor - who inspired me to take up statistics even before I met him. It is no exaggeration to state that this journey would not have been possible without his coaching, encouragement and tremendous understanding.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>RELEVANT MARKOV CHAIN THEORY</td>
<td>16</td>
</tr>
<tr>
<td>2.1</td>
<td>Regenerative Markov Chains</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Proving Geometric Ergodicity of a Markov Chain</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Regeneration in Parallel</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Minorization of Common Markov Chain Monte Carlo Samplers</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>Finite State Samplers with Mixture Candidates</td>
<td>24</td>
</tr>
<tr>
<td>2.6</td>
<td>Summary</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION ASPECTS OF PARALLEL REGENERATION</td>
<td>28</td>
</tr>
<tr>
<td>3.1</td>
<td>Achievable Speed-up from Parallelization</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Completing Unfinished Tours</td>
<td>31</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Renewal Theory</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Estimating Residual Life and Residual Reward</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Possible Diagnostics</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>R PACKAGE FOR PARALLEL REGENERATIVE MARKOV CHAINS</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Code and Method Validation</td>
<td>42</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Minorizing the Oneway Model</td>
<td>45</td>
</tr>
<tr>
<td>4.1.2</td>
<td>The Experiment</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>Hierarchical Linear Mixed Model (HLMM)</td>
<td>49</td>
</tr>
<tr>
<td>4.3</td>
<td>Topic Models</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>PIPELINED ADAPTIVE MARKOV CHAIN MONTE CARLO</td>
<td>68</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Stochastic Approximation</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>The Ordinary Differential Equation Method of Proof</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>Pipeline Version of Stochastic Approximation</td>
<td>76</td>
</tr>
<tr>
<td>5.5</td>
<td>Convergence Rate</td>
<td>79</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>5.6</td>
<td>Simulations</td>
<td>81</td>
</tr>
<tr>
<td>5.7</td>
<td>Summary</td>
<td>81</td>
</tr>
<tr>
<td>6</td>
<td>FUTURE WORK</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>APPENDIX: R PACKAGE DOCUMENTATION</td>
<td>85</td>
</tr>
<tr>
<td>A.1</td>
<td>Overview</td>
<td>85</td>
</tr>
<tr>
<td>A.2</td>
<td>Inner Workings of the Code</td>
<td>85</td>
</tr>
<tr>
<td>A.3</td>
<td>User Interface to Package Functions</td>
<td>86</td>
</tr>
<tr>
<td>A.4</td>
<td>Interactive Sessions</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>LIST OF REFERENCES</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>BIOGRAPHICAL SKETCH</td>
<td>93</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Taxonomy of parallel processing architectures</td>
<td>15</td>
</tr>
<tr>
<td>4-1</td>
<td>Numerically computed posterior means for test dataset</td>
<td>62</td>
</tr>
<tr>
<td>4-2</td>
<td>Regenerative method coverage probabilities</td>
<td>62</td>
</tr>
<tr>
<td>4-3</td>
<td>Timings for oneway model</td>
<td>63</td>
</tr>
<tr>
<td>4-4</td>
<td>REML estimates for petrol data model</td>
<td>63</td>
</tr>
<tr>
<td>4-5</td>
<td>ML estimates for petrol data model</td>
<td>63</td>
</tr>
<tr>
<td>4-6</td>
<td>Petrol data regenerative estimates</td>
<td>63</td>
</tr>
<tr>
<td>4-7</td>
<td>Timings for petrol data</td>
<td>64</td>
</tr>
<tr>
<td>4-8</td>
<td>Index notation for Latent Dirichlet Allocation model</td>
<td>64</td>
</tr>
<tr>
<td>4-9</td>
<td>Timings for topic model simulated data</td>
<td>64</td>
</tr>
<tr>
<td>4-10</td>
<td>Topic model simulated data regenerative estimates</td>
<td>65</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Speed-up from parallel regeneration</td>
<td>41</td>
</tr>
<tr>
<td>4-1</td>
<td>Coefficient of variation boxplots</td>
<td>64</td>
</tr>
<tr>
<td>4-2</td>
<td>Scaled Regeneration Quantile Plots</td>
<td>65</td>
</tr>
<tr>
<td>4-3</td>
<td>Proximity to theorised Normal approximation</td>
<td>66</td>
</tr>
<tr>
<td>4-4</td>
<td>Multi-modality of LDA model</td>
<td>67</td>
</tr>
<tr>
<td>5-1</td>
<td>Pipelined adaptive Markov chain Monte Carlo</td>
<td>82</td>
</tr>
<tr>
<td>A-1</td>
<td>Flowchart of McParre functions</td>
<td>88</td>
</tr>
</tbody>
</table>
Parallel computing is at the forefront of statistical research today. The main reason for this is the fact that it is the most scalable solution, both in terms of cost and computational ability, to the large applied problems that are being solved by contemporary researchers. As many of these solutions utilize Markov chain Monte Carlo techniques, here we investigate methods of running these chains in parallel. For a geometrically ergodic Markov chain, we begin by establishing that the most theoretically sound method of running it in parallel is through the use of regeneration, which divides the chain into truly independent blocks, or tours. Then we continue by investigating some operational issues relevant to parallelizing the regenerative method. Using renewal theory, we show that it is possible to use the information from the completed tours to complete the unfinished ones. We also derive a lower bound on the possible speed-up that can be attained by using our parallel algorithm, as compared with the sequential one. Finally, we document and provide a general purpose software package for running a minorized geometrically ergodic Markov chain on a cluster. We demonstrate our parallel methodology and the use of our software package on two problems of general interest - hierarchical linear mixed models and topic models. Finally, as an aside, we also investigate running adaptive Markov chains using a pipeline algorithm. We show that it has similar convergence properties to the sequential algorithm, but provides a substantial speed-up in clock-time.
CHAPTER 1
INTRODUCTION

There is no doubt that the role of parallel computing in statistical computations is getting bigger by the day. Journal papers that exhort the use of parallel processors in statistical applications, such as Suchard et al. (2010) and Zhou et al. (2010), have become more frequent in recent times. What exactly is driving this need for parallel computing? All signs point to increasingly large datasets, and to the need to solve problems of increasing computational complexity. In the field of genetics, for example, it can be argued that datasets are increasing in size faster than the processing hardware is improving to handle them. On the other hand, methodologies such as bootstrapping require more computing resources than typical statistical techniques, simply by design. Similarly, Monte Carlo simulation techniques allow more sophisticated analyses of data, but again, they demand more processing power. These would naturally point to a need to boost the speed of our individual processors, but the trend is very clearly towards linking up processors of moderate speed rather than a concerted effort to increase the speed of individual processors. The reason for this is simple: The former option can be executed at a fraction of the cost of the latter. Doubling the speed of a processor is costly because it requires memory access speeds and cooling to be concomitantly improved. Although adding another processor would require a rewrite of the programs in order to fully optimize the new parallel processing environment, it is inordinately cheap, easy to do and is a scalable solution. This explains why supercomputers are prohibitively expensive except to dedicated research institutes, whereas cluster computing possibilities are readily available and accessible. Most universities have their own cluster, while companies such as Amazon, IBM and now HP, have made cloud computing available to anyone at an extremely cheap cost. For this reason too, our desktops and notebooks already contain multiple processors.
For a comprehensive review of the many parallel computing techniques at our disposal today, the reader is referred to Kontoghiorghes (2006) and Matloff (2011). In this section we shall touch on some of the topics in those references, and in Suchard et al. (2010), but our main aim is to introduce the parallel computing architecture and approach that we have chosen for our purpose.

Years ago, a taxonomy of parallel processing architectures was put forward in Flynn (1972). Today, the dividing lines are not as clean as before since many contemporary parallel set-ups are hybrids of those outlined in that seminal paper. However the original breakdown is still a good starting point. Referring to Table 1-1, models for parallel computation can be categorised into 4 basic kinds. Single-core desktops and notebooks from a few years ago fall under the SISD category. They contain one processor, and they execute one instruction at a time. In essence, they are not parallel. The MISD model is used only in specialised applications, such as when the algorithm can be run in a pipeline. It is also used in cases where multiple processors are called on to carry out the same computation on the same data in order to detect and mask errors. This is sometimes termed fault-tolerant computing. The last two types of parallel processing architectures, SIMD and MIMD, are also the most widely used ones. GPU processing is one good example of an SIMD machine. A specific example of its working is when the hundreds of processors on the GPU work on refreshing separate portions of the viewing display. An example of an MIMD machine is a Beowulf cluster. The MIMD model can be further sub-divided into shared memory machines and distributed memory ones. Distributed memory MIMD machines typically co-ordinate their work by passing messages to one another.

Which architecture is chosen depends on the algorithm we intend to parallelise, and how we intend to parallelise it. In a medium-grain parallelization, it is possible to divide

---

1 A cluster of computers connected by a fast network
the domain of work into independent data “chunks” and work on them concurrently. In such problems, it is natural to utilize the SIMD architecture, in which each processor can access and work on the disparate chunks. In a more coarse-grained parallelization, where individual functions or procedures are to be run concurrently, the MIMD model is more appropriate.

Our intention is to tackle the problem of running a Markov chain Monte Carlo (MCMC) algorithm in parallel. It is not immediately apparent how such an algorithm can be adapted to run in parallel, as such procedures are inherently sequential in the sense that they require the current state to be generated before proceeding. The goal here is to study the best method of adapting a general class of MCMC algorithms so as to fully capitalise on a parallel computing environment.

There have been several attempts to run a Markov chain in parallel when it is used to perform a stochastic search. A typical scenario is variable selection. The authors in Hans et al. (2007) developed a shotgun stochastic search algorithm to look for regression models in parallel. Their method is a Markov chain since the future configurations of variables only depends on the current state, but it is neither a Metropolis nor a Gibbs algorithm and thus its properties are difficult to analyze. However, our immediate aim is to leverage on parallel computing resources when we wish to carry out estimation using a Markov chain rather than a stochastic search.

One notable attempt at running a Metropolis-Hastings chain in parallel can be found in Brockwell (2006), where the algorithm calls on a cluster to pre-compute likelihoods of the candidate distribution and thus save time. It has been extended in Strid (2010) to the case where the current state is used to make “better” future candidate predictions. By “better”, the authors mean that they can tune the candidate until an optimal acceptance rate is achieved. However, this destroys the Markov-ian nature of the chain, and would fall under the category of adaptive algorithms rather than a traditional MCMC algorithm. Another point to note is that this algorithm is applicable only when the computation
of the likelihood is very slow. Only then would we be able to see a benefit in using this parallel version instead of the serial one. Since the algorithm requires all the possible immediate states to be broadcast to the compute nodes and then returned after computation, we would also need the inter-connections between the nodes to be fast compared to the likelihood computation time.

Another approach can be found in Ren and Orkoulas (2007). The authors here parallelize a Metropolis algorithm by decomposing the domain, and sampling within the disparate regions on separate nodes. They propose switching the regions around randomly, but they acknowledge that this would introduce another time sink via the messages being passed.

Apart from these, the most direct method of transplanting an MCMC algorithm onto a cluster would be to run one chain of length $N$ on each of $n$ processors, and then use the mean of the means as a point estimate of the desired integral, and the standard error of the observed means as an estimate of the variability in order to get a confidence interval. Although appealing in its simplicity, this method provides no savings in terms of computing time and it will not always provide accurate results. For example, in Fishman (2001) (chapter 6), it is shown that under mild assumptions on the the rate at which the asymptotic bias reduces to 0, as the number of chains $n$ increases to infinity, the burnin in each chain, $k$, has to increase faster than $\log n$ in order for the coverage probability of the confidence interval to hold. Thus it almost defeats the purpose of running the chain on a cluster, where we hoped that having more processors would have gained us time and/or precision.

Regenerative simulation, on the other hand, provides a very clean way of parallelising MCMC algorithms. At every regeneration point, the random variable generated is independent of its current state, and everything before it. Hence regeneration points provide us with independent segments or “tours” that can be run on separate processors, and then concatenated later!
Our intention is to run individual tours on separate processors and then join them together. This is a coarse-grain parallelization of the MCMC algorithm, and hence is most suited to the MIMD distributed memory model architecture. There are instances where MCMC algorithms have been run on GPUs, i.e. on an SIMD model. The best example we know of is in Suchard et al. (2010), where a fine grain parallelization is applied to a Gibbs sampler in order to parallelize and provide a speed up within the iterations. GPUs afford an incredible speed-up. However, they require very careful implementation of an algorithm, which translates to a long development time. What we aim for is to allow a researcher to run a regenerative MCMC algorithm, as long as they can provide the minorization (see Chapter 2 for the definition and explanation of required MCMC concepts). Any increase in processing power (by adding nodes) will then provide a speed-up by decreasing the execution time to run a fixed number of tours.

The rest of this report is structured as follows. In Chapter 2, we provide a review of the theoretical background for regenerating Markov chains and formalize the idea of running them in parallel. In Chapter 3, we investigate some properties of parallel regenerative Markov chains. First, we touch on the possible speed-up we can gain over the serial version of the algorithm. Second, we investigate if it possible to use information from incomplete tours. This is relevant as our proposal to increase the number of processors would increase the number of incomplete tours we are left with. In chapter 4, we present the R package McParre for Markov Chain PArallel REgeneration. We introduce the parallel programming paradigm that it uses and demonstrate its use and the speed up it provides via three models. Lastly, in Chapter 5, we discuss an idea unrelated to regenerative chains - accelerating adaptive Markov chains using a pipeline algorithm on a cluster.
Table 1-1. Taxonomy of parallel processing architectures as introduced in Flynn (1972)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Parallel Computing Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD</td>
<td>Single Instruction Single Datastream</td>
</tr>
<tr>
<td>MISD</td>
<td>Multiple Instruction Single Datastream</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Datastream</td>
</tr>
<tr>
<td>MIMD</td>
<td>Multiple Instruction Multiple Datastream</td>
</tr>
</tbody>
</table>
CHAPTER 2
RELEVANT MARKOV CHAIN THEORY

2.1 Regenerative Markov Chains

This section, which is mostly condensed from Hobert et al. (2002), Jones and Hobert (2001) and Mykland et al. (1995), reviews what needs to be done before a regeneration-based Central Limit Theorem (CLT) can be applied to a particular MCMC sampler. Briefly speaking, first of all, it has to be shown that the Markov chain is geometrically ergodic. Next, it has to be ascertained that the function whose integral we wish to find has slightly more than a 2nd moment. Finally, the Markov chain has to be “split” into independent and identically distributed tours during its run.

Now we go into some detail regarding the above steps and in the process introduce the notation that will be used. Let \( \Phi = \{X_0, X_1, \ldots\} \) be a Markov chain taking values in the sample space \((E, \mathcal{E})\) with transition kernel \(P(\cdot, \cdot)\), where \(E\) is typically \(\mathbb{R}^k\) and \(\mathcal{E}\) is the Borel set on \(E\). Also assume that \(\Phi\) possesses the following properties:

1. It has a stationary distribution \(\pi\). In other words, \(\pi(A) = \int P(x, A)\pi(dx)\) for all \(A \in \mathcal{E}\).
2. It is \(\pi\)-irreducible, aperiodic and Harris recurrent.

Suppose we wish to use an MCMC algorithm to estimate \(E_\pi g = \int g(x)\pi(dx)\), and we know that \(\int |g(x)|\pi(dx) < \infty\). The assumptions above imply that we have an ergodic theorem, which yields

\[
\bar{g} = \frac{1}{n} \sum_{i=0}^{n-1} g(X_i) \rightarrow E_\pi g \text{ a.s.}
\]

If in addition we have that \(E_\pi |g|^{2+\epsilon} < \infty\) for some \(\epsilon > 0\), then we can apply a Central Limit Theorem (CLT) to obtain a Confidence Interval for \(\bar{g}\).

\[
\sqrt{n}(\bar{g} - E_\pi g) \sim N(0, \gamma^2)
\]  

(2–1)
where $\gamma^2 = \text{Var}_\pi g(X_0) + 2 \sum_{i=1}^{\infty} \text{Cov}_\pi(g(X_0), g(X_i))$. $\gamma^2$ is referred to as the asymptotic variance of the chain, because it can also be computed via

$$\gamma^2 = \lim_{n \to \infty} n \text{Var}_\pi(\bar{g})$$  \hspace{1cm} (2-2)

The CLT in equation (2-1) was introduced in Chan and Geyer (1994). Unfortunately, there does not exist an elegant estimate of $\gamma^2$, even though there is a vast amount of research being done in this area. One of the more promising candidates to estimate $\gamma^2$ is the Batch Means estimator (see Jones et al. (2006), Flegal et al. (2008) and Flegal and Jones (2010)). However, it requires two levels of asymptotics in order to achieve consistency - both the batch size and the chain length have to tend to infinity, and the batch size is typically chosen by convention rather than by derivation. On the other hand, introducing regeneration times into a Markov chain allows us to break it up into independent and identically distributed (i.i.d) blocks, or tours. Applying the classical CLT to these i.i.d blocks yields a CLT with a naturally arising consistent estimate of the variance. Using the regeneration CLT removes any need for the chain to be in stationarity. We shall display the regeneration CLT later, after we have introduced the requisite notation.

### 2.2 Proving Geometric Ergodicity of a Markov Chain

In Jones and Hobert (2001), the authors describe one method of proving that $\Phi$ is geometrically ergodic. Note that convergence will be measured in total variation norm. Under the assumptions above, we have that, for all $x \in E$,

$$||P^n(x, \cdot) - \pi(\cdot)|| \downarrow 0 \text{ as } n \to \infty$$  \hspace{1cm} (2-3)

(A proof of this can be found in Rosenthal (2001)).

**Definition 2.2.1 (Geometric Ergodicity).** *If a chain $\Phi$ is geometrically ergodic, it means that there is a $0 < t < 1$ and a non-negative function $M$ such that for all $x$,

$$||P^n(x, \cdot) - \pi(\cdot)|| \leq M(x) t^n$$  \hspace{1cm} (2-4)*
The typical method of arriving at this inequality is via a drift and an associated minorization condition. A Markov chain can be minorized in many ways, but the most general method, which encompasses all the rest, is given by the definition from Nummelin (2004).

**Definition 2.2.2 (Minorization Condition).** A Markov transition kernel \( P(\cdot, \cdot) \) satisfies a minorization condition if there is a measurable function \( s \) and a finite non-negative measure \( \nu \) such that for all \( x \in E \) and \( A \in \mathcal{E} \),

\[
P(x, A) \geq s(x)\nu(A)
\]

(2–5)

A pair \((s, \nu)\) satisfying the above inequality is referred to as an atom for the transition kernel \( P \).

The function \( s \) is called a small function, and \( \nu \) is referred to as a small measure. Since the latter is finite, we can always normalize it to ensure that it is a probability measure, and absorb the normalizing constant \( \nu(E) \) into \( s(\cdot) \). Henceforth, we shall always assume that this is the case. Since \( \nu(E) = 1 \), it follows that \( 0 \leq s(x) \leq 1 \).

Minorization implies that we can split \( P \) into a mixture of two transition kernels, because

\[
P(x, dy) = s(x)\nu(dy) + (1 - s(x))P_r(x, dy)
\]

(2–6)

where \( P_r(x, dy) = (P(x, dy) - s(x)\nu(dy))/(1 - s(x)) \). Let us call \( P_r \) the residual kernel. Given \( X_n = x \), consider generating the chain in the following manner: Generate \( \delta_n \sim \text{Ber}(s(x)) \). If \( \delta_n = 1 \), generate \( X_{n+1} \sim \nu \). Otherwise, generate \( X_{n+1} \sim P_r(x, \cdot) \).

