Advances in MCMC Methods with Applications to Particle Filtering, DSMC, and Bayesian Networks

by

Dale Jennings

B.S., Western Washington University, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2016

This thesis entitled:
Advances in MCMC Methods with Applications to Particle Filtering, DSMC, and Bayesian Networks
written by Dale Jennings
has been approved for the Department of Applied Mathematics

Prof. Jem Corcoran

Prof. Manuel Lladser

Date

Jennings, Dale (Ph.D., Applied Mathematics)

Advances in MCMC Methods with Applications to Particle Filtering, DSMC, and Bayesian Networks

Thesis directed by Prof. Jem Corcoran

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms for sampling from a desired probability distribution. While there exist many algorithms that attempt to be somewhat universal, these algorithms can struggle for tractability in specific applications. The work in this dissertation is focused on improving MCMC methods in three application areas: Particle Filtering, Direct Simulation Monte Carlo, and Bayesian Networks. In particle filtering, the dimension of the target distribution grows as more data is obtained. As such, sequential sampling methods are necessary in order to have an efficient algorithm. In this thesis, we develop a "windowed" rejection sampling procedure to get more accurate algorithms while still preserving the necessary sequential structure. Direct Simulation Monte Carlo is a Monte Carlo algorithm for simulating rarefied gas flows. In this dissertation, we review the derivation of the Kac master equation model for 1-dimensional flows. From this, we show how the Poisson process can be exploited to construct a more accurate algorithm for simulating the Kac model. We then develop an ε-perfect proof of concept algorithm for the limiting velocity distribution as time goes to infinity. Bayesian Networks (BNs) are graphical models used to represent high dimensional probability distributions. There has been a great deal of interest in learning the structure of a BN from observed data. Here, we do so by walking through the space of graphs by modeling the appearance and disappearance of edges as a birth and death process. We give empirical evidence that this novel jump process approach exhibits better mixing properties than the commonly used Metropolis-Hastings algorithm.

Acknowledgements

First, I would like to thank my advisor, Prof. Jem Corcoran, not only for all of her guidance and support, but also for her enthusiasm and many stories.

Special thanks also goes out to Dr. Jim Curry for his continued interest, advice, and frequent hallway conversations.

To the rest of my committee, Prof. Manuel Lladser, Prof. Will Kleiber, Prof. Francois Meyer, and Dr. Doug Nychka, I thank them for their helpful input and encouragement.

My gratitude also goes out to Anne Dougherty for providing me with funding in the form of TA and GPTI positions.

Lastly, I would like to thank my family for their never ending moral support and encouragement. And to all of my friends, who I could always count on to provide the necessary distractions that have made my time in graduate school especially enjoyable.

Contents

Tables

Figures

Chapter 1

Introduction

In many inference problems, direct evaluations of quantities of interest are often intractable and therefore replaced with optimizing search strategies. In high-dimensional inference problems, such search strategies are often computationally infeasible and, in some cases, even impossible. There has been much research on both deterministic and probabilistic algorithms as a workaround to these computational challenges. One of the most popular classes of probabilistic algorithms is Markov chain Monte Carlo (MCMC) algorithms. The theoretical validation of MCMC algorithms began with the seminal paper of Metropolis et al [43], and was later generalized by Hastings [27]. Their power comes from their ability to sample from distributions that cannot be directly computed, often as a result of either intractable normalizing constants or the use of simulated likelihoods in place of analytic likelihoods. While many existing MCMC algorithms apply, in theory, to large classes of models, they typically require a fair amount "tweaking" in order to make them viable for real applications of interest.

The work in this thesis focuses on constructing more efficient and/or accurate algorithms for sampling in three areas. The first is **particle filtering and smoothing** which consists of sequential Monte Carlo methods that are often employed in order to perform inference about models that evolve in time and produce new observations that need to be processed in real time. Particle filters have a wide range of applications in areas such as as signal processing, robotic navigation, and computer vision. The second area of focus for this thesis is on simulation of the **Kac model**, which is a stochastic model of the dynamics for colliding particles of a homogeneous rarefied (dilute) gas. As a simplified version of the famed Boltzmann equation, the Kac model is often used in order to verify or discard conjectures related to the statistical behavior of thermodynamic systems. The third area we focus on in this thesis is that of **Bayesian network structure inference**. Bayesian networks are graphical representations of high-dimensional probability distributions. The NP-hard problem of inferring the structure of such a representation from data has received much attention in recent years and has wide reaching applications in areas as diverse as genetic regulatory networks, speech recognition, credit risk analysis, and the burgeoning area of "web intelligence" for personalized browsing recommendation.

In the remainder of this Chapter, we review some basic building blocks for the algorithms to come.

1.1 Markov Chain Monte Carlo

1.1.1 Rejection Sampling

One of the simplest Monte Carlo methods that can be employed is rejection sampling (also known as the "accept/reject" algorithm). Suppose that one wants to generate a realization of a random variable X from a probability distribution with density $\pi(x)$, where $\pi(x)$ may be expressed as

$$\pi(x) = \frac{h(x)}{Z}$$

for some, possibly unknown, normalizing constant Z. Suppose further that it is known how to generate a realization of a random variable Y from a probability distribution with density $g(y)$ such that

$$h(y) \leq Mg(y)$$

with M a known constant. The rejection sampling algorithm proceeds as follows.

(1) Draw $Y \sim g(\cdot)$.

(2) Draw $U \sim \text{unif}(0, 1)$.

(3) Check whether or not $U \leq \frac{h(Y)}{Mg(Y)}$.

- If it is true, accept $X = Y$ as a draw from $\pi(\cdot)$.
- Otherwise, reject Y and return to step (1).

This algorithm is known to eventually accept a draw and does so with unconditional probability $\frac{1}{M}$. Clearly, the optimal choice would be to let $g \equiv \pi$, but the goal is to draw from $\pi(x)$ when direct sampling is not possible. Thus, it is very important that the distribution $g(x)$ is chosen as close to $\pi(x)$ as possible to minimize M, while still being able to draw from $g(x)$.

While this algorithm is simple to employ, it can often be challenging to choose a density $g(\cdot)$ that results in an efficient algorithm, especially for a high-dimensional "target density" π.

1.1.2 Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is one of the most popular MCMC methods. Similar to rejection sampling, the goal is to draw/simulate a value (realization) of X where

$$X \sim \pi(x) \propto h(x).$$

Again, the normalization constant may be unknown. The Metropolis-Hastings algorithm is designed to construct a Markov chain, $\{X_n\}_{n=0}^{\infty}$, whose stationary distribution is $\pi(x)$. This chain is constructed by drawing "candidate" values X' from a density $q(x|X_{n-1})$. These candidate draws are then either accepted, in which case $X_n = X'$, or rejected, in which case $X_n = X_{n-1}$. The probability of acceptance of a point x', given that the chain is currently at point x, is given by

$$\alpha(x, x') = \min\left(1, \frac{h(x')q(x|x')}{h(x)q(x'|x)}\right).$$

This choice of acceptance guarantees that the chain satisfies detailed balance with respect to the target density π, which allows for the chain to have stationary distribution π. Formally, the Metropolis-Hastings algorithm is as follows.

(1) Let $n = 0$. Start with any initial state X_0.

(2) Generate a candidate next state $X' \sim q(x|X_n)$, and, independently, a uniform draw $U \sim \text{unif}(0,1)$.

(3) Check whether or not $U \leq \alpha(X, X')$.

- If it is true, set $X_{n+1} = X'$.
- Otherwise, set $X_{n+1} = X_n$.

(4) Set $n = n + 1$ and return to step (2).

In order to actually generate draws from π, this algorithm, in theory, needs to be run for an infinite amount of time. Typically, however, it is run for "a long time" in the hope that the generated chain of values has settled down to near equilibrium. Initial values are discarded up to a "burn-in time". The choice of such a time has been extensively studied, and we refer the reader to [14] for a review of some convergence diagnostics. Another issue is that the values of X_n are usually highly correlated with each other. This issue is minimalized by only selecting every m values from the chain $\{X_n\}_{n \geq b}$, for some smart choice of m.

1.2 Deficiencies in Naive Implementations

1.2.1 The Curse of Dimensionality

While it is theoretically possible to construct an efficient candidate density for use with rejection sampling or Metropolis-Hastings schemes, implementations that do not exploit the structure of the target distribution become extremely inefficient as the dimension of the state space grows.

To illustrate this, consider the case of using rejection sampling to create uniform draws from the n-dimensional unit ball $B_1^n(0)$. To do so, we will draw a point, $\mathbf{x}$, uniformly from the cube $[-1, 1]^n$. If $\mathbf{x}$ lies within $B_1^n(0)$, we accept it, if it lies outside the ball, we reject it and draw a new point. The probability of accepting the point $\mathbf{x}$ is then the ratio of the volume of the ball to the volume of the cube

$$\begin{aligned} \mathrm{P}(\text{acceptance}) &= \frac{V(B_n(1))}{V([-1,1]^n)} \\ &= \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)2^n} \end{aligned}$$

This scales poorly as the dimension, n, increases, as shown in Figure 1.1.

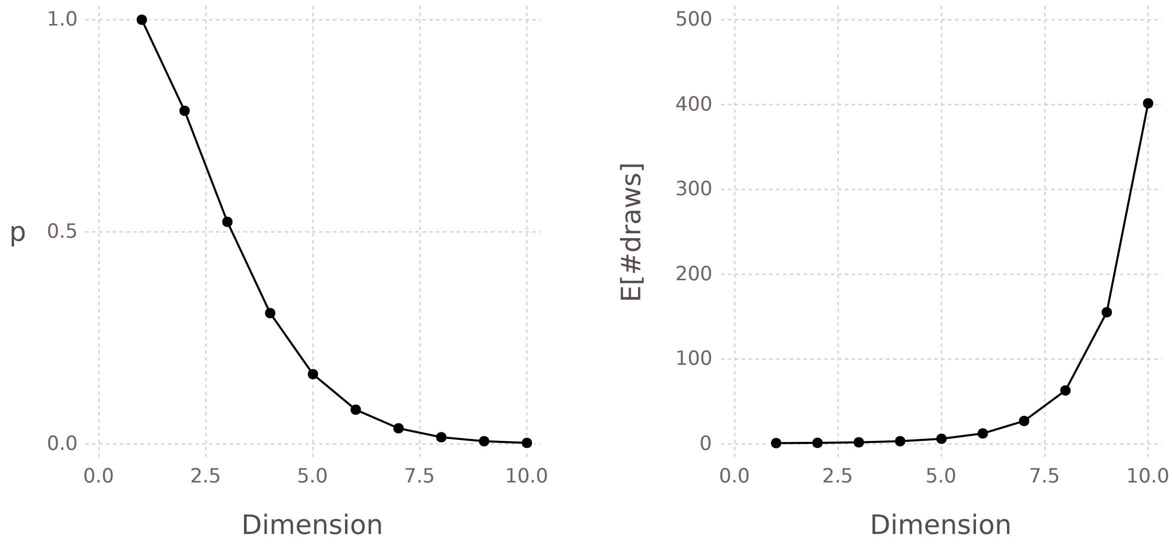


Figure 1.1: Rejection sampling using draws from the cube $[-1,1]^n$ to generate draws from the unit ball $B_1^n(0)$ as a function of the dimension, n. (Left) Proportion of accepted draws. (Right) Average number of required draws for 1 accepted draw

1.2.2 Distribution Bottlenecks

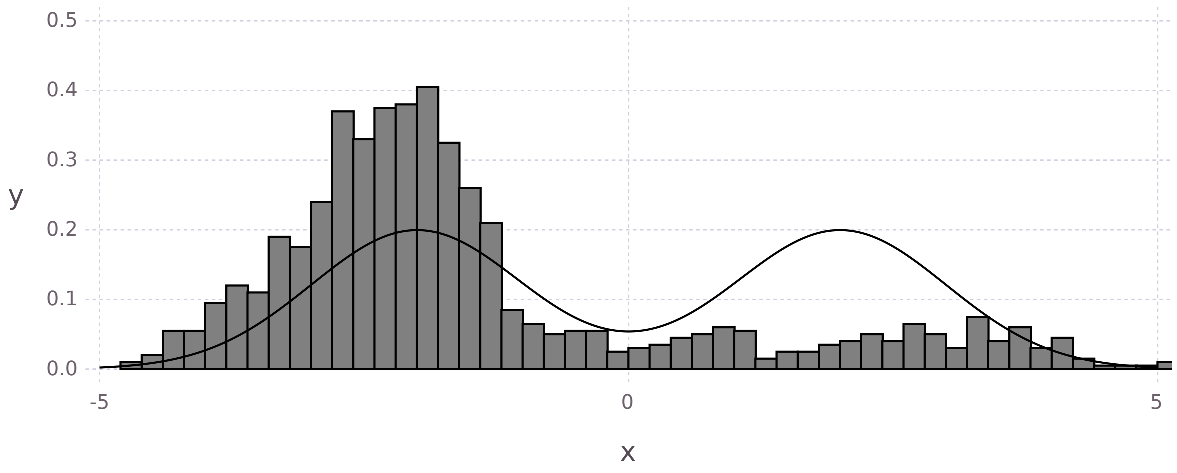
Another issue that arises in many MCMC schemes, including Metropolis-Hastings, is when there are “bottlenecks” in the target density. This is common in multi-modal densities where the density takes on small values in between the different modes. To illustrate how this affects mixing of the Markov chain, consider using Metropolis-Hastings to target the density

$$\pi(x) \propto e^{-\frac{1}{2}(x+2)^2} + e^{-\frac{1}{2}(x-2)^2}$$

using candidate densities of the form

$$q(x'|x) \sim \mathcal{N}(x'; x, \delta)$$

for different values of δ, where $\mathcal{N}(x; \mu, \sigma^2)$ denotes a Gaussian density in x with mean μ and variance σ^2. The first 1000 steps are discarded as a "burn-in", and then every sample is retained. (This is not ideal as previously discussed.) When too small of a δ value is chosen, the chain mixes well in the individual modes, but has a tendency to get stuck in one of the modes. When too large of a δ value is chosen, the chain jumps frequently between the modes, but does not mix well within the individual modes. For example, when $\delta = 0.1$ is chosen, the chain has a tendency to unpropotionately stay in one of the modes, as shown in Figure 1.2(top). On the other hand, when $\delta = 12$, the chain visits the two modes with approximately the same frequency, but the inter-mode sampling is subpar, as shown in Figure 1.2(bottom).



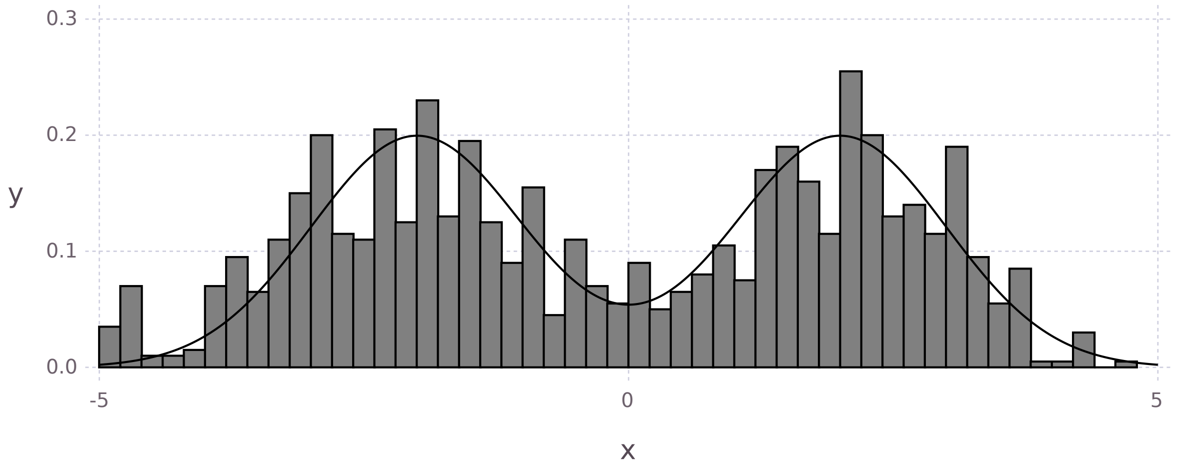


Figure 1.2: Histogram of 1000 samples using the MH algorithm targeting a mixture of 2 Gaussians, using a symmetric candidate with variance (top) $\delta = 0.1$, (bottom) $\delta = 12$. The true density is given by the smooth curve.

Chapter 2

Particle Filtering and Smoothing

Particle filters and smoothers are sequential Monte Carlo methods that are often employed to sample from and provide estimates of the distribution of a set or subset of latent variables in a hidden Markov model given observations. They are constructed specifically to provide updated sampling and estimation when additional observations become available without reprocessing all observations.

Consider the hidden Markov model with underlying and unobserved states $\{X_n\}_{n=0}^{\infty}$, transition density $\pi(x_n|x_{n-1})$, and an initial distribution with density $\pi(x_0)$. (In this chapter we will assume a continuous state space, though the sampling techniques described will apply in the discrete case as well.) Suppose that $\{Y_n\}_{n=1}^{\infty}$ represents a sequence of observable variables that are conditionally independent when the unobserved states are given and where each Y_n is related to the unobserved process through X_n and a "measurement model" density $\pi(y_n|x_n)$. Such a model is depicted in Figure 2.1(left).

Our goal is to sample from the density $\pi(x_{0:n}|y_{1:n})$, where $u_{i:j}$ denotes the vector $(u_i, u_{i+1}, \dots, u_j)$ for $j \geq i$, sequentially in the sense that samples from $\pi(x_{0:n-1}|y_{1:n-1})$ will be used along with a new observation y_n to produce the desired points. The sampled points can then be used to approximate, for example, expectations of the form $\mathsf{E}[f(X_{0:n})|y_{1:n}]$ and marginal distributions $\pi(x_i|y_{1:n})$ for $i = 0, 1, \dots, n$. The estimation of $\pi(x_n|y_{1:n})$ is known as the **filtering** problem and that of $\pi(x_i|y_{1:n})$ for $0 \leq i < n$ is known as the **smoothing** problem.