Several results can be shown immediately about this new sequence \((X_n, \delta_n)\) - the details are in Nummelin (2004), page 61. The first result is that \((X_n, \delta_n)\) is a Markov chain. Second, the marginal sequence \( \{X_n\} \) is a Markov chain that follows the law described by our original kernel \( P \). Lastly, whenever \( \delta_n = 1 \), we have a regeneration, meaning that the subsequent portion of the chain is independent of the past.
It might be difficult to generate a random variable from $P$, but this can be neatly sidestepped, since the Radon-Nikodym derivative of $\nu$ with respect to $P$ is just

$$\Pr(\delta_n = 1|X_n = x, X_{n+1} = y) = r(x, y) = \frac{s(x)\nu(dy)}{P(x, dy)} \quad (2-7)$$

Hence we can generate $\Phi$ according to $P$, and then fill in the “bell” variables $\delta_n$ later!

Suppose that $\Phi$ is started with $X_0 \sim \nu$. Let $0 < \tau_0 < \tau_1, ...$ be the random regeneration times; that is, $\tau_{t+1} = \min\{i > \tau_t : \delta_i = 1\}$. Also assume that $\Phi$ is run for a fixed number of tours, $R$. In other words, the chain was stopped on the $R$th occasion that $\delta_i = 1$.

Then if we let $N_t$ be the length of the $t$-th tour, i.e., $N_t = \tau_t - \tau_{t-1}$, and we define the sum of the observed values in the $t$-th tour to be

$$S_t = \sum_{j=\tau_{t-1}}^{\tau_t-1} g(X_j)$$

then the random vectors $(N_t, S_t)$ are independent and identically distributed for $t = 1, 2, ..., R$. If we know that $E\nu N_1^2$ and $E\nu S_1^2$ are finite, then we can apply the classical CLT to obtain

$$\sqrt{R}(\bar{g}_R - Eg) \rightsquigarrow N(0, \sigma^2) \quad (2-8)$$

where

$$\bar{g}_R = \frac{1}{\tau_R} \sum_{j=0}^{\tau_R-1} g(X_j) \quad \text{and} \quad \sigma^2 = \frac{E\nu[(S_1 - N_1 Eg)^2]}{(E\nu N_1)^2} \quad (2-9)$$

A consistent estimator of the variance is given by

$$\hat{\sigma}^2 = \frac{1}{R} \frac{\sum_{t=1}^{R} (S_t - N_t \bar{g}_R)^2}{\bar{N}^2} \quad \text{where} \quad \bar{N} = \frac{1}{R} \sum_{t=1}^{R} t = 1^R N_t \quad (2-10)$$

The full details of the above derivations can be found in Hobert et al. (2002), and observe that the result depends on $N_1$ and $S_1$ having finite second moments. The achievement of the authors was to show that geometric ergodicity and a $(2 + \epsilon)$-th moment on $g$ are sufficient for this.
In summary, here is how the two CLTs in equations (2–1) and (2–8) are linked. The authors in Chan and Geyer (1994) derived the following result.

Geometric ergodicity

\[
\begin{align*}
\text{Geometric ergodicity} & \quad + \quad (2 + \epsilon)^{-\text{th}} \text{-th moment on } g \\
& \Rightarrow \text{non-regeneration CLT (equation (2–1))}
\end{align*}
\]

The regeneration CLT only requires, along with a minorization condition, that \(E_{\nu}N_{1}^{2}\) and \(E_{\nu}S_{1}^{2}\) are finite:

\[
\begin{align*}
\text{Minorization condition (2–5)} & \quad + \\
E_{\nu}N_{1}^{2} \text{ and } E_{\nu}S_{1}^{2} \text{ finite}
& \Rightarrow \text{regeneration CLT (equation (2–8))}
\end{align*}
\]

What the authors showed in Hobert et al. (2002) is the following:

\[
\begin{align*}
\text{Geometric ergodicity} & \quad + \\
(2 + \epsilon)^{-\text{th}} \text{-th moment on } g
& \Rightarrow E_{\nu}N_{1}^{2} \text{ and } E_{\nu}S_{1}^{2} \text{ finite}
\end{align*}
\]

and hence the regeneration CLT holds. Formally, their theorem says the following:

**Theorem 2.2.1** (Hobert et al. (2002)). *Suppose assumptions (1) and (2) hold for \(\Phi\), and that \(\Phi\) can be minorized as in definition 2.2.2. If \(\Phi\) is geometrically ergodic and \(E_{\pi}|g|^{2+\epsilon} < \infty\) for some \(\epsilon > 0\), then \(E_{\nu}N_{1}^{2}\) and \(E_{\nu}S_{1}^{2}\) are both finite. It follows that the CLT in equation (2–8) is applicable.*

### 2.3 Regeneration in Parallel

If we can introduce regeneration into a Markov chain \(\Phi\), then we can run several of them in parallel, and essentially concatenate the chains. Consider a case where we have \(d\) processors, each running a Markov chain dictated by the same transition kernel
We shall index the \(d\) chains as follows:

\[
\Phi_i = \{X_{i,0}, X_{i,1}, \ldots\} \text{ for } i = 1, \ldots, d
\]

Assume that \(P(\cdot,\cdot)\) satisfies the minorization condition given in definition 2.2.2, and that all chains are initialized with the small measure

\[X_{i,0} \sim \nu \text{ independent for } i = 1, \ldots, d\]

Suppose we wish to collect a pre-determined number of tours, \(R\), and suppose we have \(d\) processors at our disposal. It is important to think about how we should spread these \(R\) tours over the \(d\) processors. If we simply tried to obtain the first \(R\) tours completed by the \(d\) processors and threw away any tours that were still running, we would be introducing a bias because we would be favouring shorter tours. A better idea would be to require each processor to generate \(\lfloor R/d \rfloor\) tours, and then require those that have finished to then work on one of the remaining tours. This would mean that the total number of tours collected is fixed and pre-determined, and that each of the \(d\) processors would run at least \(\lfloor R/d \rfloor\) tours. Some processors might run an extra tour, but no tour would be left unfinished with this approach and thus it would not favor shorter tours.

We set \(R_i\) to be the total number of tours in chain \(i\), and \(N_{i,j}\) to be the length of the \(j\)-th tour in chain \(i\), where \(i = 1, 2, \ldots, d\) and \(j = 1, 2, \ldots, R_i\). Finally, we redefine \(\bar{R}\) and \(\bar{N}\) to be \(R = \sum_{i=1}^{d} R_i\) and \(\bar{N} = (1/R) \sum_i \sum_j N_{i,j}\). The key point of this procedure is that the chains are initialized and run independently, which leads to the pairs \((N_{i,j}, S_{i,j})\) still being independent and identically distributed. As a result, the following Proposition holds.

**Proposition 2.3.1** (Regenerative CLT from parallel chains). Suppose \(\Phi_i\) satisfies the conditions in Theorem 2.2.1 for all \(i = i, \ldots, d\). Then if \(E_\pi |g|^{2+\epsilon} < \infty\) for some \(\epsilon > 0\), the following CLT holds.

\[
\sqrt{R}(\bar{g}_R - E_\pi g) \sim N(0, \sigma^2) \quad \text{as} \quad R \to \infty
\]
where

\[
\bar{g}_R = \frac{\sum_{i=1}^{d} \sum_{j=1}^{R_i} S_{i,j}}{\sum_{i=1}^{d} \sum_{j=1}^{R_i} N_{i,j}} \tag{2–12}
\]

and the consistent estimate of \(\sigma^2\) is given by

\[
\hat{\sigma}^2_g = \frac{1}{R} \sum_{j=1}^{d} \sum_{i=1}^{R_i} \left( S_{i,j} - N_{i,j} \bar{g}_R \right)^2 / N^2 \tag{2–13}
\]

Current methods of regenerative MCMC algorithms typically involve first running a long sequential chain starting from an arbitrary distribution. Bell variables are then filled in at each iteration of the sequential chain by generating Bernoulli random variables with success probabilities given by equation (2–7). This process splits the chain into independent chunks, yielding the \((N_i, S_i)\) pairs that we need in order to apply Theorem 2.2.1.

What we propose instead is generating the bell variables alongside the actual chain. Whenever the bell variable generated is a success, it indicates the completion of a tour. Theoretically, we can then begin a new tour by restarting with a draw from \(\nu\). With this approach, we can run each tour on a separate processor, and form \(\bar{g}_R\) and \(\hat{\sigma}^2_g\), which are defined in Proposition 2.3.1, from the resulting independent tours. Although \(\bar{g}_R\) is no longer the ergodic mean from a long non-regenerative chain, Proposition 2.3.1 proves that the theory is sound. In practice, we shall not throw away the currently generated state and restart from \(\nu\) whenever we encounter a regeneration. Instead, we shall continue the chain with the currently generated state. This would have a twofold benefit. Firstly, it would not waste random variables already generated. Secondly, it would allow the final point estimator to be based on a chain that is close to stationarity, and thus would intuitively reduce the ratio point estimator effect. For more on the ratio estimator, see Section 3.3.

### 2.4 Minorization of Common Markov Chain Monte Carlo Samplers

In Mykland et al. (1995), the authors described quite general methods of minorizing Metropolis-Hastings (M-H) and Gibbs samplers. Suppose our target distribution \(\pi\)
has a density with respect to $\mu$. Let us denote the density by $\pi$ as well, and denote our proposal distribution by $Q(x, dy) = q(x, y)\mu(dy)$. Denoting the usual Metropolis acceptance ratio by $\alpha(x, y)$ and the M-H kernel by $P$, the authors in Mykland et al. (1995) show how to minorize $P$ if the following 2 conditions hold:

1. An atom $(s_q, \nu_q)$ that minorizes $Q$ (see definition 2.2.2) can be found. Formally, we have that $q(x, y)\mu(dy) \geq s_q(x)\nu_q(dy)$, and

2. There exists a positive function $h$ such that

$$h(x)q(x, y) = h(y)q(y, x)$$

(2–14)

When these two conditions are both met, then $P$ can be minorized in the following manner with some $c > 0$.

$$P(x, dy) \geq s_q(x)\min\left\{ \frac{ch(x)}{\pi(x)}, 1 \right\}\nu_q(dy)\min\left\{ \frac{\pi(y)}{ch(y)}, 1 \right\}$$

(2–15)

Condition (2) above holds, in particular, when the proposal is an independent jump, or when it is symmetric. In the case where $q$ is symmetric, i.e., $q(x, y) = q(y, x)$, any constant can serve as $h$. If we then pick $\bar{x} \in E$ and a compact set $D \in \mathcal{E}$, we can split $Q$ with

$$\nu_q(dy) = \frac{q(\bar{x}, y)I_D(y)}{\int_D q(\bar{x}, u)\mu(du)} \quad \text{and} \quad s_q(x) = \inf_{z \in D} \frac{q(x, z)}{q(\bar{x}, z)} \times \int_D q(\bar{x}, u)\mu(du)$$

Since $h$ is just a constant in this case, we have that

$$\nu'(dy) = \nu_q(dy)\min\{\pi(y)/c, 1\}$$

If $E$ is $\mathcal{R}^n$, then $\pi$ will take on a minimum value on $D$. Hence by choosing a small enough $c < \pi(y)$ for all $y \in D$, we can ensure that $\nu' = \nu_q$. Then we can start the chain with a regeneration by sampling from $\nu_q$. With this “distinguished point” and “distinguished set” technique, the complete minorization that is being applied to $P$ when
$q$ is symmetric is

$$P(x, dy) \geq \inf_{z \in D} \frac{q(x, z)}{q(x, z)} \min \left\{ \frac{c}{\pi(x)}, 1 \right\} \frac{q(\tilde{x}, y)l_D(y)\mu(dy)}{\nu(dy)}$$  \hspace{1cm} (2–16)$$

This technique of working with a distinguished point and set is very amenable to minorizing a Gibbs sampler, as we shall see later in the section on linear models.

Mykland et al. (1995) point out that we should try to find a pair $(s, \nu)$ such that $s(x)\nu(dy)$ is large on average, as that would mean the regeneration probabilities would be higher, leading to more tours.

### 2.5 Finite State Samplers with Mixture Candidates

In section 2.4, we reviewed the main techniques introduced in Mykland et al. (1995) for splitting a large class of Metropolis-Hastings Markov chains. The methods rely on there being a positive function $h$ satisfying the condition in (2–14). These approaches can be extended to mixture kernels, as long as one of the component kernels satisfies the condition. Suppose that we have a kernel $P = p_1P_1 + (1 - p_1)P_2$, and $P_1$ is a Metropolis kernel that can be minorized. Then $P$ can be minorized in the following manner:

$$P(x, dy) \geq p_1s(x)\nu(dy)$$  \hspace{1cm} (2–17)$$

where $(s, \nu)$ is the atom for kernel $P_1$.

Sometimes, however, a user might want to use a candidate that is a mixture rather than a mixture of kernels. One of the desirable properties of using a mixture candidate over a mixture kernel is captured in the following result in Proposition 2.5.1. It is identical to Proposition 5 in Tierney (1998). Here we provide a proof for finite state space chains.

We need to introduce a couple of ideas first. Recall the concept of the asymptotic variance of a Markov chain introduced in equation (2–2). Note that it is specific to the function $g$ whose integral we wish to estimate using our Markov chain. The following definition can be found in Mira (2001).
**Definition 2.5.1** (Uniform efficiency). Let $P$ and $Q$ be two transition matrices for the same stationary distribution $\pi$. For a function $g$ such that $E_\pi |g|^2 < \infty$, let $\gamma^2_{P,g}$ and $\gamma^2_{Q,g}$ be the asymptotic variances when using $P$ and $Q$ respectively. We say that $P$ is at least as uniformly efficient as $Q$ if $\gamma^2_{P,g} \leq \gamma^2_{Q,g}$ for all $g$ such that $E_\pi |g|^2 < \infty$.

**Theorem 2.5.1** (Peskun Ordering, Peskun (1973)). Let $P$ and $Q$ be transition matrices for the same stationary distribution $\pi$. Then $P$ dominates $Q$ in the Peskun sense, $Q \ll P$, if each of the off-diagonal elements of $P$ is greater than or equal to the corresponding off-diagonal element of $Q$. If $P$ dominates $Q$ in the Peskun sense, then $P$ is at least as uniformly efficient as $Q$, as defined above in Definition 2.5.1.

Consider two finite-state Metropolis-Hastings chains, with the same stationary distribution $\pi$. Suppose that we have a finite sequence of $m$ candidate distributions, given by $q_1(x, y), \ldots, q_m(x, y)$, and a set of weights $w_1, \ldots, w_m$ such that $\sum w_i = 1$. One of the chains uses a mixture of $m$ kernels, with $q_i$ being the candidate for kernel $i$. The other uses a single kernel, with a mixture of the $q_i$'s a candidate. The off-diagonals of the transition kernel for the mixture candidate, $P_1$, are given by

$$P_1(x, y) = \min \left\{ \frac{\pi(y)}{\pi(x)} \sum_{i=1}^m w_i q_i(y, x), \sum_{i=1}^m w_i q_i(x, y) \right\} \quad (2-18)$$

while those for the mixture kernel are given by

$$P_2(x, y) = \sum_{i=1}^m w_i \min \left\{ \frac{\pi(y)}{\pi(x)} q_i(y, x), q_i(x, y) \right\} \quad (2-19)$$

**Proposition 2.5.1.** With $P_1$ and $P_2$ set up as above, we have that $P_2 \ll P_1$. In other words, $P_1$ dominates $P_2$ in the Peskun sense. If in addition, we have that, for two states $x$ and $y$ and for $i, j \in \{1, 2, \ldots, m\}$,

$$\pi(y) q_i(y, x) > \pi(x) q_i(x, y) \quad (2-20)$$

$$\pi(y) q_j(y, x) < \pi(x) q_j(x, y) \quad (2-21)$$

then the ordering is strict.
Proof. Refer back to equation (2–19), which applies when \( x \neq y \). For all \( i \in \{1, 2, \ldots, m\} \), we have that

\[
\min \left\{ \frac{\pi(y)}{\pi(x)} q_i(y, x), q_i(x, y) \right\} \leq \frac{\pi(y)}{\pi(x)} q_i(y, x) \quad \text{and} \quad \min \left\{ \frac{\pi(y)}{\pi(x)} q_i(y, x), q_i(x, y) \right\} \leq q_i(x, y)
\]

It follows immediately that the integral of the above LHS terms with respect to the weights \( w_i \) will also obey the inequalities. Thus

\[
P_2(x, y) = \sum_{i=1}^{m} w_i \min \left\{ \frac{\pi(y)}{\pi(x)} q_i(y, x), q_i(x, y) \right\} \leq \sum_{i=1}^{m} w_i \frac{\pi(y)}{\pi(x)} q_i(y, x) \quad \text{and} \quad P_2(x, y) \leq \sum_{i=1}^{m} w_i q_i(x, y)
\]

\[
\Rightarrow P_2(x, y) \leq \min \left\{ \sum_{i=1}^{m} w_i \frac{\pi(y)}{\pi(x)} q_i(y, x), \sum_{i=1}^{m} w_i q_i(x, y) \right\} = P_1(x, y)
\]

This proves that the off-diagonal terms of the transition matrix of \( P_1 \) are at least as large as those of \( P_2 \). We can then apply Theorem 3.1 of Mira and Geyer (1999) to conclude that \( P_1 \) is more efficient than \( P_2 \).

If in addition, the inequalities in the statement of the Proposition hold, then it follows from the first one that

\[
q_i(x, y) = \min \left\{ \frac{\pi(y)}{\pi(x)} q_i(y, x), q_i(x, y) \right\} < \frac{\pi(y)}{\pi(x)} q_i(y, x)
\]

\[
\Rightarrow \sum_{k=1}^{m} w_k \min \left\{ \frac{\pi(y)}{\pi(x)} q_k(y, x), q_k(x, y) \right\} < \sum_{k=1}^{m} w_k \frac{\pi(y)}{\pi(x)} q_k(y, x)
\]

From the second inequality, it follows that

\[
\frac{\pi(y)}{\pi(x)} q_j(y, x) = \min \left\{ \frac{\pi(y)}{\pi(x)} q_j(y, x), q_j(x, y) \right\} < q_j(y, x)
\]

\[
\Rightarrow \sum_{k=1}^{m} w_k \min \left\{ \frac{\pi(y)}{\pi(x)} q_k(y, x), q_k(x, y) \right\} < \sum_{k=1}^{m} w_k q_k(y, x)
\]

Combining the two strict inequalities, we can arrive at the strict Peskun ordering. \( \square \)
The implication of the Peskun ordering is that the variance of the limiting distribution in the non-regenerative CLT is smaller for the mixture candidate than the mixture kernel. In other words, the $\gamma^2$ from equation (2–1) will be smaller if we utilise a mixture candidate instead of a mixture kernel.

2.6 Summary

In this chapter we have provided a review of the necessary theory for regenerating Markov chains in parallel. We have also provided a result that shows that using a mixture candidate is better than a mixture kernel in terms of asymptotic variance. However, when deciding between the two in practical terms, one also has to consider which is easier to compute. In some cases, it might not even be possible to compute the mixture density. This would lead us to using the mixture kernel. In other cases, it might not be possible to obtain a minorization that provides sufficiently short tour lengths. Minorizing the mixture kernel is more straightforward, and usually provides shorter tours.
CHAPTER 3
IMPLEMENTATION ASPECTS OF PARALLEL REGENERATION

3.1 Achievable Speed-up from Parallelization

In Amdahl (1967), a method to analyse the speed-up of an algorithm was introduced. Suppose that an algorithm consists of $A$ float-point operations. Assume that a fraction $f$ of those operations can be sped up to run at $V_1$ float-point operations per second (flops), and the remaining operations have to remain running at $V_2$ flops, with $V_2 < V_1$. Then the new execution time of the algorithm is

$$t_{\text{fast}} = f \frac{A}{V_1} + (1 - f) \frac{A}{V_2} > \frac{(1 - f)A}{V_2}$$

Compared with $t_{\text{slow}} = \frac{A}{V_2}$, the speed-up afforded by the acceleration is

$$s = \frac{t_{\text{slow}}}{t_{\text{fast}}} < \frac{A/V_2}{(1 - f)(A/V_2)} = \frac{1}{1 - f}$$

(3–1)

The above inequality is known as Amdahl’s Law, and can be used to derive a bound on the maximum speed up of an algorithm. It can be extended to parallelisation of a serial algorithm in an analogous way. Suppose that a serial algorithm takes $t_1$ seconds to execute on a particular processor. Assume that a fraction $f$ of the algorithm can be executed in parallel on $p$ processors ideally. In other words, this fraction $f$ of the algorithm can be executed on $p$ processors in exactly $ft_1/p$ seconds. Though this is unrealistic, it greatly simplifies the analysis. The speed-up can again be bounded by the same quantity as above:

$$s = \frac{t_1}{ft_1/p + (1 - f)t_1} = \frac{p}{r + (1 - f)p} < \frac{1}{1 - f}$$

Now consider the serial algorithm that we wish to parallelise:

begin program

read input; initialization;

for (i=1, . . . , R)


It is not clear what fraction $f$ of our code can be parallelised. Also, as we increase the number of available processors, we would probably want to increase the number of tours we wish to run. In situations like this, it would be better to consider an alternative view. In Gustafson (1988), the author considered the inverse question to Amdahl: Given a parallel algorithm, how much faster would it be when compared to running it on a serial processor? Put in this way, our algorithm becomes easier to analyse, but it is still complicated by the fact the number of operations in each tour is a random quantity.