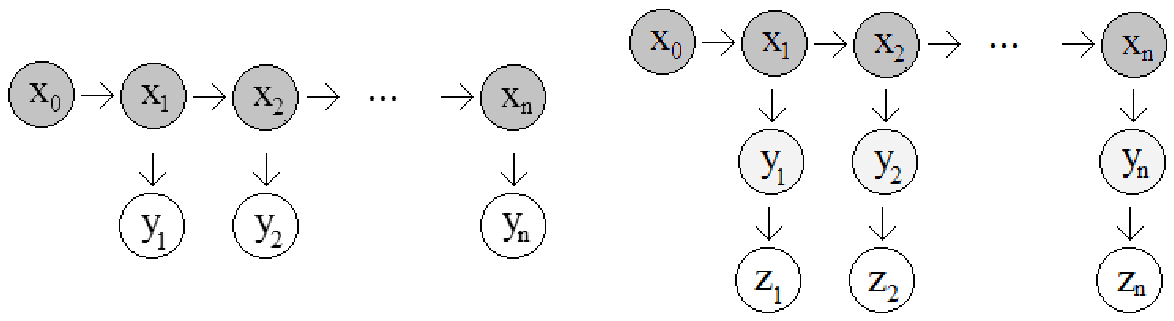


Figure 2.1: Depiction of Hidden Markov Models with (left) one hidden layer and (right) two hidden layers

2.1 Importance Sampling Based Sequential Methods

We now briefly describe existing importance sampling and resampling approaches to the problem of estimating and/or sampling sequentially from a target density

$$\pi(x_{1:n}) = h(x_{1:n})/Z_n, \tag{2.1}$$

where $Z_n = \int h(x_{1:n})\,dx_{1:n}$. A much more complete review is given by Doucet and Johansen [19]. Without loss of generality, $x_{1:n}$ can be replaced by $x_{0:n}$ and the target density can be a conditional density.

To establish notation, note that, if $X_{1:n}^{(1)}, X_{1:n}^{(2)}, \ldots, X_{1:n}^{(N)}$ represent N points ("particles") sampled from $\pi(x_{1:n})$, sequentially or otherwise, a simple unbiased estimator of the target density is given by the empirical density that puts weight $1/N$ on each of the points. We may write this succinctly as

$$\widehat{\pi}(x_{1:n}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[X_{1:n}^{(i)} = x_{1:n}] \tag{2.2}$$

where $\mathbb{1}[X_{1:n}^{(i)} = x_{1:n}]$ is the indicator function that takes on the value 1 when $x_{1:n} = X_{1:n}^{(i)}$, and zero otherwise.

We can then estimate, for example, $\mathsf{E}[f(X_{1:n})]$, for a given function f by

$$\widehat{\mathsf{E}}[f(X_{1:n})] = \frac{1}{N}\sum_{i=1}^{N} f(X_{1:n}^{(i)}). \tag{2.3}$$

It is easy to verify that (2.2) and (2.3) are unbiased for $\pi(x_{1:n})$ and $\mathsf{E}[f(X_{1:n})]$, respectively.

2.1.1 A Review of Non-Sequential Importance Sampling

When one cannot sample directly from $\pi(x_{1:n}) \propto h(x_{1:n})$, importance sampling can be used to instead sample points from a more tractable density, and these points can be used to estimate the target density. Suppose that $q(x_{1:n})$ is another density, presumably tractable, with the same support as $\pi(x_{1:n})$.

We can write

$$\pi(x_{1:n}) = \frac{h(x_{1:n})}{Z_n} = \frac{w(x_{1:n})q(x_{1:n})}{Z_n} \tag{2.4}$$

where

$$w(x_{1:n}) := \frac{h(x_{1:n})}{q(x_{1:n})}. \tag{2.5}$$

Then we proceed to estimate $\pi(x_{1:n})$ and Z_n as follows.

- Sample $X_{1:n}^{(1)}, X_{1:n}^{(2)}, \ldots, X_{1:n}^{(N)} \overset{iid}{\sim} q(x_{1:n})$.[1]
- Estimate $q(x_{1:n})$ with
$$\widehat{q(x_{1:n})} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[X_{1:n}^{(i)} = x_{1:n}] \tag{2.6}$$
- Estimate $Z_n = \int w(x_{1:n})q(x_{1:n})\,dx_{1:n} = \mathsf{E}[w(X_{1:n})]$, when $X_{1:n} \sim q(x_{1:n})$, with
$$\widehat{Z_n} = \frac{1}{N}\sum_{i=1}^{N} w(X_{1:n}^{(i)}). \tag{2.7}$$
- Estimate $\pi(x_{1:n})$ with
$$\begin{aligned} \widehat{\pi(x_{1:n})} &= \tfrac{w(x_{1:n})\widehat{q(x_{1:n})}}{\widehat{Z_n}} = \tfrac{\frac{1}{N}\sum_{i=1}^{N} w(X_{1:n}^{(i)})\mathbb{1}[X_{1:n}^{(i)}=x_{1:n}]}{\widehat{Z_n}} \\ &= \textstyle\sum_{i=1}^{N} W_n^{(i)}\,\mathbb{1}[X_{1:n}^{(i)} = x_{1:n}] \end{aligned} \tag{2.8}$$
where
$$W_n^{(i)} = \frac{w(X_{1:n}^{(i)})}{\sum_{j=1}^{N} w(X_{1:n}^{(j)})}.$$

It is routine to show that (2.8) is an unbiased estimator of $\pi(x_{1:n})$ and that the optimal importance sampling density $q(x_{1:n})$, in terms of minimizing variance, is $q(x_{1:n}) \equiv \pi(x_{1:n})$.

[1] "iid" denotes "independent and identically distributed" values.

2.1.2 Sequential Importance Sampling (SIS)

Sequential importance sampling (SIS) is importance sampling for $\pi(x_{1:n})$ in such a way where draws from $\pi(x_{1:n-1})$ are "extended" to n-dimensional points that are then reweighted to produce draws from $\pi(x_{1:n})$. To this end, the importance sampling density is chosen to have the form

$$q(x_{1:n}) = q(x_1) \prod_{i=2}^{n} q(x_i|x_{i-1})$$

so that it may be sampled from recursively.

The associated importance sample weights may then also be computed recursively since, for $n \geq 2$,

$$\begin{aligned} w(x_{1:n}) &= \frac{h(x_{1:n})}{q(x_{1:n})} = \frac{h(x_{1:n})}{q(x_{1:n-1})q(x_n|x_{n-1})} \\ &= \frac{h(x_{1:n-1})}{q(x_{1:n-1})} \cdot \frac{h(x_{1:n})}{h(x_{1:n-1})q(x_n|x_{n-1})} \\ &=: w(x_{1:n-1}) \cdot \alpha(x_{1:n}) \end{aligned} \tag{2.9}$$

where $\alpha(x_{1:n})$ is an **incremental weight function** that is defined as

$$\alpha(x_{1:n}) := \frac{h(x_{1:n})}{h(x_{1:n-1})q(x_n|x_{n-1})}.$$

SIS Algorithm

At the first time step ($n = 1$),

- Sample $X_1^{(1)}, X_1^{(2)}, \ldots, X_1^{(N)} \overset{iid}{\sim} q(x_1)$.
- Compute weights $w(X_1^{(i)})$ for $i = 1, 2, \ldots, N$ using (2.5).

At times $n \geq 2$,

- Sample $X_n^{(1)}, X_n^{(2)}, \ldots, X_n^{(N)}$ independently with $X_n^{(i)} \sim q(x_n | X_{n-1}^{(i)})$.

- Compute weights $w(X_{1:n}^{(i)}) = w(X_{1:n-1}^{(i)}) \cdot \alpha(X_{1:n}^{(i)})$ for $i = 1, 2, \ldots, N$.

At any time n, one can estimate $\pi(x_{1:n})$ and Z_n using (2.8) and (2.7). One can also obtain approximate dependent draws from $\pi(x_{1:n})$ by sampling from (2.8). That is, by sampling from the set of values $\{X_{1:n}^{(1)}, X_{1:n}^{(2)}, \ldots, X_{1:n}^{(N)}\}$ using respective weights $\{W_n^{(1)}, W_n^{(2)}, \ldots, W_n^{(N)}\}$.

In practice, iteration of the SIS algorithm leads to a "degeneracy of weights" problem (see, for example, [2], [17], [18], and [28]) where the weights of all but one particle will approach zero, causing the method to break down and give meaningless results.

One way to avoid the issue of degenerate weights is to implement a resampling scheme at each time step.

SIS Algorithm With Resampling (SIR)

At the first time step ($n = 1$),

- Sample $X_1^{(1)}, X_1^{(2)}, \ldots, X_1^{(N)} \overset{iid}{\sim} q(x_1)$.

- Compute weights $w(X_1^{(i)})$ for $i = 1, 2, \ldots, N$.

- Compute the normalized weights

$$W_1^{(i)} = \frac{w(X_1^{(i)})}{\sum_{j=1}^{N} w(X_1^{(j)})}$$

  for $i = 1, 2, \ldots, N$.

- Sample N points, $\widetilde{X}_1^{(1)}, \widetilde{X}_1^{(2)}, \ldots, \widetilde{X}_1^{(N)}$, with replacement, from the set $\{X_1^{(1)}, X_1^{(2)}, \ldots, X_1^{(N)}\}$ with respective probabilities $\{W_1^{(1)}, W_1^{(2)}, \ldots, W_1^{(N)}\}$. That is, sample $\widetilde{X}_1^{(1)}, \widetilde{X}_1^{(2)}, \ldots, \widetilde{X}_1^{(N)}$ from

$$\widehat{\pi(x_1)} = \sum_{i=1}^{N} W_1^{(i)} \, \mathbb{1}[X_1^{(i)} = x_1].$$

Note that $\widetilde{X}_1^{(1)}, \widetilde{X}_1^{(2)}, \ldots, \widetilde{X}_1^{(N)}$ are now equally weighted particles, each with weight $1/N$. Assign weights $w(\widetilde{X}_1^{(i)}) = 1/N$ for $i = 1, 2, \ldots, N$.