Consider the idealized version of our parallel algorithm to run $R$ tours. Suppose we have $R + 1$ identical processors at our disposal, each of which runs at a speed of $V$ flops. The $R$ processors that actually run the tours are usually referred to as the slaves, while the processor controlling them is known as the master. Suppose also that there are no latencies in communicating with the slaves. The following code is run:

Code on master:
begin program
read input; initialization;
start slaves (i=1, . . . ,R);
receive results (i=1, . . . ,R);
evaluate results;
end program

Code on slaves:
begin program
receive parameters from master;
generate 1 tour;
send result to master;
end program

Let $A_i$ be the random variable representing the number of float-point operations required to complete tour $i$ on processor $i$. Recall that $i$ runs from 1 to $R$ and crucially, that the $A_i$ are i.i.d. Then the following proposition summarises the expected speed up we can expect to attain with our parallel algorithm.

**Proposition 3.1.1** (Speed-up from regeneration in parallel). *If we denote $A(R)$ as the maximal order statistic from an i.i.d sample of size $R$, then the expected speed-up is bounded above and below by*

$$R \times \sup_{\alpha \in [0,1]} \alpha \Pr(A_1 \geq \alpha A(R)) \leq \mathbb{E}[S] \leq R \quad (3-2)$$

*where the random speed up $S$ is given by*

$$S = \frac{\sum_{i=1}^{R} A_i}{A(R)} \quad (3-3)$$

**Proof.** The upper bound is direct: Since $A_i \leq A(R)$ for all $i$, it follows that

$$S = \frac{\sum_{i=1}^{R} A_i}{A(R)} \leq \frac{\sum_{i=1}^{R} A(R)}{A(R)} = R \quad \Rightarrow \quad \mathbb{E}[S] \leq R$$

For the lower bound, first note that the bivariate random vectors $(A_i, A(R))$ are identically distributed. This can be seen from their joint cdfs. For $b \geq a$ and for any $i \in \{1, 2, \ldots, R\}$

$$\Pr(A_1 \leq a, A(R) \leq b) = \Pr(A_1 \leq a, \max\{A_2, \ldots, A_R\} \leq b)$$

$$= \Pr(A_1 \leq a) \Pr(\max\{A_2, \ldots, A_R\} \leq b)$$

$$= \Pr(A_i \leq a) \Pr(\max\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_R\} \leq b)$$

$$= \Pr(A_i \leq a, \max\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_R\} \leq b)$$

$$= \Pr(A_i \leq a, A(R) \leq b)$$
For $b < a$, we have that

$$
\Pr(A_1 \leq a, A(R) \leq b) = \Pr(b < A_1 \leq a, A(R) \leq b) + \Pr(A_1 \leq b, A(R) \leq b)
$$

$$
= 0 + \Pr(A_1 \leq b, A(R) \leq b)
$$

$$
= \Pr(A_i \leq b, A(R) \leq b)
$$

where the last equality follows from the $b \geq a$ case directly above. Going back to the lower bound, we have

$$
\mathbb{E}[S] = \mathbb{E} \left[ \frac{\sum_{i=1}^{R} A_i}{A(R)} \right]
$$

$$
= \sum_{i=1}^{R} \mathbb{E} \left[ \frac{A_i}{A(R)} \right]
$$

$$
= R \times \mathbb{E} \left[ \frac{A_1}{A(R)} \right] \quad \text{(by above)}
$$

$$
\geq R \times \alpha \Pr \left[ \frac{A_1}{A(R)} \geq \alpha \right] \quad \text{for all } \alpha \quad \text{(by Markov’s inequality)}
$$

$$
\therefore \mathbb{E}[S] \geq R \sup_{\alpha \in [0,1]} \Pr \left[ \frac{A_1}{A(R)} \geq \alpha \right]
$$

We must point out that the above result is only a lower bound that in itself might be very loose, but our intention in introducing it is to point out that the speed up we attain from using this algorithm depends on the distribution of the tour lengths.

For example, consider the extreme case where the tours are degenerate random variables. In that situation, the lower bound will be equal to the upper bound, $R$, and we get the maximum possible speed-up. On the other hand, suppose we know that the tour lengths are Poisson random variables with $\lambda = 50$, and we wish to run $R = 50,000$ tours. Figure 3-1 indicates that we can expect at least a 40% speed up.

### 3.2 Completing Unfinished Tours

A consequence of using regenerations as the basis of analysis of a Markov chain is that we will not know exactly how long we shall have to run the chain for. The reason for
this is that the tour lengths would be random. This problem is compounded if we perform
computations on a cluster, as there would be several independent chains, and we would
not know when to stop each of them. In such a case, the experimenter is faced with 2
options:

1. Specify the total number of tours, \( R \), that the subsequent analysis should be based
   on, and then stop the chain when \( R \) tours are completed.

2. Specify the total number of iterations that the chain should be run for, and then
   stop the chain when this is achieved. The subsequent analysis would then be
   based on the tours completed by that time point.

As pointed out earlier, the downside with option 1 is that it requires a random
amount of total computing time. Although this problem does not arise with option 2,
other issues surface. First of all, it is possible that no tours might be completed by the
end of the run. Secondly, as investigated by Meketon and Heidelberger (1982), utilising
only the completed tours to estimate the mean involves a bias of the order \( 1/t \), where
\( t \) is the length of the Markov chain that is run. The authors showed that waiting for that
final tour to end would reduce the order of the bias to \( 1/t^2 \). They were able to show
that for small values of \( t \), the bias reduction is appreciable. The reason for that bias
reduction is that that final tour is atypical. This is the so-called inspection paradox or
waiting-time paradox. It refers to the fact that once we pre-specify a time point, the tour
that contains that time point will be larger than a normal tour. The order of the bias of the
estimator can be reduced simply by waiting for the tour that contains the pre-specified
time to complete. Details about this issue can be found in Feller (1971).

Another issue with option 2 is that the number of tours, \( R \), is now random. It is not
straightforward to see if Theorem 2.2.1 and Proposition 2.3.1 still hold, although they
should do so as \( R \to \infty \).

Thus option 2 may not be the preferred one. However, when running a regenerative
Markov chain in parallel on a centrally administered cluster, a user typically has to
specify an estimate on the wall-time that his program will take to run. This estimate is
a hard upper bound. When the scheduler detects that a particular program has used
up its allotment of computing time, it will kill the program and release the memory and
compute nodes that had been assigned to it. Hence even if a user intended to wait for
a pre-specified number of tours to complete, he would still have to specify an upper
bound on the compute time, and he would have to deal with the situation of ending up
with fewer tours than he would have liked. More crucially, he would be left with several
incomplete tours, one on each processor used.

In this section, we consider the possibility of utilising the completed tours to
estimate what would have been observed had they been allowed to run to completion.
Before we proceed, we introduce notations and terms from renewal theory, which
provides one possible way to “complete” the tours.

3.2.1 Renewal Theory

Let \( \{N_i, i = 1, 2, \ldots \} \) be a sequence of positive, integer-valued, independent random
variables with a common distribution \( F \), with \( \mathbb{E}[N_1] < \infty \). Let \( T_0 = 0, T_n = \sum_{i=1}^{n} N_i \) and \( M(t) = \sup\{n : T_n \leq t\} \)

The process \( \{M(t), t \geq 0\} \) is a Renewal Process. The times \( N_1, N_2, \ldots \) are referred to
as interarrival times and the \( T_i \) are termed renewal times. Suppose also that a random
reward is earned gradually during each renewal cycle. Let the reward earned during the
\( n \)-th renewal (i.e. over \( N_n \)) be \( S_n \). This reward can, and usually will, depend on \( N_n \) - the
length of the renewal interval. However, we assume that \( (S_n, N_n) \) are i.i.d.

The link between renewal theory and our regenerative chains is clear, especially
since we use the same notation. \( N_i \) are the tour lengths, and \( S_i \) is the sum of the random
variables generated over that tour. There is a rich theory that has been developed for
renewal theory, which relies essentially on the process restarting itself probabilistically
at each renewal point. Renewal theory was first developed by Feller (1971), but a
comprehensive review of the theory can be found in Smith (1958), while an accessible mid-level introduction can be found in Ross (1992).

A number of theorems exist pertaining to the asymptotic properties of the arrival rate. Another very useful result from renewal theory is a technique for solving “renewal-type” equations. However, we shall not need to resort to those results. We only need the independence and stationarity of the renewal process for our results.

3.2.2 Estimating Residual Life and Residual Reward

Let $L(t)$ be the time from $t$ until the next renewal, and $Z(t)$ the time from $t$ since the last renewal. $L(t)$ is called the residual life at $t$, $Z(t)$ is termed the age at $t$. Formally,

\[ L(t) = T_{M(t)+1} - t \]  
\[ Z(t) = t - T_{M(t)} \]  

What happens when we stop a Markov chain at iteration (time) $t$, is that we might be in the middle of a tour. In this situation, we would know that say, $Z(t) = s$, and we would be interested in estimating $L(t)$. We have the following intuitive result:

**Proposition 3.2.1.**

\[ \mathbb{E}[L(t) | Z(t) = s] = \mathbb{E}[N_1 - s | N_1 > s] \]  

**Proof.**

\[
\mathbb{E}[L(t) | Z(t) = s] = \mathbb{E}[T_{M(t)+1} - t | t - T_{M(t)} = s] \\
= \sum_{r=1}^{\infty} r \Pr[T_{M(t)+1} - t = r | t - T_{M(t)} = s] \\
= \sum_{r=1}^{\infty} r \sum_{n=0}^{\infty} \Pr[T_{M(t)+1} - t = r | M(t) = n, T_{M(t)} = t - s] \\
\times \Pr[M(t) = n | T_{M(t)} = t - s] \\
= \sum_{r=1}^{\infty} r \sum_{n=0}^{\infty} \Pr[T_{M(t)+1} - t = r | M(t) = n, T_n = t - s] \\
\times \Pr[M(t) = n | T_{M(t)} = t - s]
\]
(since we know the process restarts at time $T_n$)

$$\begin{align*}
&= \sum_{r=1}^{\infty} \sum_{n=0}^{\infty} \Pr[N_{n+1} = r + s | N_{n+1} > s] \Pr[M(t) = n | T_M(t) = t - s] \\
&= \sum_{r=1}^{\infty} r \Pr[N_1 = r + s | N_1 > s] \sum_{n=0}^{\infty} \Pr[M(t) = n | T_M(t) = t - s] \\
&= \sum_{r=1}^{\infty} r \Pr[N_1 - s = r | N_1 > s] = \mathbb{E}[N_1 - s | N_1 > s]
\end{align*}$$

Note that the equality just before the final one follows because the inter-arrival times are independent and identically distributed.

Hence we can find the expected length of that incomplete tour:

$$\mathbb{E}[L(t) + Z(t) | Z(t) = s] = s + \mathbb{E}[L(t) | Z(t) = s]$$

$$= s + \mathbb{E}[N_1 - s | N_1 > s]$$

$$= \mathbb{E}[N_1 | N_1 > s]$$

Thus, knowing that there were no renewals since $s$ iterations ago means we do not have to concern ourselves with the inspection paradox and can estimate the length with a simple conditional distribution. We can make a very similar computation to estimate the reward from the incomplete tour. However, we need to complicate the notation a little further. Recall the specification that the reward is gained gradually. Let $S'_{s,n}$ be the accumulated reward up until iteration $s$ within the $n$-th tour. Also note that since each tour is independent and identically distributed (we are essentially re-starting the Markov chain from the same initial measure $\nu$ that is defined in equation (2–6)), we can see that for a fixed $s$, the $S'_{s,i}$ are also identically distributed. In other words, the $S'_{s,j}$ are the sum of $s$ iterations of a Markov chain with the same transition kernel and initial distribution. Since the $S_i$ random variable is absolutely continuous, we have to specify a set that it falls into. Let this set be denoted as $D$.  

35
Proposition 3.2.2.

$$\mathbb{E}[S_{M(t)+1}|Z(t) = s, S'_{s,M(t)+1} \in D] = \mathbb{E}[S_1|S'_{1,s} \in D, N_1 > s]$$  \hspace{1cm} (3–7)

Proof. Let us derive an expression for the distribution whose expectation we wish to obtain. For any Borel set $A$,

$$\Pr[S_{M(t)+1} \in A|Z(t) = s, S'_{s,M(t)+1} \in D]$$

$$= \sum_{n=0}^{\infty} \Pr[S_{M(t)+1} \in A|Z(t) = s, S'_{s,M(t)+1} \in D, M(t) = n]$$

$$\times \Pr[M(t) = n|S'_{s,M(t)+1} \in D, Z(t) = s]$$

$$= \sum_{n=0}^{\infty} \Pr[S_{M(t)+1} \in A|Z(t) = s, S'_{s,M(t)+1} \in D, M(t) = n]$$

$$\times \Pr[M(t) = n|S'_{s,M(t)+1} \in D, T_M(t) = t - s]$$

Consider the first term in the summation expression above.

$$\Pr[S_{M(t)+1} \in A|Z(t) = s, S'_{s,M(t)+1} \in D, M(t) = n]$$

$$= \frac{\Pr[S_{M(t)+1} \in A, S'_{s,M(t)+1} \in D|Z(t) = s, M(t) = n]}{\Pr[S'_{s,M(t)+1} \in D|Z(t) = s, M(t) = n]}$$

Having observed that the $n$-th renewal occurred at time $(t - s)$, and due to the independence of tours, this is exactly equal to

$$\frac{\Pr[S_{n+1} \in A, S'_{s,n+1} \in D|N_{n+1} > s]}{\Pr[S'_{s,n+1} \in D|N_{n+1} > s]}$$

Now due to the fact that tours are identically distributed, this is in turn equal to

$$\frac{\Pr[S_1 \in A, S'_{s,1} \in D|N_1 > s]}{\Pr[S'_{s,1} \in D|N_1 > s]}$$
Then returning to the summation term above, we can once again write

\[
\Pr[S_{M(t)+1} \in A | Z(t) = s, S'_{s,M(t)+1} \in D] = \Pr[S_1 \in A | N_1 > s, S'_{1,1} \in D] \sum_{n=0}^{\infty} \Pr[M(t) = n | S'_{s,M(t)+1} \in D, T_{M(t)} = t - s]
\]

\[
= \Pr[S_1 \in A | N_1 > s, S'_{s,1} \in D]
\]

This proves the proposition. \(\square\)

One way to use the results derived in Propositions 3.2.1 and 3.2.2 is to utilise the available completed tours to estimate the reward and length of the final tours. For example, suppose that after running \(R\) complete tours, the \((R + 1)\)-th tour had to be stopped after \(s\) iterations, and that out of the \(R\) completed tours, \(R_N\) of them were longer than \(s\) iterations. Then one straightforward method of estimating its length would be to use

\[
\hat{N}_{R+1} = \frac{1}{R_N} \sum_{i=1}^{R} N_i \mathbb{1}[N_i > s]
\]

If the observed partial sum for that final tour is \(S'_{R+1,s}\), then we need to fix an interval \(D\), and use the information from the subset of observed tours whose partial reward after \(s\) iterations falls into \(D\) after \(s\) iterations.

\[
\hat{S}_{R+1} = \frac{1}{R_S} \sum_{i=1}^{R} S_i \mathbb{1}[S'_{i,s} \in D, N_i > s]
\]

where \(R_S\) is now the number of completed tours that fulfilled the criterion in the above indicator function. Of course, choosing this \(D\) is a tradeoff - one that is too big would not use the information from the incomplete tour as efficiently. For example, setting \(D\) to be whole real line would completely ignore the exact partial sum observed. It would, however, result in a larger \(R_S\) and thus more tours to base an estimate on. On the other hand, if we were to set \(D\) to be too narrow, the number of tours \(R_S\) might be too small, resulting in a more imprecise estimate of \(S_{R+1}\).
A drawback of this simple approach is that if none of the observed tours were longer than $s$ iterations, then it cannot be used. This is however, very unlikely to happen, since the number of completed tours is typically in the hundreds of thousands.

### 3.3 Possible Diagnostics

When using regeneration, we essentially observe a sequence of bivariate random variables $(S_i, N_i)$ and we form our estimator out of that sequence. Notice that the point estimate that arises is a ratio of random variables:

$$\bar{g}_R = \frac{1}{R} \sum_{j=0}^{R-1} g(x_j) = \frac{\sum_{t=1}^{R} S_t}{\sum_{t=1}^{R} N_t}$$

Since $\bar{S} \to E_{\nu}(N_1)E_{\pi}g$ a.s., we know that $\bar{g}_R$ is consistent for the target parameter. However, because it holds in general that

$$E\left(\frac{\sum_{t=1}^{R} S_t}{\sum_{t=1}^{R} N_t}\right) \neq \frac{E\left(\sum_{t=1}^{R} S_t\right)}{E\left(\sum_{t=1}^{R} N_t\right)} = E_{\pi}g$$

we can see that the finite-sample estimator that we form will be biased. The asymptotic variance in the regenerative CLT (see equation (2–9)) can be derived using the delta method, starting with asymptotic normality of the bivariate estimator $(\bar{S}, \bar{N})$.

However, that derivation depends on the approximation of $\bar{S}/\bar{N} - E_{\pi}g$ by $(\bar{S} - (E_{\pi}g)\bar{N})/E_{\nu}N_1$, which is based on a Taylor series expansion. The expected absolute value of that Taylor series error term is given by

$$E\left|\left(\frac{\bar{S}}{\bar{N}} - \frac{E_{\nu}S_1}{E_{\nu}N_1}\right) - \frac{\bar{S} - (E_{\nu}S_1/E_{\nu}N_1)\bar{N}}{E_{\nu}N_1}\right| = \frac{1}{E_{\nu}N_1}E\left|\left(\frac{\bar{S}}{\bar{N}} - \frac{E_{\nu}S_1}{E_{\nu}N_1}\right)(\bar{N} - E_{\nu}N_1)\right|$$

$$\leq \left[E\left(\frac{\bar{S}}{\bar{N}} - \frac{E_{\nu}S_1}{E_{\nu}N_1}\right)^2\right]^{1/2}\left[\frac{E(\bar{N} - E_{\nu}N_1)^2}{E_{\nu}N_1^2}\right]^{1/2}$$

$$= (\text{MSE}(\bar{g}_R, E_{\pi}g)\text{CV}(\bar{N}))^{1/2}$$

where $\text{MSE}$ stands for Mean Squared Error and $\text{CV}$ stands for Coefficient of Variation.

The MSE of a statistic $T$ with respect to a parameter $\tau$ is defined to be $\text{MSE}(T, \tau) = \mathbb{E}(T - \tau)^2$, while the $\text{CV}$ of a random variable is defined to be $\text{CV}(T) = \text{Var}(T)/(\mathbb{E}[T])^2$. 