At times $n \geq 2$,

- Sample $X_n^{(1)}, X_n^{(2)}, \ldots, X_n^{(N)}$ independently with $X_n^{(i)} \sim q(x_n|\widetilde{X}_{n-1}^{(i)})$.
- Extend each particle $\widetilde{X}_{1:n-1}^{(i)}$ to particles $(\widetilde{X}_{1:n-1}^{(i)}, X_n^{(i)})$.
- Compute associated weights $w(\widetilde{X}_{1:n-1}^{(i)}, X_n^{(i)}) := \alpha(\widetilde{X}_{1:n-1}^{(i)}, X_n^{(i)})$ for $i = 1, 2, \ldots, N$.

  (This is consistent with SIS since the previous weights in (2.9) have been replaced by the constant $1/N$ and the current weights have yet to be normalized.)
- Compute the normalized weights

$$W_n^{(i)} = \frac{w(\widetilde{X}_{1:n-1}^{(i)}, X_n^{(i)})}{\sum_{j=1}^N w(\widetilde{X}_{1:n-1}^{(j)}, X_n^{(j)})}$$

  for $i = 1, 2, \ldots, N$.
- Sample N n-dimensional points, $\widetilde{X}_{1:n}^{(1)}, \widetilde{X}_{1:n}^{(2)}, \ldots, \widetilde{X}_{1:n}^{(N)}$, with replacement, from the set $\{(\widetilde{X}_{1:n-1}^{(1)}, X_n^{(i)}), (\widetilde{X}_{1:n-1}^{(2)}, X_n^{(2)}), \ldots, (\widetilde{X}_{1:n-1}^{(N)}, X_n^{(N)})\}$ with respective probabilities $\{W_n^{(1)}, W_n^{(2)}, \ldots, W_n^{(N)}\}$. That is, sample $\widetilde{X}_{1:n}^{(1)}, \widetilde{X}_{1:n}^{(2)}, \ldots, \widetilde{X}_{1:n}^{(N)}$ from

$$\widehat{\pi(x_{1:n})} = \sum_{i=1}^N W_n^{(i)} \, 1\!\!1[(\widetilde{X}_{1:n-1}^{(i)}, X_n^{(i)}) = x_{1:n}]. \tag{2.10}$$

  Note that $\widetilde{X}_{1:n}^{(1)}, \widetilde{X}_{1:n}^{(2)}, \ldots, \widetilde{X}_{1:n}^{(N)}$ are now equally weighted particles, each with weight $1/N$. Assign weights $w(\widetilde{X}_{1:n}^{(i)}) = 1/N$ for $i = 1, 2, \ldots, N$.

As with SIS, we may, at any time n, estimate $\pi(x_{1:n})$ using (2.10) or by using

$$\widehat{\pi(x_{1:n})} = \frac{1}{N} \sum_{i=1}^N 1\!\!1[\widetilde{X}_{1:n}^{(i)} = x_{1:n}]. \tag{2.11}$$

We may obtain approximate dependent draws from $\pi(x_{1:n})$ by sampling from (2.11). That is, by sampling uniformly, with replacement, from the set of values $\{\widetilde{X}_{1:n}^{(1)}, \widetilde{X}_{1:n}^{(2)}, \dots, \widetilde{X}_{1:n}^{(N)}\}$.

An obvious issue with the SIR algorithm, known as the **sample impoverishment problem**, is a decreasing diversity of particles as those with higher weights will likely be drawn multiple times. Additionally, the resampling step may make the algorithm prohibitively slow and can lead to increased variance of estimates. One way to address the speed and variance is to only resample in the case of a small (user chosen threshold for) **effective sample size**

$$N_{eff} := \left[\sum_{i=1}^{N} (W_n^{(i)})^2\right]^{-1}. \tag{2.12}$$

A small effective sample size indicates higher variability of the weights which, in turn, indicates impending degeneracy. In this case, resampling would be especially prudent.

It is important to note that, when resampling, one should be originally sampling a very large number of points in order to minimize the effects of resampling repeated values from a discrete distribution. Additional details about SIS and SIR procedures can be found in Doucet **et al.** [17, 18] and in Andrieu and Doucet [1].

2.2 Rejection Sampling Based Sequential Methods

2.3 Windowed Rejection Sampling

In order to draw from a possibly unnormalized density $\pi(x) \propto h(x)$ using rejection sampling, one must find a density $q(x)$ and a constant M such that

$$h(x) \leq Mq(x).$$

Then, one proceeds as in Algorithm 1.

Algorithm 1 Rejection Sampling from $\pi(x)$:

```
1. Generate $X \sim q$.
2. Generate $U \sim$ Uniform$(0, 1)$.
3. If $U > \frac{h(X)}{Mq(X)}$, return to 1. Otherwise, $X$ is a draw from $\pi$.
```

Since $\pi(x)$ may be a high-dimensional and conditional density, we could, in theory, use rejection sampling to draw exactly from

$$\begin{aligned}\pi(x_{0:n}|y_{1:n}) \propto h(x_{0:n}|y_{1:n}) &:= \pi(x_0)\textstyle\prod_{i=1}^{n}\pi(y_i|x_i)\pi(x_i|x_{i-1}) \\ &= \underbrace{\left[\prod_{i=1}^{n}\pi(y_i|x_i^*)\right]}_{M} \cdot \underbrace{\pi(x_0)\prod_{i=1}^{n}\pi(x_i|x_{i-1})}_{q(x_{0:n})}\end{aligned}$$

where

$$x_i^* = \arg\max_{x_i} \pi(y_i|x_i).$$

For most applications of particle filtering in the literature, x_i^*, or at least an upper bound on $\pi(y_i|x_i)$, with respect to x_i, is easily attainable. However, for even small values of n, the acceptance probability, $\prod_{i=1}^{n}\pi(y_i|x_i)/\pi(y_i|x_i^*)$, can be prohibitively small and of course we are losing the benefit of sequential sampling. Thus, we propose rejection sampling "in windows".

2.3.1 Rejection Sampling in Windows

"Windowed Rejection Sampling" (WRS) is based on the idea that, depending on the covariance structure of the chain, at some point future observed y's will eventually cease to significantly affect earlier x's. For example, if $\pi(x_0|y_1,y_2,y_3,y_4) \approx \pi(x_0|y_1,y_2,y_3)$, we can write

$$\begin{aligned}\pi(x_0,x_1,x_2,x_3,x_4|y_1,y_2,y_3,y_4) &= \pi(x_1,x_2,x_3,x_4|x_0,y_1,y_2,y_3,y_4)\cdot\pi(x_0|y_1,y_2,y_3,y_4) \\ &\approx \pi(x_1,x_2,x_3,x_4|x_0,y_1,y_2,y_3,y_4)\cdot\pi(x_0|y_1,y_2,y_3),\end{aligned}$$

and so we can sample approximately from $\pi(x_{0:4}|y_{1:4})$ by first sampling X_0 from $\pi(x_0|y_{1:3})$ and then sampling $X_{1:4}$ from $\pi(x_1,x_2,x_3,x_4|x_0,y_1,y_2,y_3,y_4)$. Sampling from $\pi(x_0|y_{1:3})$ can be achieved by sampling $X_{0:3}$ from $\pi(x_{0:3}|y_{1:3})$ and considering only the x_0 values. In this example, we say that we are using rejection sampling in a "window of length 4".

More formally, the WRS algorithm is described in Algorithm 2 for a given window length w. We discuss the choice of this tuning parameter in Sections 4 and 5.

Algorithm 2 Windowed Rejection Sampling from $\pi(x_{0:n}|y_{1:n})$:

Find $x_i^* = \arg\max_{x_i} \pi(y_i|x_i)$, and set a window length parameter w.

1. Generate $X_{0:w-1} \sim \pi(x_{0:w-1}|y_{1:w-1})$ using rejection sampling with

$$q(x_{0:w-1}) = \pi(x_0) \prod_{i=1}^{w-1} \pi(x_i|x_{i-1})$$

and $M = \prod_{i=1}^{w} \pi(y_i|x_{i-1}^*)$. Set $m = 1$.

2. Generate $X_{m:m+w-1} \sim \pi(x_{m:m+w-1}|x_{m-1}, y_{1:m+w-1}) = \pi(x_{m:m+w-1}|x_{m-1}, y_{m:m+w-1})$ using rejection sampling with

$$q(x_{m:m+w-1}|x_{m-1}) = \prod_{i=m}^{m+w-1} \pi(x_i|x_{i-1})$$

and $M = \prod_{i=m}^{m+w-1} \pi(y_i|x_i^*)$.

Note that

$$\begin{aligned} \pi(x_{m:m+w-1}|x_{m-1}, y_{m:m+w-1}) &\propto h(x_{m:m+w-1}|x_{m-1}, y_{m:m+w-1}) \\ &:= \textstyle\prod_{i=m}^{m+w-1} \pi(y_i|x_i)\pi(x_i|x_{i-1}) \leq M \cdot q(x_{m,n+w-1}|x_{m-1}). \end{aligned}$$

Set $m = m + 1$ and return to the beginning of Step 2. Continue until $m + w - 1 = n$.