38
The final product term in the above display is exactly what the authors put forward in p.234 of Mykland et al. (1995). This is the reason they suggest that when running regenerative simulations, the coefficient of variation of $\bar{N}$ should be monitored. Due to this, Bratley et al. (1987) and Ripley (1987) both explain that regenerative simulation works best when the tours we observe are of almost equal length (i.e., $\bar{N}$ has little variability).

In order to fix the possible bias problem highlighted earlier, Bratley et al. (1987) and Ripley (1987) suggest jacknifing the estimates of the mean and the variance. However, we should be wary when using the jacknifed estimate, because, as pointed out in Section 2.4 of Shao and Tu (1995), although asymptotic normality is unaffected, jackknifing does increase the MSE of the finite-sample estimator.

However, the Normal approximation dictates that the bias will reduce to zero as we collect more and more tours. The purpose of this section is to consider the use of diagnostics for assessing if we have sufficient tours.

In Mykland et al. (1995), the authors mention two ways of monitoring when the Normal approximation is good. Here, we introduce them and expound on them a little. As always, let $N_i$ denote the individual tour lengths, $T_n = \sum_{i=1}^{n} N_i$ and suppose that $R$ tours were collected.

The first statistic they suggest tracking is the $\hat{CV}(\bar{N})$, the estimated coefficient of variation of $\bar{N}$. If this is greater than, say, 1%, then the workings in the beginning of this section suggests that the error term in the delta method derivation of the Central Limit Theorem could be large. They suggest collecting more tours in these situations.

The second diagnostic they suggested comes from viewing the regenerative method in the context of renewal theory. On page 234 of Mykland et al. (1995), the authors mention that the proportion of renewals that fall in a fraction $\alpha$ of the observation period converges to $\alpha$. Hence they put forward the idea of a scaled regeneration
quantile plot (SRQ): $T_i / T_R$ against $i / R$. The motivation for this diagnostic is rooted in the following quote, lifted directly from their paper:

“the proportion of renewals that fall in a fraction $\alpha$ of the observation period converges to $\alpha$.”

This is similar to the exact result for Poisson renewal processes, which says that if we know there are $R$ renewals in an interval $(0, t]$, then those $R$ renewals are independently and uniformly distributed over the interval $(0, t]$. A proof of this result can be found in Theorem 2.3 of Ross (1992). Now consider plotting a quantile-quantile plot for the observed regeneration times, to see if they are similar to a uniform distribution. The empirical quantile for $1/R$ would be $T_1$. The theoretical quantile, based on the uniform distribution on $(0, T_R]$, would be $1/R \times T_R$. Thus the first point on the plot would be $(T_R/R, T_1)$. The second point would be $(2T_R/R, T_2)$, and so on. Scaling this plot by dividing both axes by $T_R$, we get the one that the authors suggested: $T_i / T_R$ against $i / R$. If the plot of these points were to fall close to the $y = x$ line, we would want to believe that sufficient tours have been collected.

However, in our simulations, the above diagnostics were not particularly helpful. Neither the coefficient of variation nor the scaled regenerated quantile plots were particularly helpful in determining if more tours should be run (see Section 4.1 and Figures 4-1 and 4-2). However we have included both in our package, which we describe next.

### 3.4 Summary

In this chapter we have provided a couple of results. The first allows us to obtain some idea of the benefit we might gain from parallelizing a specific Markov chain we have. The result in Proposition 3.1.1 provides a lower bound for this. After an initial run of our chain to estimate tour lengths, we could use a plot similar to the one in Figure 3-1 to visualise the possible gain.
In Propositions 3.2.1 and 3.2.2, we have provided a method for a user to utilise the complete tours in order to estimate the missing ones and shown that we need not be concerned with the inspection paradox when we do so. This area has scope for further research (see Chapter 6).

Figure 3-1. The figure above allows us to visualise the lower bound from Proposition 3.1.1. The level-plot is created using the equation $z = xy$, where $x, y \in [0, 1]$. The overlayed curve is $\Pr(A_1 \geq \alpha A(R))$ against $\alpha$ for a Poisson random variable with $\lambda = 50$ and taking $R = 50,000$ tours. 100 samples of 50,000 tours were taken in order to estimate the curve. The closer the curve is to the top right hand corner, the greater the lower bound of the speed up will be. Here, we can estimate the lower bound to be slightly more than $40\% \times R$. In this small example, we can in fact compute the true speed-up for each of the 100 realisations of the 50,000 tours. Then we can compute the true value of $\mathbb{E}[S]$ and compare it to the value provided by the lower bound. When we do so, we obtain $\mathbb{E}[S] = 60\% \times R$. 
CHAPTER 4
R PACKAGE FOR PARALLEL REGENERATIVE MARKOV CHAINS

In this section, we present the models included with our R package McParre. The functions in the package, and instructions on how to use it are documented in detail in the Appendix. They are the oneway model with proper priors from Hobert and Geyer (1998), the linear mixed model from Hobert et al. (2006). The package also includes parallel regeneration code for the oneway model with improper priors (from Tan and Hobert (2009)). In this chapter we also present a minorization for a model that is used to search for structure in collections of documents. It is known as a Latent Dirichlet Allocation (LDA) model or a Topic Model. The code to carry out regeneration of the LDA model with the minorization proposed in Section 4.3 is included with McParre. Its usage is demonstrated here with a simulated dataset.

4.1 Code and Method Validation

In this section, we assess if running regenerative chains in the manner we purport - by running chains in parallel and then concatenating them, will provide confidence intervals with the correct coverage. The model we shall consider is the oneway hierarchical linear model, as specified in Hobert and Geyer (1998):

\[ y_{ij} = \theta_i + \epsilon_{ij}, \quad i = 1, 2, \ldots, K; \quad j = 1, 2, \ldots, m_i \]  

(4–1)

where

\[ y_{ij}|\theta, \lambda_e \sim N(\theta_i, \lambda_e^{-1}) \]
\[ \theta|\mu, \lambda_\theta \sim N(1\mu, 1\lambda_\theta^{-1}) \quad \lambda_e \sim \Gamma(a_2, b_2) \]
\[ \mu \sim N(\mu_0, \lambda_\mu^{-1}) \quad \lambda_\theta \sim \Gamma(a_1, b_1) \]

Suppose that \( K = 2 \), and \( m_i = m \) for all \( i \). In this simple case, we can analytically integrate out \( \mu \) and \( \lambda_e \) and then numerically obtain (any function of) the posterior means of \( \theta_1, \theta_2 \) and \( \lambda_\theta \). We used the triplequad function of MATLAB, which allows computation.
of a triple integral, but there are many other programs that will do the job too. Let the joint density be

\[ h(\theta, \mu, \lambda, \lambda_e, y) = f(y|\theta, \lambda_e)f(\theta|\mu, \lambda)f(\lambda_e)f(\mu)f(\lambda) \]  

(4–2)

Then the following simple proposition reduces the expression for the posterior mean for \( \theta \) and \( \lambda_e \) from a 5-dimensional integral to a 3-dimensional one.

**Proposition 4.1.1. As long as the posterior means for \( \theta_1, \theta_2 \) and \( \lambda_e \) exist,**

\[
\mathbb{E}(\zeta|y) = \frac{\int \zeta h(\theta, \mu, \lambda_e, \lambda_e, y)d(\theta, \lambda, \lambda_e, \mu)}{\int h(\theta, \mu, \lambda_e, \lambda_e, y)d(\theta, \lambda, \lambda_e, \mu)} = \frac{\int \zeta f_1(\lambda_e, \theta)f_2(\theta, y)f(\lambda_e)d(\theta_1, \theta_2, \lambda_e)}{\int f_1(\lambda_e, \theta)f_2(\theta, y)f(\lambda_e)d(\theta_1, \theta_2, \lambda_e)}
\]  

(4–3)

where \( \zeta = \theta_1, \theta_2 \) or \( \lambda_e \), and

\[ f_1(\theta, \lambda_e) = \frac{\lambda_e}{(\lambda_0 + 2\lambda_e)^{1/2}} \exp \left\{ -\frac{\lambda_e}{2}(\theta_1^2 + \theta_2^2) + \frac{(\lambda_0\mu_0 + \lambda_e\theta_1 + \lambda_e\theta_2)^2}{2(\lambda_0 + 2\lambda_e)} \right\} \]

\[ f_2(\theta, y) = \left( b_2 + \frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} (y_{ij} - \theta_j)^2 \right)^{-m-a_2} \]

**Proof.** The first equality holds because the joint posterior density is given by

\[ f(\theta, \mu, \lambda, \lambda_e|y) = \frac{h(\theta, \mu, \lambda, \lambda_e, y)}{\int h(\theta, \mu, \lambda, \lambda_e, y)d(\theta, \lambda, \lambda_e, \mu)} \]

For the second equality, note that in simplifying the integrals, we can drop any terms that do not involve \( \theta, \lambda_e, \lambda_e \) or \( \mu \) since they will cancel out in the ratio of integrals. Whenever...
this is done in the sequel, we shall indicate it with a $\varnothing$. To get rid of $\mu$, observe that

$$
\int f(\theta|\mu, \lambda_\theta) f(\mu) d\mu = \left( \frac{\lambda_\theta}{2\pi} \right)^{\frac{1}{2}} \int e^{-\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2} e^{\frac{\lambda_\theta}{\lambda_0} (\theta_2 - \mu)^2} \frac{d\mu}{\lambda_0 + 2\lambda_\theta} \int e^{-\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2} \exp \left\{ -\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2 + \frac{(\lambda_0 \mu_0 + \lambda_\theta \theta_1 + \lambda_\theta \theta_2)^2}{2(\lambda_0 + 2\lambda_\theta)} \right\} d\mu
$$

$$
\cong \lambda_\theta \int e^{-\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2} \exp \left\{ -\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2 + \frac{(\lambda_0 \mu_0 + \lambda_\theta \theta_1 + \lambda_\theta \theta_2)^2}{2(\lambda_0 + 2\lambda_\theta)} \right\} d\mu
$$

$$
= \lambda_\theta \exp \left\{ -\frac{\lambda_\theta}{\lambda_0} (\theta_1^2 + \theta_2^2) \right\} \int e^{-\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2} \exp \left\{ -\frac{\lambda_\theta}{\lambda_0} (\theta_1 - \mu)^2 + \frac{(\lambda_0 \mu_0 + \lambda_\theta \theta_1 + \lambda_\theta \theta_2)^2}{2(\lambda_0 + 2\lambda_\theta)} \right\} d\mu
$$

$$
= \frac{\lambda_\theta}{(\lambda_0 + 2\lambda_\theta)^{\frac{1}{2}}} \exp \left\{ -\frac{\lambda_\theta}{\lambda_0} (\theta_1^2 + \theta_2^2) + \frac{(\lambda_0 \mu_0 + \lambda_\theta \theta_1 + \lambda_\theta \theta_2)^2}{2(\lambda_0 + 2\lambda_\theta)} \right\}
$$

$$
\equiv f_1(\theta, \lambda_\theta)
$$

To get rid of $\lambda_e$, observe that

$$
\int f(\theta|\lambda_\theta, \lambda_e) f(\lambda_e) d\lambda_e = \left( \frac{\lambda_e}{2\pi} \right)^m \exp \left\{ -\frac{\lambda_e}{2} \sum_{j=1}^m \sum_{i=1}^m (y_{ij} - \theta_j)^2 \right\} \frac{b_2^{a_2}}{\Gamma(a_2)} \lambda_e^{-1} \exp \left\{ -\lambda_e b_2 \right\} d\lambda_e
$$

$$
= \frac{b_2^{a_2}}{(2\pi)^m \Gamma(a_2)} \int \lambda_e^{m+a_2-1} \exp \left\{ -\lambda_e \left( b_2 + \frac{1}{2} \sum_{j=1}^m \sum_{i=1}^m (y_{ij} - \theta_j)^2 \right) \right\} d\lambda_e
$$

$$
= \frac{\Gamma(m+a_2)}{(b_2 + \frac{1}{2} \sum_{j=1}^m \sum_{i=1}^m (y_{ij} - \theta_j)^2)^{m+a_2}}
$$

$$
\equiv f_2(\theta, \lambda_e)
$$

Substituting the above analytical expressions into $h(\cdot)$ in equations (4–2) and (4–3), we have that

$$
\int \zeta h(\theta, \mu, \lambda_\theta, \lambda_e, \lambda_e, \mu) d(\theta, \lambda_\theta, \lambda_e, \mu) = \int \zeta f_1(\lambda_\theta, \theta) f_2(\theta, \lambda_e) f(\lambda_e) d(\theta_1, \theta_2, \lambda_\theta)
$$
and
\[
\int h(\theta, \mu, \lambda_\theta, \lambda_e, y) d(\theta, \lambda_\theta, \lambda_e, \mu) = \int f_1(\lambda_\theta, \theta) f_2(\theta, y) f(\lambda_\theta) d(\theta_1, \theta_2, \lambda_\theta)
\]  
(4–5)

4.1.1 Minorizing the Oneway Model

For this model, the usual way of obtaining samples from the posterior distribution is by running a Gibbs sampler. Suppressing the dependence on \(y\), the univariate conditional densities needed for the Gibbs sampler are

- \(\lambda_\theta | \lambda_e, \mu, \theta \sim \text{Gamma} \left( \frac{K}{2} + a_1, b_1 + \frac{1}{2} \sum_i (\theta_i - \mu)^2 \right)\)
- \(\lambda_e | \lambda_\theta, \mu, \theta \sim \text{Gamma} \left( \frac{N}{2} + a_2, b_2 + \frac{1}{2} \sum_{i,j} (y_{ij} - \theta_i)^2 \right)\)
- \(\mu | \lambda_\theta, \lambda_e, \theta \sim N \left( \frac{\lambda_\theta \mu_0 + K \lambda_\theta \bar{\theta}}{\lambda_0 + K \lambda_\theta}, \frac{1}{\lambda_0 + K \lambda_\theta} \right)\)
- \(\theta_i | \lambda_\theta, \lambda_e, \mu, \theta_{-i} \sim N \left( \frac{\lambda_\theta \mu + m_i \lambda_e \bar{y}_i}{\lambda_\theta + m_i \lambda_e}, \frac{1}{\lambda_\theta + m_i \lambda_e} \right) \quad i = i, \ldots, K\)

(4–6)

It is not necessary to restrict the dimensions of the problem to \(K = 2\) in order to minorize this chain, so we shall work with the general case. Denote the state of the Gibbs sampler using a vector of length \(K + 3\). Thus \(X_n = x\) means that \(\lambda^n_\theta = x_1, \lambda^n_e = x_2, \mu^n = x_3\) and \(\theta^n = (x^n_1, \ldots, x^n_{K+3})\). Updating \(\lambda_\theta\) first, then \((\lambda_e, \mu)\), and finally \(\theta\) yields a 3-stage Gibbs sampler. This is a cyclic permutation of the chain that was proven to be geometrically ergodic in Hobert and Geyer (1998). If we use \(p_1, \ldots, p_{K+3}\) to represent each of the conditional densities above, we can write the entire Markov transition density
as

\[ p(\mathbf{x}, \mathbf{w}) = p_1(w_1|x_2, x_3, \ldots, x_{K+3}) \]
\[ \times p_2(w_2|w_1, x_3, x_4, \ldots, x_{K+3}) \]
\[ \times p_3(w_3|w_1, w_2, x_4, \ldots, x_{K+3}) \]
\[ \times p_4(w_4|w_1, w_2, w_3, x_5, \ldots, x_{K+3}) \cdots \]
\[ \times p_{K+3}(w_{K+3}|w_1, w_2, w_3, w_4, \ldots, w_{K+2}) \]

Following Mykland et al. (1995), we shall minorize this Gibbs sampler in the following way. Choosing a distinguished point \( \bar{x} \) and a compact set \( D \), we have

\[
\begin{align*}
p(\mathbf{x}, \mathbf{w}) &= \inf_{z \in D} \left\{ p(\mathbf{x}, \mathbf{z}) \mid p(\bar{x}, \mathbf{w}) \right\} \int_{D} p(\mathbf{y} \mid \mathbf{w} \in D) \, d\mu(\mathbf{y}) \\
&= \inf_{z \in D} \left\{ p(\mathbf{x}, \mathbf{z}) \mid p(\bar{x}, \mathbf{w}) \right\} \int_{D} \nu(\mathbf{y}) \\
&= \mathbb{P}(\mathbf{x} \mid \mathbf{w} \in D) \int_{D} \nu(\mathbf{y})
\end{align*}
\]

The infimum over \( D \) can be computed analytically, so as to avoid using a software optimization routine. Here we provide the details. Consider \( p(\mathbf{x}, \mathbf{z}) / p(\bar{x}, \mathbf{z}) \), and define \( \bar{x} = (1/K) \sum_{i=1}^{K} x_{i+3} \) and \( \bar{y} = (1/K) \sum_{i=1}^{K} \bar{x}_{i+3} \).

\[
\begin{align*}
p(\mathbf{x}, \mathbf{z}) &\sim \frac{b_1 + \frac{1}{2} \sum_i (x_{i+3} - x_3)^2} {b_1 + \frac{1}{2} \sum_i (\bar{x}_{i+3} - \bar{x}_3)^2} \\
&\times \frac{z_1^{K/2+a_1} \exp (-z_1 b_1 + \frac{1}{2} \sum_i (x_{i+3} - x_3)^2)} {z_1^{K/2+a_1} \exp (-z_1 b_1 + \frac{1}{2} \sum_i (\bar{x}_{i+3} - \bar{x}_3)^2)} \\
&\times \frac{z_2^{N/2+a_2} \exp (-z_2 b_2 + \frac{1}{2} \sum_{ij} (y_{ij} - x_{i+3})^2)} {z_2^{N/2+a_2} \exp (-z_2 b_2 + \frac{1}{2} \sum_{ij} (\bar{y}_{ij} - \bar{x}_{i+3})^2)} \\
&\times \left( \frac{\lambda_0 + Kz_1}{2\pi} \right)^{1/2} \exp \left( -\frac{\lambda_0 + Kz_1}{2} \left[ z_3 - \frac{\lambda_0 \mu_0 + Kz_1}{\lambda_0 + Kz_1} \right]^2 \right) \\
&\times \left( \frac{\lambda_0 + Kz_1}{2\pi} \right)^{1/2} \exp \left( -\frac{\lambda_0 + Kz_1}{2} \left[ z_3 - \frac{\lambda_0 \mu_0 + Kz_1}{\lambda_0 + Kz_1} \right]^2 \right)
\end{align*}
\]
Minimizing this is equivalent to minimizing $\log(p(x, z)/p(\bar{x}, z))$, and

$$
\log \frac{p(x, z)}{p(\bar{x}, z)} = \text{Constant} - z_1 \frac{1}{2} \sum_{i=1}^{K} \left[ (x_{i+3} - x_3)^2 - (\bar{x}_{i+3} - \bar{x}_3)^2 \right] c_1
$$

$$
- z_2 \frac{1}{2} \sum_{i,j} \left[ (y_{ij} - x_{i+3})^2 - (y_{ij} - \bar{x}_{i+3})^2 \right] c_2
$$

$$
- \left( \frac{\lambda_0 + Kz_1}{2} \right) \left( 2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{x} + \bar{x})}{\lambda_0 + Kz_1} \right)
$$

The final expression in the above display can be simplified to yield

$$
- \left( \frac{\lambda_0 + Kz_1}{2} \right) \left( 2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{x} + \bar{x})}{\lambda_0 + Kz_1} \right)
$$

Hence we can define the function

$$
g(z_1, z_2, z_3) = -z_1 C_1 - z_2 C_2 - \frac{Kz_1(\bar{x} - \bar{x})}{2} \left( 2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{x} + \bar{x})}{\lambda_0 + Kz_1} \right)
$$

(4–8)

and attempt to find $(z_1^*, z_2^*, z_3^*)$ that minimizes $g$ over

$$
D = [d_{11}, d_{12}] \times [d_{21}, d_{22}] \times [d_{31}, d_{32}]
$$

From (4–8), the partial derivatives are

$$
\frac{\partial g}{\partial z_1} = -C_1 - K(\bar{x} - \bar{x})z_3 + \frac{1}{2} K(\bar{x} - \bar{x}) \left[ \frac{2\lambda_0\mu_0 + Kz_1(\bar{x} + \bar{x})}{\lambda_0 + Kz_1} \right]
$$

$$
\frac{\partial g}{\partial z_2} = -C_2
$$

$$
\frac{\partial g}{\partial z_3} = -Kz_1(\bar{x} - \bar{x})
$$

Since $z_1 > 0$, we can set $z_2^*$ and $z_3^*$ once we know the signs of $\partial g/\partial z_2$ and $\partial g/\partial z_3$ above.