Note that if it is not obtainable, $\pi(y_i|x_i^*)$ in expressions for M can be replaced by any upper bound.

2.4 Examples

2.4.1 One Dimensional Normals with One Hidden Layer

We begin, for the purpose of illustration, with a very explicit description of the algorithm with a window length of $w = 3$, for the simple model

$$X_0 \sim N(\mu_0, \sigma_0^2), \qquad X_{n+1} = aX_n + \sigma_X \varepsilon_{n+1}, \qquad Y_n = bX_n + \sigma_Y \nu_n$$

where $\{\varepsilon_n\} \overset{iid}{\sim} N(0,1)$ and $\{\nu_n\} \overset{iid}{\sim} N(0,1)$ are independent sequences. Assume that only the Y_n are observed.

Let $N(x;\mu,\sigma^2)$ denote the $N(\mu,\sigma^2)$ density.

We begin by using rejection sampling to produce N iid draws from $\pi(x_{0:2}|y_{1:2})$

$$\pi(x_{0:2}|y_{1:2}) \propto [\pi(y_1|x_1)\pi(y_2|x_2)] \cdot [\pi(x_2|x_1)\pi(x_1|x_0)\pi(x_0)]$$

using $q(x_{0:2}) = \pi(x_2|x_1)\pi(x_1|x_0)\pi(x_0) = N(x_2; ax_1, \sigma_X^2) \cdot N(x_1; ax_0, \sigma_X^2) \cdot N(x_0; \mu_0, \sigma_0^2)$.

It is easy to see that

$$x_i^* = \arg\max_{x_i} \pi(y_i|x_i) = \arg\max_{x_i} N(y_i; bx_i, \sigma_Y^2) = y_i/b \qquad \text{for } \ i = 1, 2, \ldots.$$

We repeatedly draw independent realizations $x_{0:2}$ of $X_{0:2} \sim q(x_{0:2})$ and u of $U \sim \text{Uniform}(0,1)$ until the first time that

$$u \leq \frac{\pi(y_1|x_1)\pi(y_2|x_2)}{\pi(y_1|x_1^*)\pi(y_2|x_2^*)}.$$

Repeating this procedure N times, we collect only the values of x_0 and record them as $X_0^{(1)}, X_0^{(2)}, \ldots, X_0^{(N)}$.

Now, moving our window of length $w = 3$ to the right by 1, we use rejection sampling to produce N iid draws from $\pi(x_{1:3}|x_0, y_{1:3})$

$$\pi(x_{1:3}|x_0, y_{1:3}) \propto [\pi(y_1|x_1)\pi(y_2|x_2)\pi(y_3|x_3)] \cdot [\pi(x_3|x_2)\pi(x_2|x_1)\pi(x_1|x_0)]$$

using

$$q(x_{1:3}|x_0) = \pi(x_3|x_2)\pi(x_2|x_1)\pi(x_1|x_0) = N(x_3; ax_2, \sigma_X^2) \cdot N(x_2; ax_1, \sigma_X^2) \cdot N(x_1; ax_0, \sigma_X^2).$$

That is, for each $i = 1, 2, \ldots, N$, we repeatedly draw independent realizations $x_{1:3}$ of $X_{1:3} \sim q(x_{1:3}|X_0^{(i)})$ and u of $U \sim \text{Uniform}(0,1)$ until the first time that

$$u \leq \frac{\pi(y_1|x_1)\pi(y_2|x_2)\pi(y_3|x_3)}{\pi(y_1|x_1^*)\pi(y_2|x_2^*)\pi(y_3|x_3^*)},$$

collecting the resulting value of x_1 and recording it as $X_1^{(i)}$.

Move the window of length $w = 3$ to the right by 1 to produce N draws from $\pi(x_{2:4}|x_1, y_{1:4}) = \pi(x_{2:4}|x_1, y_{2:4})$, and retain the resulting values for $X_2^{(i)}$ for $i = 1, 2, \ldots, N$.

Continuing in this manner, we generated $N = 100,000$ independent values for $X_{0:11}$ for this model using parameters $\mu_0 = 3.0$, $\sigma_0 = 2.0$, $a = 0.9$, $b = 1.2$, $\sigma_X = 3.0$, and $\sigma_Y = 2.3$. (We looked first at such a large N in order to really see the resulting distribution without being concerned about sampling variability.) For comparison, we produced N independent draws directly from the 11-dimensional distribution $\pi(x_{0:10}|y_{1:10})$ using 11-dimensional (non-windowed) rejection sampling. (As expected, the acceptance probabilities are quite small in this case and such direct simulation from the target density is not reasonably achieved for much larger n.) Finally, we produced N dependent draws approximately from $\pi(x_{0:10}|y_{1:10})$ using the SIR algorithm. Figure 2.2 shows the average marginal values for X_0 through X_{10} for each algorithm. Only the WRS results change between graphs, showing the anticipated increasing accuracy of the WRS algorithm as the window length increases. For the given model and parameters, a window length of $w = 3$ appears to be sufficient. Indeed, as the perfect high-dimensional rejection sampling algorithm is a special case of the WRS algorithm for fixed n and maximal window length, it is easy to code the WRS algorithm once and run it first to get perfect draws from $\pi(x_{0:n}|y_{1:n})$ for the highest feasible n and then to run it with lower w, gradually increasing w until sample statistics such as those shown in Figure 2.2 reach the desired accuracy.

For this example, the WRS algorithm produced $100,000$ iid draws from an approximation to $\pi(x_{0:11}|y_{1:10})$ whereas the SIR algorithm (resampling every time) produced dependent draws with only roughly 50% unique 11-dimensional values and only about 10% unique values for X_0 marginally. Figure 2.3 shows marginal distributions for X_7 produced by the WRS and SIR algorithms. The overlaid curves are the target normal densities that can be computed analytically for this simple illustrative model. The time to run the WRS and SIR algorithms were comparable. Coded in C++, the WRS algorithm completed in less than 2 seconds on a standard[2] laptop while the SIR algorithm completed in 8-10 seconds. The SIR algorithm would obviously speed up if resampling was not done at every step and would likely be the faster algorithm if both were programmed in R where resampling is efficient and accept/reject looping is inefficient.

[2] We wish to convey relative speed between algorithms in lieu of particular machine specifications.

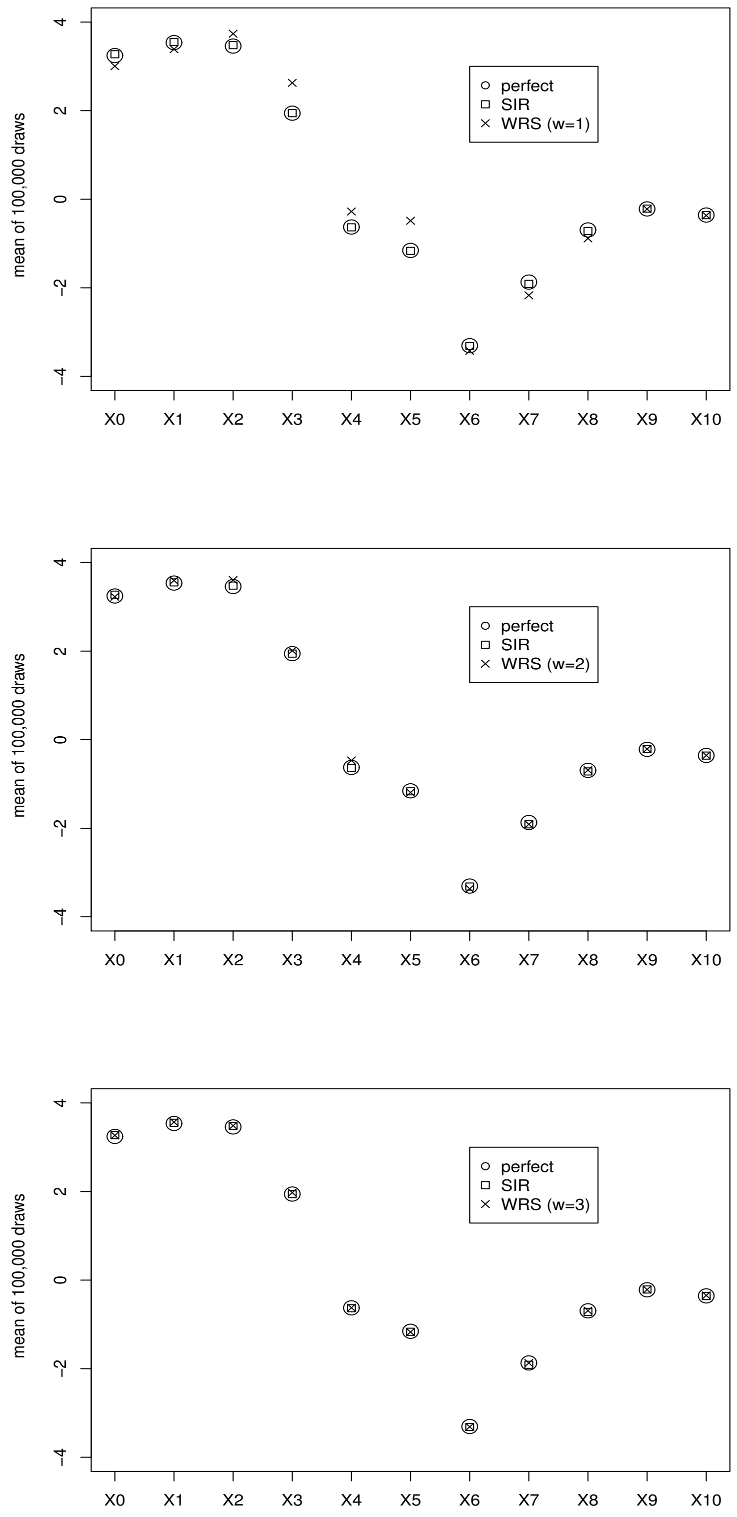


Figure 2.2: Means for marginal components in $100,000$ draws from $\pi(x_{0:10}|y_{1:10})$ in the One Dimensional Normal model using perfect 11-dimensional rejection sampling, the SIR algorithm, and the WRS algorithm with various window lengths.