For $z_1^*$, we can set $\partial g/\partial z_1 = 0$ and solve for the roots. Here are the coefficients of the
quadratic equation in $z_1$.

\[
\text{Coeff of } z_1^2 = \frac{1}{2} K^2 \left(-2C_1 - K(\bar{x} - \bar{x})(2z_3 - \bar{x} - \bar{x})\right)
\]

\[
\text{Coeff of } z_1 = 2K\lambda_0 (-C_1 - Kz_3(\bar{x} - \bar{x})) + K(\bar{x} - \bar{x})(K\lambda_0(\bar{x} + \bar{x}))
\]

\[
\text{Constant term} = \lambda_0^2 \left[-C_1 - K(\bar{x} - \bar{x})z_3 + K\mu_0(\bar{x} - \bar{x})\right]
\]

### 4.1.2 The Experiment

In order to assess the performance of the regenerative chains, we shall fix the hyper-parameters as $\lambda_0 = 20.3$, $\mu_0 = 2.19$, $a_1 = 2.1$, $b_1 = 4.3$, $a_2 = 2.1$ and generate a dataset with $m_1 = m_2 = 7$ observations. Using the method outlined in Proposition 4.1.1, we can numerically compute the posterior means of $\theta_1$, $\theta_2$ and $\lambda_\theta$. These values are displayed in Table 4-1. They are the gold standard that the regenerative procedure should pick up and form confidence intervals around.

For the experiment, 50000 tours were generated on 5 processors. This was done by making each of the processors complete 10000 tours. This entire process was repeated 100 times. In order to assess whether sufficient tours had been collected, we could imagine stopping each of those 100 chains after 5000, 25000 and then 50000 tours. In practice, we did this by considering the first 1000, 5000 and then 10000 tours generated by each processor, at each of the 100 iterations. This would mimic stopping the chain prematurely. At these “pseudo-breakpoints”, we can form estimates of the posterior means that we are after, and also of the Coefficient of Variation of the tours. The 100 estimates of the posterior mean were then used to form density estimates of theorised Normal approximation that they should be converging in distribution to. These plots can be found in 4-3.

The results from the simulation runs are summarised in Table 4-2 and Figure 4-3. The tables indicate that the coverage probabilities approach the nominal value as we collect more tours. The figure 4-3 shows us that as we collect more and tours, the distribution of $\bar{g}_R$ approaches what we expect. Figures 4-2 and 4-1 contain plots of the
diagnostics considered in Section 3.3. In this particular case, they do not appear to be useful in determining if sufficient tours have been collected. Both of them seem to suggest that 5000 tours are sufficient, but the density plots in Fig 4-3 suggest otherwise.

The main aim of our methodology is to improve timings, while retaining a theoretically water-tight inference procedure. Hence we re-ran the dataset from previous subsection on 1, 2 and 5 processors in order to assess the speed-up that we gain. This time, we only ran 10 replications for each configuration of processors. The results are in Table 4-3. The code that was used to run this analysis in parallel is available in McParre as genNextStateOnewayPlain and regenProbsOnewayPlain.

### 4.2 Hierarchical Linear Mixed Model (HLMM)

In this section, we demonstrate the use of McParre on the Hierarchical Linear Mixed Model introduced in Hobert et al. (2006). The block Gibbs sampler for this chain has recently been proven to be geometrically ergodic. First we introduce the model and the notation used, which closely follows that of Hobert et al. (2006). The minorization introduced there will also be presented and used in the numerical example that follows.

The model is the Bayesian version of the usual frequentist general linear mixed model:

\[
Y | \beta, u, R, D \sim N_n(X\beta + Zu, R^{-1}) \\
\beta | u, R, D \sim N_p(\beta_0, B^{-1}) \\
u | D, R \sim N_q(0, D^{-1})
\]

\[R = \lambda_R I_n \text{ where } \lambda_R \sim \text{Gamma}(r_1, r_2)\]

\[D = \lambda_D I_q \text{ where } \lambda_D \sim \text{Gamma}(d_1, d_2)\]

(4–9)

The parameters \(\beta_0, B, r_1, r_2, d_1\) and \(d_2\) are all assumed to be known, and \(X\) is assumed to be of full column rank.
If we let $\xi = (u, \beta)$, then the Gibbs chain samples the variables in the following manner:

$$(\lambda_D, \lambda_R, \xi) \rightarrow (\lambda_D', \lambda_R', \xi) \rightarrow (\lambda_D', \lambda_R', \xi')$$

We essentially have a 2-stage Gibbs sampler, as we can see from the conditionals that $\lambda_D$ and $\lambda_R$ are independent of each other when given $\xi$. These are the full conditionals:

$$(\lambda_D|\xi, Y) \sim \text{Gamma} \left(\frac{q}{2} + d_1, \frac{d_2 + 1}{2} u^T u\right)$$

$$(\lambda_R|\xi, Y) \sim \text{Gamma} \left(\frac{n}{2} + r_1, \frac{r_2 + 1}{2} (Y - X\beta - Zu)^T (Y - X\beta - Zu)\right)$$

$$(\xi|R, D, Y) \sim N_{p+q}(\xi_0, \Sigma^{-1})$$

where the mean and the covariance matrix for $\xi$ are given by

$$\Sigma = \begin{pmatrix}
Z^T R Z + D & Z^T R X \\
X^T R Z & X^T R X + B
\end{pmatrix}$$

and

$$\xi_0 = \Sigma^{-1} \begin{pmatrix}
Z^T R Y \\
X^T R Y + B \beta_0
\end{pmatrix}$$

Choosing a distinguished set for the 2 variables $\lambda_R$ and $\lambda_D$, the Mykland et al. (1995) method would lead us to the following minorization.

$$p((\lambda_D, \lambda_R, \xi), (\lambda'_D, \lambda'_R, \xi')) \geq \left[ \inf_{\lambda_D \in D_1} \frac{\pi(\lambda_D|\xi)}{\pi(\lambda_D'|\xi)} \right] \left[ \inf_{\lambda_R \in D_2} \frac{\pi(\lambda_R|\xi)}{\pi(\lambda_R'|\xi)} \right] \times p((\tilde{\lambda}_D, \tilde{\lambda}_R, \tilde{\xi}), (\lambda'_D, \lambda'_R, \xi')) I[\lambda_D' \in D_1] I[\lambda_R' \in D_2]$$

The small function can be analytically evaluated, avoiding any numerical optimization routines.

For a numerical example, we can consider the dataset that is available in the MASS library, and analysed in Chapter 10 of Venables and Ripley (2002). This data was originally used to build an estimation equation for the yield of the refining process of gasoline. The possible predictors were the specific gravity, vapour pressure, volatility and desired volatility of the samples. In total 10 samples were taken; within each sample varying numbers of yield readings were computed. We assume that the samples give
rise to a random intercept term in the linear model. Thus the frequentist linear model is of the form:

\[ y_{ij} = \mu + \zeta_i + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \beta_4 x_{4i} + \epsilon_{ij} \]  

(4–10)

where the \( x_i \)'s represent the predictors mentioned earlier. The \( \zeta_i \sim N(0, \sigma_1^2) \) and the \( \epsilon_{ij} \sim N(0, \sigma^2) \).

In Venables and Ripley (2002), the above model was fit using REML and ML. The results are contained in Tables 4-4 and 4-5. McParre was used to regenerate a Markov chain for 1000 tours in order to estimate the parameters of the model above. The estimates are contained in Table 4-6. For the priors, we used parameters of \( r_1 = d_1 = 100 \) and \( r_2 = d_2 = 3 \) in order to let the “data” speak for the model. We also include timings for 10 runs each, using 1, 2, and 5 processors to complete the 10,000 tours. These times are in Table 4-7. The code that was used to analyze this data, and HLMM in general, is included in McParre as genNextStateHLMM, smallFnHLMM, smallMeasureHLMM and transDensHLMM.

### 4.3 Topic Models

The latent dirichlet allocation (LDA) model describes one approach to describing collections of discrete data such as a collection of text documents. It is fully developed in Blei et al. (2003). Here we describe the full model, that allows us to run a Gibbs sampler in order to estimate the posterior distributions.

Using terminology from text collections, suppose that we have \( D \) documents in a corpus, and that after removing the stop words, each document has \( n_d \) words. Also let \( V \) denote the vocabulary of the corpus - the total number of unique words. Then each document can be represented by a sequence of \( n_d \) binary vectors, each of length \( V \), representing the \( n_d \) words in that document. We shall use \( w_{1,d}, w_{2,d}, \ldots, w_{n_d,d} \) to represent the words. The co-ordinate indicator \( w_{t,d} = 1 \) will tell us exactly which word of the vocabulary is being picked up, with \( t \) running from 1 to \( V \). The words are observed, but for each word in document \( d \), let \( z_{i,d} \) denote a latent variable, also a binary
vector, that identifies the topic that \( w_{i,d} \) belongs to. A topic can be defined simply as a distribution over the vocabulary of words. Assume we know that there are \( k \) topics in the corpus. One goal of LDA is to identify words that define a topic. The topics themselves are unlabelled, but from the model parameters, they can be identified by the words in the vocabulary that are most likely to appear under that topic. Table 4-8 contains a summary of the index labels used in this section.

Using the binary vector notation described above simplifies the notation in expressions for the joint density. However, when programming, the \( z_d \) and variables will be represented with a vector of length equal to the total number of words. Each element will be an integer between 1 and \( k \), indicating the topic to which that word has been assigned. Similarly, the observed words \( w \) can be represented by integers between 1 and \( V \) representing the exact word in reference to the vocabulary. This secondary notation style is more concise for discussion and will be used later when describing the example.

Now let \( \beta \) represent a random \( k \times V \) matrix, where each row represents the probability vector over the vocabulary corresponding to one topic. Formally, the data generating hierarchy is as follows:

\[
\begin{align*}
\beta_j & \sim \text{Dirichlet}(b_1, b_2, \ldots, b_V) \quad \text{for} \quad j = 1, 2, \ldots, k \\
\theta_d & \sim \text{Dirichlet}(\alpha_1, \alpha_2, \ldots, \alpha_k) \quad \text{for} \quad d = 1, 2, \ldots, D
\end{align*}
\]

For document \( d \), for \( i = 1, 2, \ldots, n_d \):

\[
\begin{align*}
\text{For document } d, \text{ for } i = 1, 2, \ldots, n_d: \\
z_{i,d} & \sim \text{Multinomial}(1, \theta_d) \\
w_{i,d} & \sim \text{Multinomial} \left( 1, \sum_{j=1}^{k} \beta_j \left[ z_{ij,d} = 1 \right] \right)
\end{align*}
\]

(4–11)

Informally, the above display simply says that each document is spread across the topics, with the spread being determined by \( \theta_d \). For each word within the document, we
pick a random topic first \((z_{i,d})\), and then pick the word according to the distribution \((\beta)\) determined by that topic. The next step is to write down the conditional distributions for \(z_{i,d}\) and \(w_{i,d}\) for each document \(d\) and simplify them.

\[
\Pr(z_d|\theta_d) = \prod_{i=1}^{n_d} \prod_{j=1}^{k} \theta_{z_{ij},d}^{z_{ij},d}
\]

\[
= \prod_{j=1}^{k} \theta_{n_{j,d}}^{n_{j,d}} \quad \text{where } n_{j,d} = \sum_{i=1}^{n_d} z_{ij,d}
\]

\[
\Pr(w_d|z_d, \beta) = \prod_{i=1}^{n_d} \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{w_{it},d}^{w_{it},d,z_{ij},d}
\]

\[
= \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{m_{jt},d}^{m_{jt},d} \quad \text{where } m_{jt,d} = \sum_{i=1}^{n_d} w_{it,d}z_{ij,d}
\]

In this context, \(n_{j,d}\) represents the number of words in document \(d\) that are assigned to topic \(j\), and \(m_{jt,d}\) represents the number of times that word \(t\) in document \(d\) is assigned to topic \(j\). The joint density can be written as

\[
\Pr(z, \theta, \beta, w|\alpha, b) = \prod_{d=1}^{D} \Pr(w_d|z_d, \beta) \Pr(z_d|\theta_d) \Pr(\theta_d|\alpha_1, \ldots, \alpha_k) \Pr(\beta|b_1, \ldots, b_n)
\]

The joint posterior is given by

\[
\Pr(z, \theta, \beta|w, \alpha, b) \propto \prod_{d=1}^{D} \left( \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{m_{jt},d}^{m_{jt},d} \right) \left( \prod_{j=1}^{k} \theta_{n_{j,d}}^{n_{j,d}} \right) \prod_{j=1}^{k} \theta_{\alpha_j-1}^{\alpha_j-1} \prod_{j=1}^{k} \beta_{b_{jt}-1}^{b_{jt}-1}
\]

\[
= \prod_{d=1}^{D} \left( \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{m_{jt},d}^{m_{jt},d} \right) \left( \prod_{j=1}^{k} \theta_{n_{j,d}+\alpha_j-1}^{n_{j,d}+\alpha_j-1} \right) \prod_{j=1}^{k} \beta_{b_{jt}-1}^{b_{jt}-1}
\]

\[
= \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{\sum_{d=1}^{D} m_{jt,d} + b_{jt}-1}^{\sum_{d=1}^{D} m_{jt,d} + b_{jt}-1} \left( \prod_{d=1}^{D} \prod_{j=1}^{k} \theta_{n_{j,d}+\alpha_j-1}^{n_{j,d}+\alpha_j-1} \right)
\]

\[
= \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{\sum_{d=1}^{D} m_{jt,d} + b_{jt}-1}^{\sum_{d=1}^{D} m_{jt,d} + b_{jt}-1} \left( \prod_{d=1}^{D} \prod_{j=1}^{k} \theta_{n_{j,d}+\alpha_j-1}^{n_{j,d}+\alpha_j-1} \right)
\]

We can now pick off the full conditionals of a 2-stage Gibbs sampler.
1. Given $\Delta = (\beta, \theta)$, and noting that $n_{j,d} = \sum_{t=1}^{V} \sum_{i=1}^{n_d} z_{ij,t} w_{it,d} = \sum_{t=1}^{V} m_{jt,d}$, the full conditional for $z_{i,d}$ is

$$
\Pr(z_{i,d} | \theta_d, \beta, w_d) \propto \left( \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{jt}^{m_{jt,d}} \right) \left( \prod_{j=1}^{k} \theta_{j,d}^{n_{j,d}} \right)
$$

$$
= \left( \prod_{i=1}^{n_d} \prod_{t=1}^{V} \prod_{j=1}^{k} \beta_{jt}^{z_{ij,t} w_{it,d}} \right) \left( \prod_{j=1}^{k} \prod_{t=1}^{V} \prod_{i=1}^{n_d} \theta_{j,d}^{z_{ij,t} w_{it,d}} \right)
$$

Thus we can generate $z_{i,d}$ according to

$$
z_{i,d} \sim \text{Multinomial}(1, (p_{i,1,d}, p_{i,2,d}, \ldots, p_{i,k,d})) \quad \text{for} \quad i = 1, 2, \ldots, n
$$

where $p_{ij,d} \propto \prod_{t=1}^{V} (\beta_{jt} \theta_{j,d})^{w_{it,d}}$ for $j = 1, 2, \ldots, k$

2. Given $z$, generate $\beta$ and $\theta$ via

$$
\theta_d \sim \text{Dirichlet}(n_{1,d} + \alpha_1, \ldots, n_{k,d} + \alpha_k) \quad \text{for} \quad d = 1, 2, \ldots, D
$$

$$
\beta_j \sim \text{Dirichlet} \left( \sum_{d=1}^{D} m_{j1,d} + b_1, \ldots, \sum_{d=1}^{D} m_{jV,d} + b_V \right) \quad \text{for} \quad j = 1, 2, \ldots, k
$$

As this is a 2-stage Gibbs sampler with one of the random variables over a finite state space, we can apply the Duality Principle (see Section 9.2.3 of Robert and Casella (2004) and Roberts and Rosenthal (2001) for further details) and conclude that the overall chain is uniformly ergodic. Hence Theorem 2.2.1 holds.

Now we turn to minorizing this chain. First observe that there are 2 ways to run it. Let $(z_x, \beta_x, \theta_x)$ and $(z_y, \beta_y, \theta_y)$ denote two possible states from the posterior distribution. To go from point $(z_x, \beta_x, \theta_x)$ to $(z_y, \beta_y, \theta_y)$, we could do

$$(z_x, \beta_x, \theta_x) \rightarrow (z_y, \beta_x, \theta_x) \rightarrow (z_y, \beta_y, \theta_y) \quad (4-13)$$

or we could do

$$(z_x, \beta_x, \theta_x) \rightarrow (z_x, \beta_y, \theta_y) \rightarrow (z_y, \beta_y, \theta_y) \quad (4-14)$$
Consider using transition 4–13 first. In this case, the Markov transition density is given by

\[ g_1((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) = p_1(z_y|\beta_x, \theta_x)p_{2a}(\beta_y|z_y)p_{2b}(\theta_y|z_y) \]  \hspace{1cm} (4–15) 

We shall derive two possible minorizations, both based on the Mykland et al. (1995) approach of distinguished points and sets. Denote the distinguished point by \((\tilde{z}, \tilde{\beta}, \tilde{\theta})\).

Recall that \(z\) represents the assignments of words to latent topics. Suppose we know that a certain group of such assignments is significantly probable. Let it be denoted by \(D_1\). Then we can arrive at a very direct and simple-to-understand minorization:

\[ g_1((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) \geq \left[ \inf_{z^* \in D_1} \frac{p_1(z^*|\beta_x, \theta_x)}{p_1(z^*|\tilde{\beta}, \tilde{\theta})} \right] \times g_1((\tilde{z}, \tilde{\beta}, \tilde{\theta}), (z_y, \beta_y, \theta_y)) I[z_y \in D_1] \]  \hspace{1cm} (4–16) 

The small function is given by the expression in square brackets while the small measure is given by the rest of the right-hand-side.

Naturally the question arises, of how best to choose to \(D_1\). Note that the number of possible assignments is extremely large - for 50 words and \(k = 3\) topics, there are theoretically \(3^{50}\) such \(z\) assignments. Computationally, we would have to store \(D_1\) as a look-up table, probably using a hash function. Every time a \(z\) is generated, we would have to search through this table for a match, and then compute the infimum. This approach might not scale very well as the number of words increases, since the number of possible \(z\)'s grows exponentially.