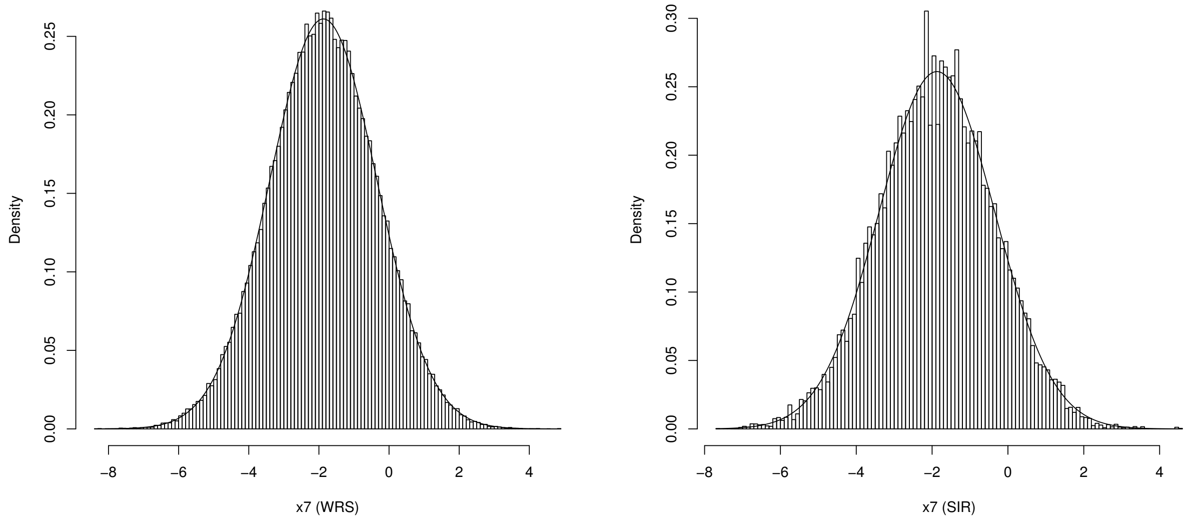


Figure 2.3: Histogram of values of X_7 from 100, 000 draws from $\pi(x_{0:11}|y_{1:10})$ in the One Dimensional Normal model using (left) WRS with $w = 3$ and (right) SIR

When comparing the algorithms for larger n, it becomes crucial for efficiency to not resample at every time step of the SIR algorithm. As per the discussion at the end of Section 2.1, we resample only when the effective sample size N_{eff}, given by (2.12) falls below the a threshold N_{thres}. Following a suggestion in [18], we choose $N_{thres} = N/3$. Due to the Markov nature of the model, we are able to continue forward with the window length $w = 3$ for increasing n. When $n = 1,000$, the estimated filtering distribution $\pi(x_{1000}|y_{1:1000})$ is shown in Figure 2.4 for both algorithms. The SIR algorithm resulted in approximately 52% distinct values. An estimated smoothing distribution, namely an estimate of $\pi(x_{300}|y_{1:1000})$ is shown in Figure 2.5. It is well known (see [19]) that the SIR algorithm will produce severely impoverished samples for $\pi(x_i|y_{1:n})$ and i small relative to n, and in this example we are left with only 0.07% distinct values. Suggested approaches to mitigate degeneracy in SIR are discussed in [19], but the WRS algorithm performs well without modification.

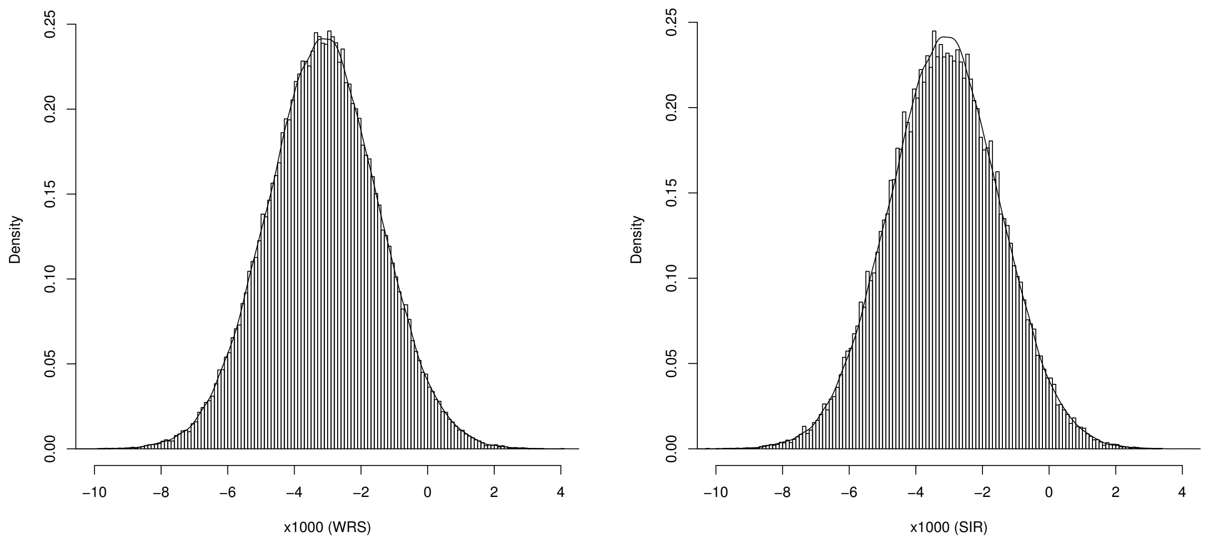


Figure 2.4: Histogram of values of X_{1000} from 100,000 draws from $\pi(x_{0:1000}|y_{1:1000})$ in the One Dimensional Normal model using (left) WRS with $w = 3$ and (right) SIR

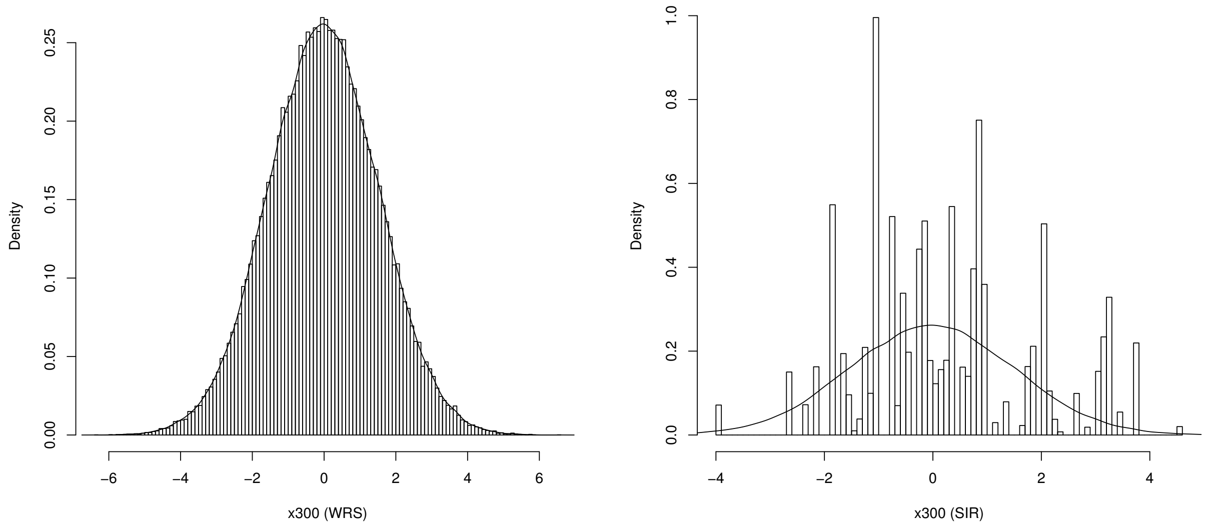


Figure 2.5: Histogram of values of X_{300} from $100,000$ draws from $\pi(x_{0:1000}|y_{1:1000})$ in the One Dimensional Normal model using (left) WRS with $w = 3$ and (right) SIR

2.4.2 Stochastic Volatility Model

We now consider the **stochastic volatility** model, often used in finance and also considered in [19], where

$$X_0 \sim N(0, \sigma^2/(1-\alpha^2)), \qquad X_{n+1} = \alpha X_n + \sigma \varepsilon_{n+1}, \qquad Y_n = \beta e^{X_n/2} \nu_n$$

where $\{\varepsilon_n\} \overset{iid}{\sim} N(0,1)$ and $\{\nu_n\} \overset{iid}{\sim} N(0,1)$ are independent sequences. Again we assume that only the Y_n are observed.