An alternative approach is to take the infimum over all possible \(z\)'s. This would allow us to consider regeneration points at every iteration, instead of only at points where \(z \in D_1\). The downside is that the regeneration probabilities would be extremely small. Here are the details:
Then we can re-write the product term in the small function as

$$
\inf_{z^* \in D_2} \frac{p_1(z^* | \beta, \theta)}{p_1(z^* | \tilde{\beta}, \tilde{\theta})} = \inf_{z^* \in D_2} \prod_{d=1}^{D} \prod_{i=1}^{n_d} \prod_{j=1}^{k} \left( \frac{p_{ij,d}}{\tilde{p}_{ij,d}} \right) z_{ij,d}^*
$$

Define the normalising constant present in the $p_{ij,d}$ by

$$
C_{i,d,\beta,\theta} = \sum_{j=1}^{k} \prod_{t=1}^{V} (\beta_{jt} \theta_{j,d})^{w_{a,t,d}}
$$

Then we can re-write the product term in the small function as

$$
\prod_{d=1}^{D} \prod_{i=1}^{n_d} \prod_{j=1}^{k} \left[ \prod_{t=1}^{V} (\beta_{jt} \theta_{j,d})^{w_{a,t,d}} / C_{i,d,\beta,\theta} \right] z_{ij,d}^* = \prod_{d=1}^{D} \prod_{i=1}^{n_d} \left[ \frac{C_{i,d,\beta,\tilde{\theta}}}{C_{i,d,\beta,\theta}} \right] \times \prod_{d=1}^{D} \prod_{i=1}^{n_d} \prod_{j=1}^{k} \prod_{t=1}^{V} \left( \frac{\beta_{jt} \theta_{j,d}}{\beta_{jt} \tilde{\theta}_{j,d}} \right)^{w_{a,t,d} z_{ij,d}^*}
$$

We can see that this expression is greater than or equal to

$$
s_1(z, \beta, \theta) = \prod_{d=1}^{D} \prod_{i=1}^{n_d} C_{i,d,\beta,\tilde{\theta}} \prod_{d=1}^{D} \left( \min_{j} \frac{\theta_{j,d}}{\tilde{\theta}_{j,d}} \right)^{n_d} \left( \min_{j} \frac{\beta_{jt}}{\tilde{\beta}_{jt}} \right)^{n_d}
$$

(4–17)

This $s_1$ function is no longer a function of $z^*$ as, essentially, the infimum is being taken over the entire space. This gives us a second possible minorization to consider.

$$
g_1((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) \geq s_1(z_x, \beta_x, \theta_x) \times g_1((\tilde{z}, \tilde{\beta}, \tilde{\theta}), (z_y, \beta_y, \theta_y)) I[z_y \in D_1]
$$

(4–18)

Note that the normalizing constant in the expression for $s_1$ will cancel out when we compute the regeneration probability. The latter probability is given by

$$
\frac{s_1(z_x, \beta_x, \theta_x) g_1((\tilde{z}, \tilde{\beta}, \tilde{\theta}), (z_y, \beta_y, \theta_y)) I[z_y \in D_1]}{g_1((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y))}
$$

The constant in the ratio of the $g_1$ functions will be

$$
\prod_{d=1}^{D} \prod_{i=1}^{n_d} C_{i,d,\beta_x,\theta_x} \prod_{d=1}^{D} \prod_{i=1}^{n_d} C_{i,d,\beta,\theta}
$$

56
which is the reciprocal of the normalizing constant present in the small function \( s_1 \) (see equation 4–17).

We can derive a third possible minorization by running the chain using the scheme specified in (4–14), in which case the Markov transition density is given by

\[
g_2((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) = p_{2a}(\beta_y|z_x)p_{2b}(\theta_y|z_x)p_1(z_y|\beta_y, \theta_y) \tag{4–19}
\]

The most apparent minorization is given by

\[
g_2((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) \geq \left[ \inf_{\beta^*, \theta^* \in D_2} \frac{p_{2a}(\beta^*|z_x)p_{2b}(\theta^*|z_x)}{p_{2a}(\beta^*|\tilde{z})p_{2b}(\theta^*|\tilde{z})} \right] \times g_2((\tilde{z}, \tilde{\beta}, \tilde{\theta}), (z_y, \beta_y, \theta_y)) \tag{4–20}
\]

The small function given in the square brackets is not at all straightforward to minimize, so it has to be simplified further. As an example, consider the ratio term for \( \theta \) for one of the documents. Hence we shall drop the \( d \) subscript for now.

\[
p_{2b}(\theta|z_x) = \frac{\prod_{j=1}^{k} \Gamma(\tilde{n}_j + \alpha_j)}{\prod_{j=1}^{k} \Gamma(n_j + \alpha_j)} \prod_{j=1}^{k} \theta_j^{n_j - \tilde{n}_j} \geq \prod_{j=1}^{k} \frac{\Gamma(n_j + \alpha_j)}{\Gamma(\tilde{n}_j + \alpha_j)} \theta_j^{(n_j - \tilde{n}_j)I[n_j > \tilde{n}]} \tag{4–20}
\]

The above inequality can be seen to be true by taking logs, and remembering that \( \log \theta_j < 0 \):

\[
\sum_{j=1}^{k} (n_j - \tilde{n}_j) \log \theta_j + \text{constant} \geq \sum_{j=1}^{k} I[n_j > \tilde{n}] (n_j - \tilde{n}_j) \log \theta_j + \text{constant}
\]

We can now define our small function.

\[
\inf_{\beta, \theta \in D_2} \frac{p_{2a}(\beta|z_x)p_{2b}(\theta|z_x)}{p_{2a}(\beta|\tilde{z})p_{2b}(\theta|\tilde{z})} \geq \prod_{j=1}^{k} \prod_{t=1}^{V} \frac{\Gamma(m_{jt} + b_t)}{\Gamma(m_{jt} + b_t)} \beta_{jt}^{(m_{jt} - \tilde{m}_{jt})I[m_{jt} - \tilde{m}_{jt} > 0]} \\
\times \prod_{d=1}^{D} \prod_{j=1}^{k} \frac{\Gamma(n_{jd} + \alpha_j)}{\Gamma(n_{jd} + \alpha_j)} \theta_{jd}^{(n_{jd} - \tilde{n}_{jd})I[n_{jd} > \tilde{n}_{jd}]} \tag{4–21}
\]

\[
\geq s_2(z_x, \beta_x, \theta_x)
\]
The third minorization is thus given by
\[ g_2((z_x, \beta_x, \theta_x), (z_y, \beta_y, \theta_y)) \geq s_2(z_x, \beta_x, \theta_x) \times g_2((\bar{z}, \bar{\beta}, \bar{\theta}), (z_y, \beta_y, \theta_y))I[\beta_y, \theta_y \in D_2] \] (4–21)

Two problems present themselves immediately.

1. How do we specify, and choose, \(D_2\)?
2. How can we compute the infimum?

The two questions are closely inter-related, because of the form of the function we need to minimize. As finding the minimum of a function is equivalent to finding the minimum of the log of the function, our task can be reduced to finding the minimum of functions of a specific form. Note that \(\log(s(z_x, \beta_x, \theta_x))\) is a sum of functions to be minimized over the corresponding set in \(D_2\), and each function has the form
\[
f(\theta_1, \ldots, \theta_k) = \sum_{j=1}^{k} a_j \log \theta_j
\]
where \(0 < \theta_j < 1, \theta_k = 1 - \sum_{j=1}^{k-1} \theta_j\) and \(a_j \geq 0\). Reparametrizing, we have to minimize
\[
f(\theta_1, \ldots, \theta_{k-1}) = \sum_{j=1}^{k-1} a_j \log \theta_j + a_k \log(1 - \theta_1 - \cdots - \theta_{k-1})
\]
where \(0 < \theta_1 + \cdots + \theta_{k-1} < 1\). This function \(f\) is concave, since the Hessian matrix can be shown to be negative definite. An equivalent, and more concise argument, is to point out that the objective function is a multinomial density, which is log-concave since it is from the exponential family.

What this means is that we have to minimize a concave function over \(D_2\). This is a difficult problem, and an active research area. A review of the problem can be found in Horst et al. (2000). However, what we have working for us is that we get to choose the set \(D_2\). Since we know that the minimum of a concave function over a convex set can be found at one of its vertices, we can choose a set \(D_2\) such that we know all the vertices.
In order to outline our idea, we return to a simple form of our small function $s_2$.

Consider $\theta$ for one document, and suppose that $k = 3$. Our equation would look like this.

$$f(\theta_1, \theta_2) = \sum_{j=1}^{2} a_j \log \theta_j + a_3 \log(1 - \theta_1 - \theta_2)$$

The support of $(\theta_1, \theta_2)$ in $\mathbb{R}^2$ is the simplex given by the hyperplanes $\theta_1 = 0$, $\theta_2 = 0$ and $\theta_1 + \theta_2 = 1$. Our strategy to choose a distinguished set for the $\theta$-vector is to “shrink” the simplex towards where the points are clustered. We can do this by choosing $q_1, q_2$ and $q_3$ and setting the distinguished set to be those $(\theta_1, \theta_2)$ such that

$$\theta_1 \geq \hat{F}_{\theta_1}^{-1}(q_1)$$
$$\theta_2 \geq \hat{F}_{\theta_2}^{-1}(q_2)$$
$$\theta_2 + \theta_1 \leq \hat{F}_{\theta_1+\theta_2}^{-1}(q_3)$$

where we have to impose the condition $q_3 > q_1 + q_2$.

The inequalities define the interior of a simplex given by the set $\{w \in \mathbb{R}^2 : Aw \leq c\}$, where

$$A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{pmatrix} \quad c = \begin{pmatrix} \hat{F}_{\theta_1}^{-1}(q_1) \\ \hat{F}_{\theta_2}^{-1}(q_2) \\ \hat{F}_{\theta_1+\theta_2}^{-1}(q_3) \end{pmatrix}$$

This set is convex, and has exactly 3 vertices.

In general, the approach will be to determine matrices for each of the probability vectors we are sampling, leading to

$$D_2 = \{\theta_1, \ldots, \theta_d, \beta_1, \ldots, \beta_k : A_{\theta_1} \leq c_{\theta_1}, \ldots, A_{\theta_d} \leq c_{\theta_d}, \ldots A_{\beta_k} \leq c_{\beta_k}\}$$

Thus every time we generate a new set of $\theta$ and $\beta$, we carry out $k + d$ matrix multiplications to see if that new set of $\theta$’s and $\beta$’s have fallen in the distinguished set. To find the infimum, we have to evaluate each of the $k + d$ functions at their vertices. Notice that the number of vertices grows linearly with the number of words, and that if
we fall upon a $z$ such that the $n_{j,d}$ and $m_{jt}$ are equal to that of the distinguished set, we can regenerate with probability 1.

Unfortunately, even this is not a good minorization. As the vocabulary size $V$, or the number of words $n$ increases, the regeneration probabilities drop drastically to the order of $10^{-78}$ even with just 6 documents of 30 words each. The code to carry out an analysis has been included in McParre, through the functions `regenProbsTopicModel` and `genNextStateTopicModel` and a demonstration script in dem0/ directory.

Before we proceed with a small example, we wish to point out that the joint posterior distribution from the model defined in display $(4–11)$ is multi-modal in nature. To see this more explicitly, refer to the joint distribution in $(4–12)$. Observe that when the $\alpha_j$’s are equal, then any cyclic permutation of the $z_d$ labels and corresponding switching of the $\theta_d$ and $\beta_j$ values will result in the same value for the joint posterior $(4–12)$. It is clear to see this if we consider the following small case with $D = 2$, $k = 2$, $V = 3$, $n_1 = n_2 = 3$ and the 6 words being $w = (1, 2, 2, 3, 3, 2)$. Now take the following two points from the state-space:

1. $z_1 = (1, 1, 2), z_2 = (2, 2, 1), \theta_1 = (0.6, 0.4), \theta_2 = (0.3, 0.7), \beta_1 = (0.1, 0.2, 0.7)$ and $\beta_2 = (0.2, 0.5, 0.3)$

2. $z_1 = (2, 2, 1), z_2 = (1, 1, 2), \theta_1 = (0.3, 0.7), \theta_2 = (0.6, 0.4), \beta_1 = (0.2, 0.5, 0.3)$ and $\beta_2 = (0.1, 0.2, 0.7)$

The height of the joint posterior density for both of them will be exactly the same. In the case where the $\alpha_j$’s are not equal, then it is not clear if the result is a desirable one. The reason is that the topics are defined by the distribution over the words, but the prior information from the $\alpha$ vector enters through the $z$-labels. In the above test-case, say we name topic 1, given by the distribution $(0.1,0.2,0.7)$, as SOCCER. Suppose we can identify the other topic, $(0.2,0.5,0.3)$, as BASKETBALL. If $\alpha_1 = 10$ and $\alpha_2 = 1$, then in state 1, SOCCER is given a high chance of being assigned, whereas in state 2, BASKETBALL has the advantage.
We have conducted some simple simulations in order to test the above insight. The results can be viewed in Figure 4-4. The issue stems from the lack of identifiability of the cluster labelings. Hence for the example that follows, we shall sidestep this issue by assuming that, apart from the words themselves, we also observe the true topic assignments of some words.

As a small instructive example of the implementation of the LDA model in McParre, we generated data from the generating hierarchy (4–11) with the following corpus configuration: $K = 2$, $D = 2$, $V = 6$, $n_1 = n_2 = 30$. Instead of generating $\beta_j$ and $\theta_d$, we used the following values:

1. $\theta_1 = (0.05, 0.95)$,
2. $\theta_2 = (0.95, 0.05)$,
3. $\beta_1 = (0.43, 0.05, 0.05, 0.08, 0.07, 0.05, 0.05, 0.22)$
4. $\beta_2 = (0.02, 0.20, 0.41, 0.10, 0.11, 0.05, 0.05, 0.06)$

We also assume that we observed the true topic assignments for the first 5 words of each document. This has the effect of lowering the height of the other modes in the joint posterior. After an initial run of 250,000 iterations, we were able to choose the distinguished set for $\theta$ to be

$$A_{\theta_1} = A_{\theta_2} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad c_{\theta_1} = \begin{pmatrix} -0.024 \\ 0.110 \end{pmatrix} \quad \text{and} \quad c_{\theta_2} = \begin{pmatrix} -0.880 \\ 0.974 \end{pmatrix}$$

The way to interpret this is that the generated $\theta_1 = (\theta_{11}, \theta_{12})$ will be deemed to have fallen into the distinguished set if $A_{\theta_1} \theta_{11} \leq c_{\theta_1}$, or equivalently, $0.034 \leq \theta_{11} \leq 0.178$. Correspondingly, the generated $\theta_2$ has fallen in the distinguished set if it satisfies
$A_{\theta_2 \theta_1} \leq c_{\theta_2}$. The corresponding distinguished sets for $\beta$ can be represented as

$$A_{\beta_1} = A_{\beta_2} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad c_{\beta_1} = \begin{pmatrix} -0.366 \\ -0.049 \\ -0.015 \\ -0.053 \\ -0.055 \\ -0.032 \\ 0.878 \end{pmatrix}, \quad c_{\beta_2} = \begin{pmatrix} -0.025 \\ -0.147 \\ -0.336 \\ -0.053 \\ -0.098 \\ -0.015 \\ 0.970 \end{pmatrix}$$

We used McParre to generate 30 tours on 1, 2 and 5 processors. The timings for the simulation runs are available in Table 4-9. The computed Confidence Intervals from just these 30 tours are given in Table 4-10.

### 4.4 Summary

We have provided a package for general use in running regenerative Markov chains in parallel computers. We have demonstrated that it provides an almost linear speed-up in some cases. Our experience also shows that the minorization is still a crucial part of the analysis. Our attempts at minorizing the LDA model have some scope for further work, which we have outlined in Chapter 6.

**Table 4-1.** Numerically computed posterior means for test dataset

| $E(\lambda | y)$ | $E(\theta_1 | y)$ | $E(\theta_2 | y)$ |
|-----------------|-----------------|-----------------|
| 0.1361          | 0.5208          | -3.9258         |

**Table 4-2.** Coverage of 85% and 90% confidence intervals as the number of tours collected increases.

<table>
<thead>
<tr>
<th>Completed tours</th>
<th>85% CI coverage</th>
<th>90% CI coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_\theta$</td>
<td>$\theta_1$</td>
<td>$\theta_2$</td>
</tr>
<tr>
<td>5000</td>
<td>0.77</td>
<td>0.82</td>
</tr>
<tr>
<td>25000</td>
<td>0.83</td>
<td>0.86</td>
</tr>
<tr>
<td>50000</td>
<td>0.80</td>
<td>0.86</td>
</tr>
</tbody>
</table>
Table 4-3. Time taken (in seconds) to run 50,000 tours on 1, 2 and 5 processors. 10 chains were run with each configuration of processors.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Mean time</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 processor</td>
<td>119.127</td>
<td>1.771</td>
</tr>
<tr>
<td>2 processors</td>
<td>53.299</td>
<td>1.421</td>
</tr>
<tr>
<td>5 processors</td>
<td>32.484</td>
<td>0.333</td>
</tr>
</tbody>
</table>

Table 4-4. REML estimates for petrol data model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>19.707</td>
<td>0.568</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.219</td>
<td>0.147</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>0.546</td>
<td>0.521</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>-0.154</td>
<td>0.040</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td>0.157</td>
<td>0.006</td>
</tr>
<tr>
<td>$\sigma^2$</td>
<td>3.510</td>
<td>-</td>
</tr>
<tr>
<td>$\sigma^2_1$</td>
<td>2.090</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4-5. ML estimates for petrol data model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>19.694</td>
<td>0.478</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.221</td>
<td>0.123</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>0.549</td>
<td>0.441</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>-0.153</td>
<td>0.034</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td>0.156</td>
<td>0.006</td>
</tr>
<tr>
<td>$\sigma^2$</td>
<td>3.340</td>
<td>-</td>
</tr>
<tr>
<td>$\sigma^2_1$</td>
<td>0.860</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4-6. 95% Confidence Intervals from 10,000 tours of regenerative Markov chain estimates for petrol data model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Lower bound</th>
<th>Point estimate</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_0$</td>
<td>19.711</td>
<td>19.714</td>
<td>19.718</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.217</td>
<td>0.218</td>
<td>0.219</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>0.537</td>
<td>0.541</td>
<td>0.544</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>-0.157</td>
<td>-0.157</td>
<td>-0.156</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td>0.158</td>
<td>0.158</td>
<td>0.158</td>
</tr>
<tr>
<td>$\lambda^R_{-1}$</td>
<td>2.754</td>
<td>2.756</td>
<td>2.758</td>
</tr>
<tr>
<td>$\lambda^D_{-1}$</td>
<td>0.521</td>
<td>0.523</td>
<td>0.525</td>
</tr>
</tbody>
</table>
Table 4-7. Time taken to run 100000 tours for the Petrol data on 1, 2 and 5 processors. The time unit is minutes, and the standard deviations are obtained from 20 runs of each configuration of processors.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Mean time</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 processor</td>
<td>15.474</td>
<td>0.605</td>
</tr>
<tr>
<td>2 processors</td>
<td>7.804</td>
<td>0.483</td>
</tr>
<tr>
<td>5 processors</td>
<td>4.608</td>
<td>0.145</td>
</tr>
</tbody>
</table>

Table 4-8. Index notation for Latent Dirichlet Allocation model

<table>
<thead>
<tr>
<th>Model component</th>
<th>Index and range</th>
</tr>
</thead>
<tbody>
<tr>
<td>D documents</td>
<td>( d = 1, \ldots, D )</td>
</tr>
<tr>
<td>( n_d ) words in each document</td>
<td>( i = 1, \ldots, n_d )</td>
</tr>
<tr>
<td>k topics</td>
<td>( j = 1, \ldots, k )</td>
</tr>
<tr>
<td>V unique words in corpus</td>
<td>( t = 1, \ldots, V )</td>
</tr>
</tbody>
</table>

Table 4-9. Time taken (in minutes) to run 30 tours for the simulated data from the Topic Model on 1, 2 and 5 processors.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Mean time</th>
<th>Standard error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 processor</td>
<td>8.094</td>
<td>1.392</td>
</tr>
<tr>
<td>2 processors</td>
<td>4.262</td>
<td>0.978</td>
</tr>
<tr>
<td>5 processors</td>
<td>2.397</td>
<td>0.977</td>
</tr>
</tbody>
</table>

Figure 4-1. Although the coefficient of variation is already below the threshold specified by Mykland et al. (1995) after just 5000 tours, Figure 4-3 indicates that it is insufficiently close to the theorised Normal distribution.
Table 4-10. 95% Confidence Intervals from 30 tours of regenerative Markov chain estimates for simulated data from the Topic Model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Lower bound</th>
<th>Point estimate</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_{11}$</td>
<td>0.0779</td>
<td>0.0781</td>
<td>0.0783</td>
</tr>
<tr>
<td>$\theta_{12}$</td>
<td>0.9217</td>
<td>0.9219</td>
<td>0.9221</td>
</tr>
<tr>
<td>$\theta_{21}$</td>
<td>0.9280</td>
<td>0.9282</td>
<td>0.9283</td>
</tr>
<tr>
<td>$\theta_{22}$</td>
<td>0.0717</td>
<td>0.0718</td>
<td>0.0720</td>
</tr>
<tr>
<td>$\beta_{11}$</td>
<td>0.4093</td>
<td>0.4094</td>
<td>0.4096</td>
</tr>
<tr>
<td>$\beta_{12}$</td>
<td>0.0777</td>
<td>0.0778</td>
<td>0.0778</td>
</tr>
<tr>
<td>$\beta_{13}$</td>
<td>0.0345</td>
<td>0.0346</td>
<td>0.0346</td>
</tr>
<tr>
<td>$\beta_{14}$</td>
<td>0.0814</td>
<td>0.0815</td>
<td>0.0816</td>
</tr>
<tr>
<td>$\beta_{15}$</td>
<td>0.0825</td>
<td>0.0825</td>
<td>0.0826</td>
</tr>
<tr>
<td>$\beta_{16}$</td>
<td>0.0549</td>
<td>0.0550</td>
<td>0.0550</td>
</tr>
<tr>
<td>$\beta_{17}$</td>
<td>0.1023</td>
<td>0.1024</td>
<td>0.1024</td>
</tr>
<tr>
<td>$\beta_{18}$</td>
<td>0.1568</td>
<td>0.1569</td>
<td>0.1570</td>
</tr>
<tr>
<td>$\beta_{21}$</td>
<td>0.0495</td>
<td>0.0495</td>
<td>0.0496</td>
</tr>
<tr>
<td>$\beta_{22}$</td>
<td>0.1844</td>
<td>0.1844</td>
<td>0.1845</td>
</tr>
<tr>
<td>$\beta_{23}$</td>
<td>0.3789</td>
<td>0.3790</td>
<td>0.3791</td>
</tr>
<tr>
<td>$\beta_{24}$</td>
<td>0.0813</td>
<td>0.0814</td>
<td>0.0815</td>
</tr>
<tr>
<td>$\beta_{25}$</td>
<td>0.1304</td>
<td>0.1305</td>
<td>0.1305</td>
</tr>
<tr>
<td>$\beta_{26}$</td>
<td>0.0341</td>
<td>0.0341</td>
<td>0.0341</td>
</tr>
<tr>
<td>$\beta_{27}$</td>
<td>0.0858</td>
<td>0.0858</td>
<td>0.0859</td>
</tr>
<tr>
<td>$\beta_{28}$</td>
<td>0.0552</td>
<td>0.0552</td>
<td>0.0553</td>
</tr>
</tbody>
</table>

Figure 4-2. The above plots are for one of the Markov chain runs. A random sample of 100 of the points to be plotted was taken after 5K, 10K and 50K tours. The above plots do not look very different, suggesting they might not be useful as a diagnostic.
Figure 4-3. All $100 \times 50000 = 5000000$ tours were used to compute an estimate of $\sigma^2$, the asymptotic variance in the regenerative CLT. Then taking into consideration only the first 25K tours, the density plot from the $\bar{X}$ of the 100 chains gives the solid red lines in the top 2 panels. The solid cyan line comes from using all 50000 tours in each of the 100 chains. The dashed lines depict the theoretical Normal distributions that the $\bar{X}$ should be close to.
Figure 4-4. Data was generated from the LDA model with $V = 3$, $D = 2$, $K = 2$, $n_1 = n_2 = 20$ and the true topic mixture probabilities $\theta_1 = (0.05, 0.95)$ and $\theta_2 = (0.95, 0.05)$. Thus the two documents should separate more or less cleanly into 2 topics. The Gibbs sampler was run for 275,000 iterations. The top left-hand plot depicts the sampled values of $\theta_{1,1}$ for the first 50,000 iterations. The bottom left diagram summarises the topic assignments of words to topics for these initial 50,000 iterations. Configuration 1 denotes a state where most of the words in document 1 were assigned label 2, and most of the words in document 2 were assigned label 1. Configuration 3 denotes the converse, and configuration 2 refers to a state where either document 1 or 2 had an equal assignment of words to topics. The histograms on the right clearly show the multimodal nature of the posterior distribution. The ergodic means for all the $\theta$ parameters were estimated to be 0.50.
CHAPTER 5
PIPELINED ADAPTIVE MARKOV CHAIN MONTE CARLO

5.1 Introduction

Adaptive MCMC is a comparatively new sub-field of Markov chain theory. Two useful reviews of the topic can be found in Andrieu and Thoms (2008) and Andrieu and Robert (2001). In this subsection, we introduce the idea before focusing on a technique to parallelise the running of an adaptive MCMC chain.

Markov chain Monte Carlo algorithms form a large class of algorithms that allow a user to generate approximate samples from a target distribution $\pi$, when it is infeasible to sample directly from it. When deciding on the particular algorithm to run however, the user is typically faced with the problem of deciding on a candidate distribution to generate from. In the case of a Metropolis-Hastings algorithm, the more similar this candidate is to $\pi$, the closer the final random variables will be to $\pi$. Hence there is some motivation indeed, to choose a “good” candidate.

The circular nature of the problem is that a lot of times, a user does not know what the key features of $\pi$ are. In fact, he might intend to use the obtained approximate sample to estimate these features, e.g. an integral with respect to $\pi$. In such situations, when the user has little prior knowledge about $\pi$, how is he expected to choose the right candidate?

Typically, the user is deciding on candidates from within a family of distributions parametrised by $\theta \in \mathbb{R}^n$. In such cases, one solution is to use an adaptive MCMC technique. With this, the user first specifies a cost function $h(\theta)$ that represents a criterion to be optimised. There are several criteria that a user can pick. For example, if the aim is to estimate an integral $\int f d\pi$, and it is known that the sample mean satisfies a CLT, then one criteria could be the asymptotic variance of the CLT. Note that this criterion, however, depends on the function $f$, and will be hard to even estimate for each value of $\theta$. An alternative criterion is the optimal acceptance rate, which is known in
some specific cases. Then the goal would be to run the chain with the $\theta$ that gives rise to this optimal acceptance rate. Many more optimality criteria are covered in Section 4.1 of Andrieu and Thoms (2008).

Once an optimality criterion is chosen, the chain is started with any pre-determined $\theta$. Every few steps of the chain, the value of $\theta$ is updated according to a Stochastic Approximation (SA) algorithm. This algorithm will, under certain conditions, direct the sequence of $\theta$ values to the one that optimises $h$. We give more details on when, how and why the SA algorithm will converge to the right value, in the subsections below. Thus eventually, the user will be running the optimal chain.

Adaptive MCMC solutions provide an alternative to running a range of pilot chains, each with a different value of $\theta$, and then picking the best one. In most of the current literature, authors warn that the sequence of $\theta$ values have to change less and less in order to preserve stationarity of the Markov chain. They refer to this as Vanishing Adaptation - see Roberts and Rosenthal (2007) and Andrieu and Thoms (2008) for more details. We feel that it is best instead to use the adaptive technique to whittle down the possible values of $\theta$ in an automated manner. When the SA algorithm has converged, the value of $\theta$ should be fixed, and the chain restarted. In this way, we will have a true Markov chain with the optimal $\theta$ from then on.

5.2 Stochastic Approximation

Stochastic Approximation (SA) algorithms are applicable in slightly different forms in many different scenarios. The convergence of these algorithms can be proved using quite general methods. The purpose of this subsection is to provide an overview of SA algorithms and their proofs of convergence. The results presented in this section are from Chen (2002) and Kushner and Yin (1997).

Loosely speaking, SA is an optimization technique that is applicable when the optimization problem can be reduced to finding the zeroes (roots) of an unknown function. By “unknown”, we mean that the function can be observed, but
these observations may be corrupted by errors. In cases where we can observe the function and its gradient without error, there are recursive algorithms such as the Newton-Raphson that will work well.

SA was first introduced by Robbins and Monro in Robbins and Monro (1951). It is a simple, recursive algorithm, and can be described thus: Let \( g : \mathbb{R}^n \rightarrow \mathbb{R}^n \) be an unknown, nondecreasing function with a root at \( \theta^* \), i.e. \( g(\theta^*) = 0 \). Assume \( g(\cdot) \) can be observed at each point \( \theta_k \) with a following noise term:

\[
Y_n = g(\theta_n) + \epsilon_n
\]

Explicitly, at time \( n \), \( \theta_n \) is our estimate of \( \theta^* \), \( \epsilon_n \) is the (unobserved) noise, and \( Y_n \) is our observation or measurement of the function \( g(\cdot) \). According to the Robbins-Monro algorithm, the estimate of \( \theta^* \) at iteration \( n + 1 \) should be

\[
\theta_{n+1} = \theta_n - a_n Y_n
\]

They proved that as long as the step sizes \( \{a_n\} \) satisfy the following conditions: \( a_n > 0 \), \( a_n \rightarrow 0 \) and \( \sum a_n = \infty \), \( \theta_n \rightarrow \theta^* \) in \( L^2 \). Since 1951, much research has gone into weakening the conditions on \( g \) and the error terms while retaining convergence of the algorithm. For example, observations of \( g(\cdot) \) could even contain a structural error:

\[
Y_n = g_n(\theta_n) + \epsilon_n = g(\theta_n) + \epsilon_n + (g_n(\theta_n) - g(\theta_n))
\]

Note that there is now an additional error term arising from a measurement on \( g_n(\cdot) \) instead of \( g(\cdot) \) at every iteration. As long the observation errors can be shown to average out to zero, the algorithm will still converge.

Current proofs target almost sure convergence of the iterates to the root of the target function. There are 3 main methods. The first method requires the noise term to be a martingale difference. However, it also requires a linear growth condition on \( g(\cdot) \). A more general technique was invented in the 1970s that relied on Ordinary Differential
Equations (ODEs). The idea of the method is this: The iterates $\theta_n$ are interpolated to form a piecewise continuous function, whose tail part is shown to satisfy the ODE

$$\dot{x} = g(x)$$

The sought-after root $\theta^*$ is an equilibrium point of this ODE. From here, it can be shown that, based on the stability of the ODE, the iterates $\theta_n \to \theta^*$ a.s. This method of proof does away with the linear growth condition, but requires that the iterates be bounded. If they are not known a priori to be bounded, then modifications to this ODE method can be used to prove convergence. One modification involves proving that the iterates enter a compact set infinitely often. Another modification involves assuming that the compact set expands as more and more iterates are obtained, essentially allowing the iterates to be unbounded. The latter methods of proof involve decreasing step sizes, and allow for almost sure convergence. In some cases, the step size cannot be decreased (for physical reasons) and has to be kept constant. For these cases, weak convergence of the iterates can be proved. This is the third main technique of proof in SA.

It is apparent that there are many flavors to SA. For full details on the methods and many example applications, the reader is referred to Chen (2002) and Kushner and Yin (1997). In the sections below, we shall introduce one of the simpler algorithms and outline the proof of its convergence. Then we shall introduce a parallel version of it, demonstrate that it converges if the parent one does, and show that it has the same convergence rate as the original.

To tie things back to adaptive MCMC, consider a running example for the rest of this chapter:

**Example 5.2.1** (Optimal Scale Parameter). This example is from Gelman et al. (1996), which consists of simulations that demonstrate applications of the more general theorems proved in Roberts et al. (1997). In the former paper, the authors consider simulating from a multivariate standard normal distribution of dimension $d$, using
a normal distribution with variance matrix $\sigma_d^2 I$ as the proposal in a Random Walk Metropolis-Hastings algorithm. Their aim was to determine the optimal $\sigma_d^2$ to use. In the univariate case, optimality was determined by the asymptotic relative efficiency of the estimator, as compared to i.i.d sampling from the target distribution. In other words, if $\bar{X}$ is an estimate from $N$ i.i.d samples and $\bar{Y}$ is an estimate from the Markov Chain of length $N$, the efficiency when using a proposal with variance $\sigma^2$ is given by:

$$
\text{eff}_{\sigma_d} = \frac{\lim_{N \to \infty} N \text{Var}(\bar{X})}{\lim_{N \to \infty} N \text{Var}(\bar{Y})} = \frac{1}{\lim_{N \to \infty} N \text{Var}(\bar{Y})}
$$

The authors have an asymptotic result (for more general target distributions) that leads to the following heuristic rule when utilising a normal distribution as the candidate in a RW Metropolis-Hastings algorithm: Set the scale of the candidate $\sigma_d$ such that the acceptance rate of the algorithm is approximately $1/4$. This is an asymptotic result, that we shall not use.

Instead, we focus on the lower-dimensional results in Section 3.2 of their paper. There, they find that when $d = 1$, the optimal $\sigma_d$ is 2.4. The relationship between the scale of the proposal and the acceptance rate of the RW Metropolis-Hastings algorithm is given by:

$$
p_{\text{accept}} = \frac{2}{\pi} \arctan \left( \frac{2}{\sigma_d} \right)
$$

For the purpose of our example, we shall pretend that we are unaware of the optimal scale value, but that we do know that it is the one which yields an acceptance rate of

$$
p_{\text{opt}} = \frac{2}{\pi} \arctan \left( \frac{2}{2.4} \right) = 0.442
$$

This is in tune with the heuristic recommendation given in their paper: tune the parameter of interest until the acceptance rate is approximately $1/4$. Here, our parameter of interest would be $\sigma_d$.

Dropping the $d$ from our notation, our adaptive MCMC chain would then involve starting at $\sigma_0$, running the chain for, say 10000 steps, and then estimating the acceptance
rate based on this short chain. This forms the noisy estimate $Y_0 = (\hat{\alpha}(\sigma_0) - 0.442) + \epsilon_0$. Based on this noisy observation, we would update $\sigma_0$ according to the following rule for $n = 0, 1, ...$

$$\sigma_{n+1} = \sigma_n + a_n Y_n$$

Using this SA algorithm, we should expect to eventually get to the value 2.4 in the univariate case. It is possible to extend this example to higher dimension examples, based on the table in Gelman et al. (1996) that provides the optimal $\sigma_d$ and the corresponding optimal acceptance rate for $d = 1, 2, ..., 10$.

### 5.3 The Ordinary Differential Equation Method of Proof

In this subsection we consider an SA algorithm of moderate complexity and outline the ODE method of proof. Suppose we wish to estimate the root $\theta^*$ of a function $g : \mathbb{R} \to \mathbb{R}$. The algorithm can be written as

$$\theta_{n+1} = (\theta_n + a_n Y_n) \mathbb{I}\{\theta_n + a_n Y_n \in H\} + \theta^H \mathbb{I}\{\theta_n + a_n Y_n \notin H\}$$

$$= \Pi_H[\theta_n + a_n Y_n]$$

(5–1)

where $H$ is a compact set in $\mathbb{R}$. The symbol $\Pi_H$ denotes the projection onto $H$. It simply means that if at any iteration, $\theta_n + a_n Y_n$ falls outside $H$, then $\theta_{n+1}$ is set to be the point on the boundary of $H$ that is closest to $\theta_n + a_n Y_n$. This closest point is what we denote by $\theta^H$. Remember that in the context of the running example outlined in the previous subsection, $Y_n$ would be an estimate of $\alpha(\theta_n) - \alpha^*$ - the difference between the acceptance rate of the Metropolis-Hastings algorithm when using the candidate parametrised by $\theta_n$, and the known optimal acceptance rate.

Note that in reality, running this version of a SA algorithm requires knowledge of where the true $\theta^*$ lies. As mentioned earlier, there are SA algorithms that do not require this knowledge. However, we stick with this for now. We can rewrite the above algorithm as

$$\theta_{n+1} = \theta_n + a_n Y_n + a_n Z_n$$

(5–2)
where \( a_n Z_n = \theta_{n+1} - \theta_n - a_n Y_n \), the shortest vector needed to take \( \theta_n + a_n Y_n \) back to \( H \) if it is not already there. We shall need the following assumptions on \( g \), the random variables and the sequences:

1. \( \sup_n \mathbb{E}|Y_n|^2 < \infty \)
2. \( \mathbb{E}[Y_n|\theta_0, Y_i, i < n] = g(\theta_n) + \beta_n \)
3. \( g \) continuous, \( g(\theta^*) = 0, \theta^* \in H \)
4. \( \sum a_n^2 < \infty, a_n > 0, a_n \rightarrow 0 \) and \( \sum a_n = \infty \)
5. \( \sum a_n|\beta_n| < \infty \) w.p. 1

Observe that a sequence of random variables \( \{\beta_n\} \) has been introduced. This sequence is in addition to the error terms \( \{\epsilon_n\} \), but is “asymptotically negligible” in the manner prescribed by assumption (5). In our running example, we can think of it as a bias that arises because our chain is not starting in stationarity. As the chain length increases, this will go to 0.

First, define \( \delta M_n = Y_n - Z_n - g(\theta_n) - \beta_n \), and decompose the algorithm given by equation (5–2) to

\[
\frac{\theta_{n+1}}{\theta_n} = \frac{\theta_n + a_n Z_n + a_n g(\theta_n) + a_n \beta_n + a_n \delta M_n}{\theta_n + a_n Z_n + a_n g(\theta_n) + a_n \beta_n + a_n \delta M_n}
\]

Next, we define the time intervals via the step-sizes. In other words, let \( t_0 = 0 \) and \( t_n = \sum_{i=0}^{n-1} a_i \). Now we define the continuous time interpolation of iterates:

\[
\theta^0(t) = \theta_n \text{ for } t_n \leq t < t_{n+1}
\]

The shifted process is given by

\[
\theta^n(t) = \theta^0(t + t_n) \text{ for } t \geq 0
\]

The whole idea behind the ODE method is to show that as \( n \) goes to infinity, the shifted process converges to a trajectory of the solution to the ODE \( \dot{\theta} = g(\theta) \). Lastly we define \( m(t) \) as the unique value of \( n \) such that \( t_n \leq t < t_{n+1} \). Then we can write the shifted
process in this way:

$$
\theta^n(t) = \theta_n + \sum_{i=n}^{m(t+\tau_n)-1} a_ig(\theta_i) + \sum_{i=n}^{m(t+\tau_n)-1} a_iZ_i + \sum_{i=n}^{m(t+\tau_n)-1} a_i\delta_iM_i + \sum_{i=n}^{m(t+\tau_n)-1} a_i\beta_i
$$

(5–6)

$$
= \theta_n + \int_0^t g(\theta^n(s))ds + Z^n(t) + M^n(t) + \beta^n(t) + \rho(t)
$$

where the $Z^n(\cdot), M^n(\cdot)$ and $\beta^n(\cdot)$ are defined through the discrete sums in the first line.

The discrete summation involving $g(\cdot)$ is re-written as $\int g(\theta^n(s))ds + \rho(t)$. The initial part of the proof is to show that $M^n(t), \beta^n(t)$ and $\rho(t)$ all go to 0 as $n$ goes to infinity. The key step is to then show that each term on the right-hand side that is a function of $t$, when considered as a sequence in $n$, is equicontinuous. Applying the Arzela-Ascoli theorem, we can show that almost surely, there is a subsequence $\{\theta^n(\cdot)\}$ that converges to a solution of $\dot{\theta} = g(\theta)$. From this point, it can be argued that the limit of the entire sequence of $\{\theta^n(\cdot)\}$ must lie in an invariant set of the mean ODE. The full details of the proof for this SA algorithm can be found in Section 5.2 of Kushner and Yin (1997).

We emphasise however, the proof begins with writing $\theta^n(t)$ as in equations (5–6) and proceeding from there. Here is a complete statement of the convergence result.