Following the same procedure in Section 2.4.1, using a general window length w, we need to define $q(x_{0:w-1})$, x_i^* and $q(x_{j:(j+w-1)}|x_{j-1})$. It is easy to show that

$$x_i^* = \arg\max_{x_i} \pi(y_i|x_i) = \arg\max_{x_i} N(y_i; 0, \beta^2 e^{x_i}) = \ln(y_i^2/\beta^2) \qquad \text{for } i = 1, 2, \ldots.$$

Taking

$$q(x_0) = \pi(x_0) = N(x_0; 0, \sigma^2/(1-\alpha^2)) \text{ and } q(x_i|x_{i-1}) = \pi(x_{i|x_{i-1}}) = N(x_i; \alpha x_{i-1}, \sigma^2),$$

we have

$$q(x_{0:w-1}) = q(x_0) \prod_{i=1}^{w-1} q(x_i|x_{i-1}) \tag{2.13}$$

and

$$q(x_{j:(j+w-1)}|x_{j-1}) = \prod_{i=j}^{j+w-1} q(x_i|x_{i-1}). \tag{2.14}$$

We ran Step 1 of the WRS algorithm (Algorithm 2) with $w = 11$ in order to get $N = 100,000$ perfect draws via rejection sampling from $\pi(x_{0:10}|y_{1:10})$. ($n = 10$ was chosen as the maximal value after which rejection sampling from $\pi(x_{0:n}|y_{1:n})$ became prohibitively slow.) We then ran the complete WRS algorithm with $n = 10$ and increasing w, starting from $w = 1$, until the marginal sample means from WRS for $X_i|y_{1:10}$ produced rough (non-simultaneous) two standard deviation confidence intervals which contained the perfect draw estimates of the means. That is, if $\hat{\mu}_i$ denotes the sample mean of the $100,000$ values of X_i resulting from rejection sampling from $\pi(x_{0:10}|y_{1:10})$ and $\overline{X}_i$ and S_i^2 denote the sample means and variances of the marginal distributions produced by WRS, w was increased until $\hat{\mu}_i \in \overline{X}_i \pm 2S_i/\sqrt{N}$ for all $i \in \{0, 1, \ldots, 10\}$. This resulted in a window length of 5. Figure 2.6 shows the sample means for the marginal draws with WRS and $w = 5$ aligning with those from both perfect and SIR draws. Figure 2.7 shows the proportion of surviving distinct values for each marginal distribution using SIR, which is in contrast to the WRS algorithm giving 100% distinct values for all marginals. Figure 2.8 shows histograms of the marginal distribution of $X_5|y_{1:10}$ obtained from the draws from $\pi(x_{0:10}|y_{1:10})$ using 11-dimensional rejection sampling, the SIR algorithm, and the WRS algorithm with $w = 5$. The overlaid curve in each case is the density estimated by 100,000 perfect draws of X_5 marginalized from 11-dimensional rejection sampling. The SIR algorithm results stand out as the roughest estimate, though, again, it is well known [19] that this "straight" application of the SIR algorithm does not produce good smoothing estimates and that additional steps should be taken to improve it. The WRS algorithm, on the other hand gives a nice estimate of the smoothing distribution straight away with a speed that is comparable to the initial pass of the SIR algorithm. (By "comparable" we mean that, for the examples in this chapter as well as others we tried, the WRS algorithm was either faster than the SIR algorithm or slower but still finishing only a few seconds behind. In the worst case, the WRS algorithm was about one minute behind the SIR algorithm. No particular effort was made to fully optimize either code.)

2.4.3 A "Highly Nonlinear" Model

In this section, we apply the WRS algorithm the "highly nonlinear" model considered in several papers including [3], [11], [18], and [25].

$$\begin{aligned} X_0 &\sim N(\mu, \sigma^2) \\ X_{n+1} &= 0.5X_n + \tfrac{25X_n}{1+X_n^2} + 8\cos(1.2n) + \varepsilon_{n+1} \\ Y_n &= 0.05X_n^2 + \nu_n \end{aligned}$$

for independent $\varepsilon_n \overset{iid}{\sim} N(0, \sigma_X^2)$ and $\nu_n \overset{iid}{\sim} N(0, \sigma_Y^2)$. The parameters were chosen as $\mu = 0$, $\sigma^2 = 5$, and $\sigma_X^2 = \sigma_Y^2 = 10$.

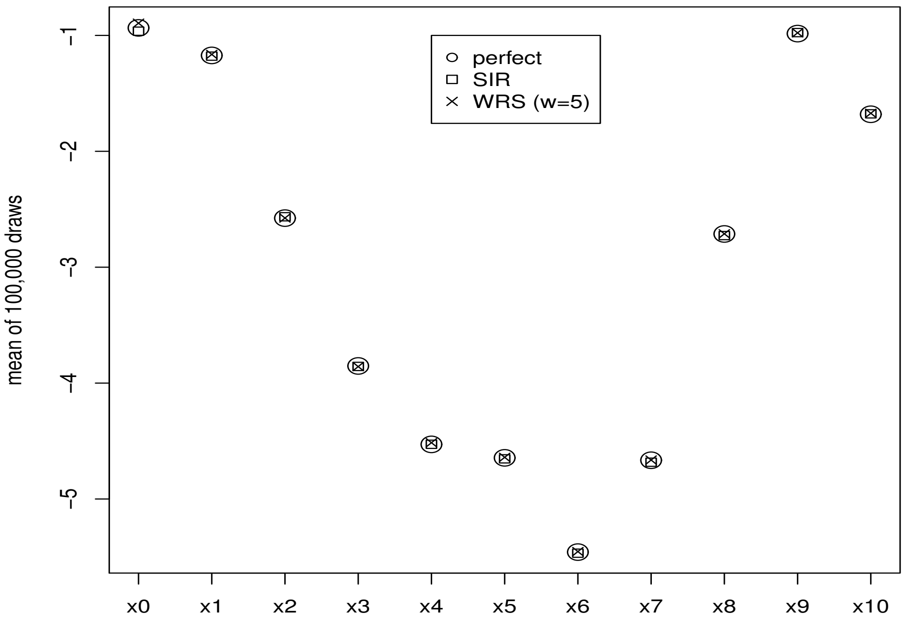


Figure 2.6: Means for marginal components in 100,000 draws from $\pi(x_{0:10}|y_{1:10})$ in the Stochastic Volatility model using perfect 11-dimensional rejection sampling, the SIR algorithm, and the WRS algorithm with $w = 5$.

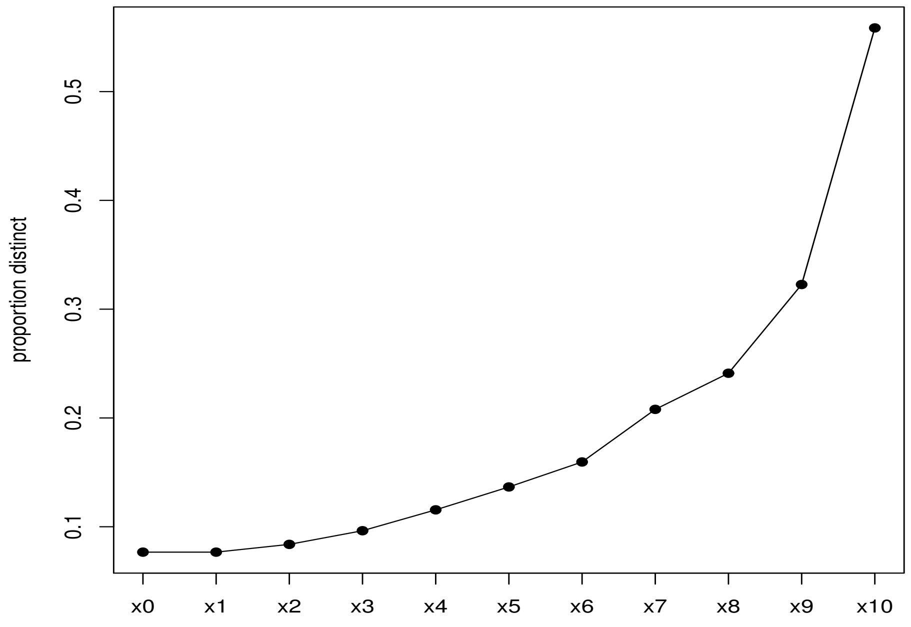


Figure 2.7: Proportion of surviving distinct marginal values using SIR for the Stochastic Volatility model