**Theorem 5.3.1.** Let the assumptions 1-5 hold for the algorithm given in equation (5–1). Then there is a set $N$ of probability 0, such that for $\omega \notin N$, the set of functions $\{(\theta^n(\omega, \cdot), Z^n(\omega, \cdot))\}$ is equicontinuous. Let $(\theta(\omega, \cdot), Z(\omega, \cdot))$ be the limit of some convergent subsequence. Then this pair satisfies the projected ODE given by:

$$
\dot{\theta} = g(\theta) + z
$$

(5–7)

where $z(\cdot)$ is the minimum force required to keep the solution in $H$. Also, $\{\theta_n(\omega)\}$ converges to some limit set of the ODE in $H$. If the constraint set is dropped, but $\{\theta_n\}$ is bounded with probability 1, then for almost all $\omega$, the limits $\theta(\omega, \cdot)$ of convergent subsequences of $\{\theta^n(\omega, \cdot)\}$ are trajectories of $\dot{\theta} = g(\theta)$ in some bounded invariant set and $\{\theta_n(\omega)\}$ converges to this invariant set.
5.4 Pipeline Version of Stochastic Approximation

In a situation where we have only a single processor, the above algorithm could spend a significant amount of time obtaining a single observation $Y_n$. Going back to our example of achieving an optimal acceptance rate in a Metropolis-Hastings algorithm, the single processor would have to fix $\theta_n$, and then run the chain for a number iterations in order to estimate the acceptance rate under $\theta_n$. Thus the clock time between updates of the SA algorithm could be unacceptably long.

When we have numerous processors at our disposal, we can pipeline these computations in the following manner. This idea of pipelining the SA algorithm can be found in Kushner and Yin (1997). Here, we apply it to adaptive Markov chains and formally investigate its properties. Suppose we have $d$ identical processors and that, in our acceptance rate example, we wish to fix $\theta_n$ and run each chain for $d - 1$ steps before updating the SA algorithm. Suppose each step of an individual Markov chain takes 1 unit of computation time, and denote the time scale in terms of computation time by $T$.

Initialise the SA algorithm with $\theta_0 = \theta_1 = \ldots = \theta_{d-1}$. Start the first chain on processor 1 at $T = 0$. After 1 unit of computation, at $T = 1$, start the second processor with $\theta_1$. One unit of time later, start processor 3 with $\theta_2$, and so on and so forth. At $T = d - 1$, processor 1 would have finished $d - 1$ iterations of the chain, and would have $Y_0$. It would receive $\theta_{d-1}$ from processor $d$, and perform the following update:

$$\theta_d = \theta_{d-1} + a_0 Y_0$$

At $T = d$, processor one would restart its chain with a new parameter $\cdot \theta_d$. At this very same time point, processor 2 would have obtained $Y_1$, and would be ready to update $\theta_1$. It would obtain $\theta_d$ from processor 1 and perform the following update:

$$\theta_{d+1} = \theta_d + a_1 Y_1$$
Thus after the first round of updates, when \( \theta_0, \ldots, \theta_{d-1} \) have all been updated, the SA algorithm would be updating every unit of computation time instead of every \( d \) units of computation time.

For a clearer, diagram-aided explanation, please refer to Figure 5-1. Consider the case where we have 3 processors at our disposal to estimate the optimal scale parameter in Example 5.2.1. Assume that each processor takes 2 units of time to run the chain with a fixed parameter. Assume that a further single unit of time is needed to do the following: (i) estimate the acceptance rate, (ii) receive a parameter from the predecessor in the pipeline, (iii) update the parameter to the new value, and (iv) send the new parameter value to the next processor. For each processor, steps (i) - (iv) are carried out in the red section in the above figure. The blue dashed arrows indicate a processor sending its most updated value of the parameter to the next processor in the pipeline. To begin, \( \theta_0, \theta_1 \) and \( \theta_2 \) are initialised to the same value and the chain on processor 1 is started. After 1 unit of time, processor 2 is started. After 2 time units, processor 3 sends \( \theta_2 \) to processor 1 and then starts its own chain. Processor 1 receives \( \theta_2 \), and together with its estimate of the acceptance rate \( \hat{\alpha}_0 \), performs the update \( \theta_3 = \theta_2 + a_0(\hat{\alpha}_0 - \alpha^*) \). After sending \( \theta_3 \) to processor 2, it begins estimation of the acceptance rate with \( \theta = \theta_3 \). The cycle then continues - processor 2 receives \( \theta_3 \), updates to \( \theta_4 \) using its estimate \( \hat{\alpha}_1 \), sends \( \theta_4 \) to processor 3 and starts its next round of acceptance rate estimation with \( \theta_4 \).

The modified form of the SA algorithm (from equation (5–1)) is now

\[
\theta_{n+1} = \Pi_H(\theta_n + a_{n-(d-1)}Y_{d-(d-1)}) \text{ for } n \geq d - 1
\]

We can now define analogous quantities to those in equations (5–3), (5–4) and (5–5). Let \( t_0 = 0, t_n = \sum_{i=0}^{n-1} a_i \). Keeping the same assumptions as in the original sequential SA
algorithm described in the previous subsection, we decompose the algorithm in (5–8) as

\[
\theta_{n+1} = \theta_n + a_{n-(d-1)}Z_{n-(d-1)} + a_{n-(d-1)}g(\theta_{n-(d-1)}) \\
+ a_{n-(d-1)}\beta_{n-(d-1)} + a_{n-(d-1)}\delta M_{n-(d-1)}
\]  

(5–9)

Lastly, the continuous time interpolation function becomes

\[
\theta^0(t) = \theta_{n+(d-1)} \text{ for } t_n \leq t < t_{n+1}
\]  

(5–10)

The shifted process, still defined as \( \theta^n(t) = \theta^0(t + t_n) \), can now be written as

\[
\theta^n(t) = \theta_{n+(d-1)} + \sum_{i=n}^{m(t+t_n)-1} a_i g(\theta_i) + \sum_{i=n}^{m(t+t_n)-1} a_i Z_i + \sum_{i=n}^{m(t+t_n)-1} a_i \delta M_i + \sum_{i=n}^{m(t+t_n)-1} a_i \beta_i 
\]  

(5–11)

which is exactly the same breakdown as in equation (5–6), except for a shifted starting
point. Thus the exact same proof can be used to prove convergence of the pipeline
algorithm, as long as the original algorithm convergences.

**Proposition 5.4.1.** Under the assumptions of Theorem 5.3.1, the pipeline version of the
SA algorithm (given formally by equation (5–8)) has the same convergence properties
as the original sequential version given by equation (5–1).

The pipelining case outlined above is most efficient when the number of time units
taken for a single processor to complete a round of computations and an update is
equal to the number of processors available. For example, in Figure 5-1, each of the
3 processors takes 2 time units to estimate the acceptance and 1 time unit to receive
and send messages, after which it is ready for the next run. Thus at every time unit, an
update is taking place.

In reality, it is almost impossible to achieve this perfect synchronization. Again in the
framework of the algorithm in Figure 5-1, it is possible that the message passing portion
of the algorithm is slow to such an extent that processor one finishes its computing
portion before processor 3 receives a value from processor 2. In such a situation, processor 1 would be left waiting for the message from processor 3 in order to proceed.

Alternatively, it might be the case that a cluster assigns us nodes of unequal speeds, in which case the message passing segment might be faster than the computation segment. As a result, there would be time units when no updates take place. However, even in such a case, all processors would still be working in parallel. This concurrent use of the processors is what leads to the faster clock time convergence of the pipelining algorithm.

### 5.5 Convergence Rate

In practical terms, we never truly run an algorithm till infinity. Typically we stop running the algorithm at some finite time and proceed with a decision based on the final value, or the final few iterations. It would be good, then, if we were able to determine the error in our final value. In this subsection we are concerned with this task - the asymptotic distribution of the normalised deviation from the true optimal parameter:

$$U_n = \frac{\theta_n - \theta^*}{\sqrt{a_n}}$$

Unfortunately, we require a stack of assumptions in order to obtain asymptotic normality of $U_n$.

1. $a_n = \frac{1}{n}$
2. The sequence of random variables $\{Y_nI[|\theta_n - \theta^*| \leq \rho]\}$ is uniformly integrable for some small $\rho$.
3. For $\theta^*$, an isolated stable equilibrium point of $\dot{\theta} = g(\theta)$ in $H$, the shifted process $\{\theta_n(\cdot)\}$ converges weakly to the constant process $\theta^*$.
4. The sequence of distributions given by $\{((\theta_n - \theta^*)/\sqrt{a_n})\}$ is tight.
5. $\mathbb{E}[Y_n|\theta_0, Y_i, i < n] = g_n(\theta_n)$, where $g_n(\cdot)$ is continuously differentiable for each $n$, and can be expanded as

$$g_n(\theta) = g_n(\theta^*) + g_n'(\theta^*)(\theta - \theta^*) + o(|\theta - \theta^|).$$
where \( \alpha(\cdot) \) is uniform in \( n \)

6. \[
\lim_{n,m} \frac{1}{\sqrt{m}} \sum_{j=n}^{n+mt-1} g_j(\theta^*) = 0
\]
where the limit is uniform on some bounded \( t \)-interval.

7. There is a Hurwitz matrix such that

\[
\lim_{n,m} \frac{1}{m} \sum_{j=n}^{n+m-1} [g'_n(\theta^*) - A] = 0
\]

8. For some \( p > 0 \) and small \( \rho > 0 \)

\[
\sup_n \mathbb{E} \left[ |\delta M_n|^{2+p} |\{|\theta_n - \theta^*| \leq \rho\} \right] < \infty
\]

Finally, there is a non-negative definite matrix \( \Sigma_1 \) such that

\[
\mathbb{E} \left[ \delta M_n \delta M_n^T |\{|\theta_n - \theta^*| \leq \rho\}|\theta_0, Y_i, \text{ for } i < n \right] \rightarrow \Sigma_1
\]

Now for some definitions. Let \( U^n(\cdot) \) denote the piecewise right continuous interpolation of \( U_n \), with interpolation intervals \( a_n \). Define \( W^n(\cdot) \) by

\[
W^n(t) = \begin{cases} 
\sum_{i=n}^{m(t_n+t)-1} \sqrt{a_n} \delta_i M_i & \text{if } t \geq 0 \\
-\sum_{i=m(t_n+t)}^{n-1} \sqrt{a_n} \delta_i M_i & \text{if } t < 0
\end{cases} \tag{5–12}
\]

The following theorem is from chapter 10 of Kushner and Yin (1997).

**Theorem 5.5.1.** Assume the above conditions hold for the sequential SA algorithm (with \( \theta \in \mathbb{R}^d \)) given in equation (5–1). Then the sequence \( \{(U^n(\cdot), W^n(\cdot))\} \) converges weakly in \( D'[0, \infty) \times D'[0, \infty) \) to a limit denoted by \( (U(\cdot), W(\cdot)) \), where \( W(\cdot) \) is a Wiener process with covariance matrix \( \Sigma_1 \) and \( U(\cdot) \) is stationary. Specifically, the limit is given by

\[
U(t) = \int_{-\infty}^{t} \exp (A + I/2)(t - s)dW(s)
\]

Informally, it means that \( U_n \sim N(0, S) \) as \( n \rightarrow \infty \), where \( S = \int_0^\infty e^{(A+I/2)t} \Sigma_1 e^{(A'+I'/2)t} dt \).
5.6 Simulations

As a comparison, the pipeline algorithm was run on Example 5.2.1 using a starting value of 2.30, and a target acceptance rate of 0.442. 5 processors were used, for 40 rounds of updates. This translates to 200 updates in total. The total time taken was 3:27 minutes and the final parameter value was 2.357. For the sequential version, 200 updates took 14:20 minutes and the final value was 2.349. This was a common theme - the pipeline version was in general 5 times faster, and was closer to the true optimal value at the end of the run.

5.7 Summary

From the statement of the assumptions and the theorem alone, it is clear that, if the original algorithm satisfies the assumptions, then so too would the pipeline version, and that it would have the same covariance for the limiting distribution. Thus the performance of the pipeline algorithm will not degrade in terms of the limiting variance. However, it will save greatly in terms of the clock time of the algorithm.
Figure 5-1. A diagram to explain the case where we have 3 processors at our disposal to estimate the optimal scale parameter in Example 5.2.1.
In Propositions 3.2.1 and 3.2.2, we provided a method for a user to utilise the complete tours in order to estimate the missing ones and showed that we need not be concerned with the inspection paradox when we do so. This work has some scope for further investigation. In particular, it would be interesting to see if imputing those final values would reduce the order of the bias \((1/t)\) as much as waiting for that final tour to finish. It is also of interest to see if, with typical values of \(t\) for Markov chains being so large, it is worth imputing at all. Possibly it might be, if we envision the parallel version running on hundreds of nodes, thus leading to several incomplete tours.

Closely related to this area is the issue of whether the regenerative CLT still holds true if the number of tours generated is random. This is a pertinent issue in the situation where we run a chain for a pre-specified number of iterations and consider the tours completed up to that point. It appears that this should hold true, at least in the asymptotic case, but should be formalized is possible.

In the LDA model, there are a couple of problems to tackle. Firstly, the multiple modes in the posterior have to be resolved somehow before a Markov chain can be run on it. Secondly, best minorization we have still leads to infrequent regenerations except in trivial cases. This is certainly an area for future research since the model is of great interest in the machine learning world. It feels like it should be possible to find a better minorization, especially since the chain itself converges very quickly. Another idea is how we choose the distinguished set. Again in the LDA model, we have to choose a simplex that is small so that the infimum taken is small, but that is large enough so that the chain visits it frequently enough for regular regeneration opportunities. In section 4.3, we have chosen the simplex in a very basic way. Some alternatives might be to use the eigenvector decomposition in order to pick up directions where most of the variation of the data is in. On choosing these vertices, we can transform this simplex into a set of
inequalities of the form $Ax \leq d$. If possible, we should avoid increasing the number of vertices in the simplex as the dimension of $\beta$ or $\theta$ increases though.

We could also consider how we can choose this simplex such that we direct the path of the chain to areas we wish to explore. This latter idea is a more general one, and should be investigated in the context of general Markov chains instead of the just the LDA model? Can we choose the distinguished set to direct the path of the Markov chain, and is there a benefit to doing so?

Another idea is the notion of running a stochastic search in parallel. Can the results in regeneration be applied to the case when a Markov chain is used in a stochastic search, in variable selection for example? One possible application is to the linear model in Gopal et al. (2011). The current version of the R package for BAMD already carries out the variable selection in parallel, but it does not capitalise on any of the regeneration results. Related to that model is the question of proving its geometric ergodicity and minorizing it in order to carry out estimation using McParre.
APPENDIX
R PACKAGE DOCUMENTATION

A.1 Overview

The R package, named McParre (Mc-Pear-Ree) has been written to allow a user to run a regenerative Markov chain on a cluster. All the user has to provide are the following basic functions:

1. The one-step generation function of the Markov chain. Given the current state, this function should return a draw from the Markov chain - the next state.

2. If the minorization is carried out using the distinguished point and set method, then there is almost always cancellation of normalization constants when computing the regeneration probability in equation (2–7). In this case it is not necessary to program separate functions for the small function, small measure and the transition density. It would be simpler to simply program a function that returns the regeneration probability given \( x \) and \( y \). On other occasions, the minorization might not allow for such a cancellation. Hence the user has two options here, and has to specify one of the following two sets of functions:

   (a) A function that returns the regeneration probability for the bell variable, given the current state and the next one.

   (b) Functions that compute the small function, small measure and the transition density for the chain.

A.2 Inner Workings of the Code

The package uses the message passing paradigm, as implemented according to the Message Passing Interface (MPI) standard that is documented in Gropp et al. (1998). This standard is the de facto standard for MIMD architectures, which many high performance computing centers use. The MPI libraries that are installed on these clusters are usually open-source implementations of the MPI standard in C or FORTRAN.

The way MPI works is simple - on a cluster, the same piece of code is run on all of them. During the initialization phase, a communication world is set up, that allows the individual nodes or processors to send and receive messages. In order to identify themselves, each node is assigned a rank during initialization. If there are \( p \) processors,
then the ranks will run from 0 to $p - 1$. The processor with rank 0 is typically referred to as the master, while the rest are referred to as slaves.

MPI code is typically split up into a series of if-else statements, interspersed with code segments that are run on all the processors. Each if-else statement checks to see what the rank of the processor it is running on is, and then performs the appropriate actions. Along with functions to send and receive messages, the MPI standard allows for collective calls. These require that all the processors execute them before the code proceeds. Examples of these would be broadcast calls, or synchronization points.

The Rmpi package, described in some detail in Mansmann et al. (2009), allows a user to call the MPI functions through R. However, a large focus is on allowing a user to run interactive jobs. This requires slaves to be “spawned” from a master, and doing this through R puts the slaves in an infinite loop, waiting for instructions from the master. This is not reliable, and I have encountered several problems. Even after extensive Googling and consulting with the administrators at the UFHPC, I could not fix some errors/warnings when using Rmpi in this mode.

My suggestion is to use R in batch mode when using the MPI routines. This would closely mimic the MPI standard. I have found this to be much more reliable. The user has to keep in mind that the same R code will now be executed on all the nodes and has to write his instructions accordingly.

### A.3 User Interface to Package Functions

Using the package is very straightforward. Functions that are meant to be run on parallel nodes will end with a capital P, while functions that run only on a single processor will end with an S. The main workhorse functions are `runMarkovChainRegenP()` and `runMarkovChainRegenS()`. The state of the Markov chain must be coded as a numeric vector, even if the random variable is categorical. This allows a general interface, and for the wrapper functions to be re-used for several chains.
Once the user has installed the MPI libraries, and the Rmpi package, he should check and confirm that the message-passing features of Rmpi are working properly. The flow-chart in Figure A-1 outlines the steps needed to run and analyse a regenerative Markov chain once this is done.

The `inst/` directory in the package folder contains several files that are necessary for running the package and for learning how to use McParre. The Rprofile file provided there should replace the one provided by Rmpi. The latter caters to those who wish to run the parallel R session interactively, and is responsible for keeping the slaves in an infinite loop. The McParre profile simply loads the Rmpi library and provides a clean up function for exiting. The `inst/` directory also includes sample scripts for running Hierarchical Linear Models and Topic Models.

Since we are running R in batch mode, and the same code is being executed on all nodes, they would all be writing to the same output file and hence overwriting each other. Hence the solution is to pipe all the R output to the null device, and get each node to write to it's own specific output file. This is exactly what `runMarkovChainRegenP()` does, in conjunction with the following MPI command to run the parallel R session:

```
mpirun -np 3 -machinefile $PBS_NODEFILE \
R --no-save -q CMD BATCH src/initializeOnewayP.R /dev/null
```

The log files are necessary because, when working with MPI, it is very easy to enter a deadlock - a situation when node(s) are waiting for a message that never arrives. The log files enable the user to trace where the communications broke down. Message passing is kept to a minimum in the package. The only messages passed are the output matrix of generated tours from slaves to the master at the end of the run.

The `demo/` directory contains functions that can be run using the `demo()` function in R once the package has been loaded. They give detailed examples on how to specify the arguments for the functions in McParre.
A.4 Interactive Sessions

Although we do not recommend using interactive sessions with Rmpi, we have included code that will allow a user to do so. We envision that this would be for testing purposes, because typically a Markov chain is run over a long period of time during which the user is not waiting at the console for the results to return. However, for short testing runs, the function runMarkovChainRegenIP will run the regenerative chains in a parallel interactive session. The user should remember to use the correct Rprofile file that is included in the inst/ directory.

Figure A-1. The above flow-chart outlines the functions involved at each step of running a regenerative Markov chain using McParre. Examples for the models described in chapter 4 are available in McParre with the suffixes OnewayPlain, HLMM and TopicModel.
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

Vikneswaran Gopal was born and raised in Singapore. The younger of two sons, he graduated from Raffles Junior College in 1995. After serving part of his National Service, he attended the National University of Singapore (NUS) and earned a B.Sc (Hons) degree in 2001, majoring in mathematics.

After returning to complete his army stint, he worked for the Centre for Strategic Infocomm Technologies. His role was to assess the security of cryptographic algorithms, protocols and implementations. While working there, he obtained a M.Sc. in statistics from NUS. That piqued his interest in the subject and eventually led to his pursuit of a PhD at the University of Florida Department of Statistics in 2006.

Upon completion of his Ph.D program, Vikneswaran will be joining the IBM Research Collaboratory in Singapore. He has been married to Cecilia Tan Chew Yin for 5 years.