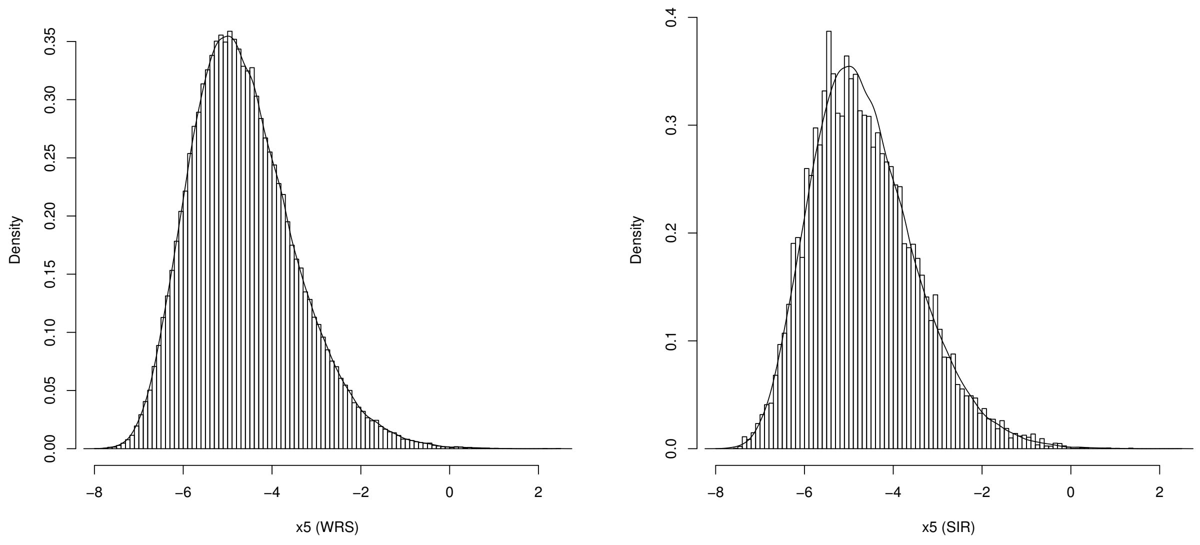


Figure 2.8: 100,000 draws from $X_5|y_{0:10}$ in the Stochastic Volatility model using (left) WRS with $w = 5$ and (right) SIR

Taking

$$q(x_0) = \pi(x_0) = N(x_0; \mu, \sigma^2) \text{ and } q(x_i|x_{i-1}) = \pi(x_i|x_{i-1}) = N(x_i; , m(x_{i-1}), \sigma_X^2),$$

with $m(x_{i-1}) = 0.5x_{i-1} + 25x_{i-1}/(1 + x_{i-1}^2) + 8\cos[1.2(i-1)]$, we define $q(x_{0:w-1})$ and $q(x_{j:(j+w-1)}|x_{j-1})$ using (2.13) and (2.14).

It is easy to show that

$$x_i^* = \arg\max_{x_i} \pi(y_i|x_i) = \begin{cases} 0 & , \text{ if } y_i < 0 \\ \pm\sqrt{y_i/0.05} & , \text{ if } y_i \geq 0. \end{cases}$$

An analysis of marginal means such as those depicted in Figures 2.2 and 2.6 reveal that a window length of 4 is sufficient for this example. Figure 2.9 shows the proportion of surviving distinct values for each marginal distribution using SIR, which, again, is in contrast to the WRS algorithm giving 100% distinct values for all marginals. Figures 2.10 show histograms of $X_5|y_{1:10}$ produced by the WRS and SIR algorithms along with superimposed densities estimated by 11-dimensional rejection sampling. As expected, the WRS results are smoother.

2.4.4 Two Layer Hidden Markov Model

In [1], the authors discuss a Rao-Blackwellized particle filtering technique, based on the SIR algorithm, for estimating $\pi(x_{0:n}|z_{1:n})$ in a two-layer hidden Markov model as depicted in Figure 2.1(right). It involves using SIR to first estimate $\pi(y_{1:n}|z_{1:n})$ and then estimating $\pi(x_{0:n}|z_{1:n})$ using the relationship

$$\pi(x_{0:n}|z_{1:n}) = \int \pi(x_{0:n}|y_{1:n})\pi(y_{1:n}|z_{1:n})$$

and draws from $\widehat{\pi}(y_{1:n}|z_{1:n})$. The WRS algorithm can be used here in place of the SIR algorithm.

In this example, however, we consider a particularly simple two-layer model from [1] which can be dealt with more directly. The "dynamic tobit model" is described as follows.

$$X_0 \sim N\left(0, \frac{\sigma_X^2}{1-\phi^2}\right)$$

$$X_{n+1} = \phi X_n + \sigma_X \varepsilon_{n+1}, \qquad Y_n = X_n + \sigma_Y \nu_n, \qquad Z_n = \max(0, Y_n)$$

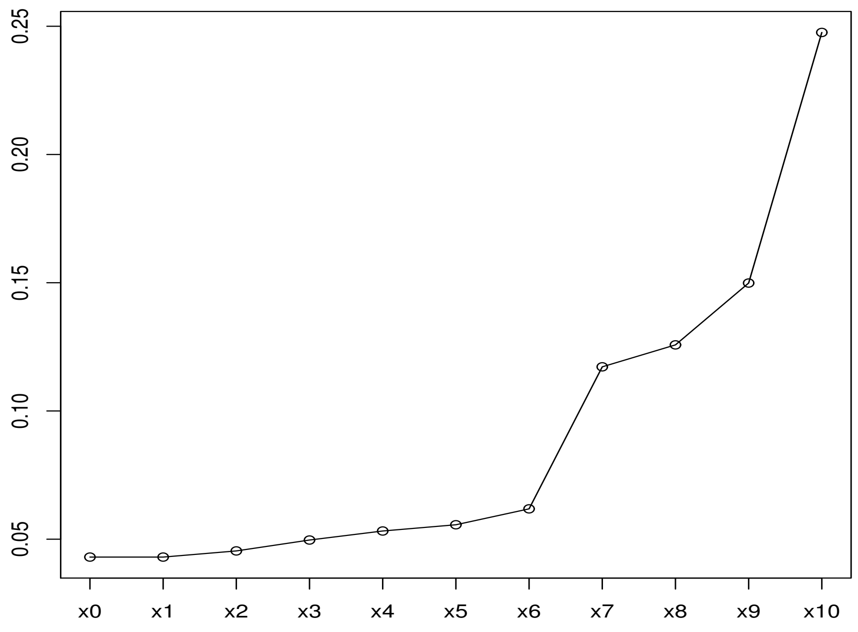


Figure 2.9: Proportion of Surviving Distinct Marginal Values using SIR for the Highly Nonlinear Model

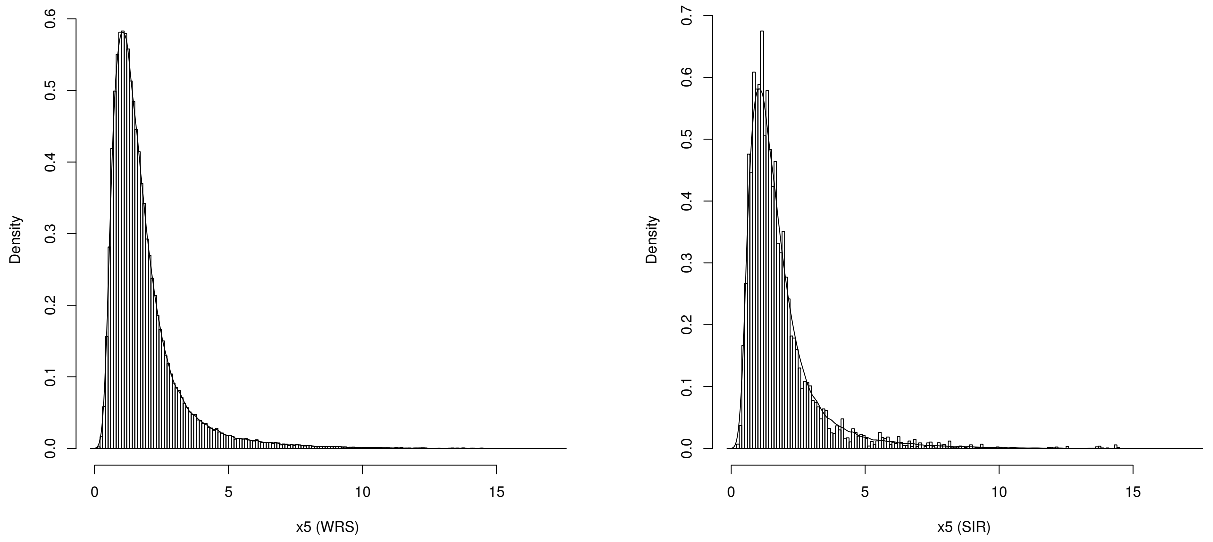


Figure 2.10: 100,000 draws from $X_5|y_{1:10}$ in the Highly Nonlinear model using (left) WRS with $w = 4$ and (right) SIR

where $\{\varepsilon_n\} \overset{iid}{\sim} N(0,1)$ and $\{\nu_n\} \overset{iid}{\sim} N(0,1)$ are independent sequences. Assume that only the Z_n are observed. The parameters were chosen as $\sigma_X^2 = 0.05$, $\sigma_Y^2 = 0.30$, and $\phi = 0.99$, as in [1].

Consider the case first when $n = 1$. If $z_1 > 0$, we wish to draw from

$$\pi(x_0, x_1|z_1) = \pi(x_0, x_1|y_1) \propto h(x_0, x_1|y_1) = \pi(y_1|x_1)\pi(x_1|x_0)\pi(x_0)$$

where $y_1 = z_1$.

To implement the SIR algorithm, we take $q(x_0, x_1) := \pi(x_1|x_0)\pi(x_0)$ and use the weight $w(x_{0:1}) := \pi(y_1|x_1)$. To implement rejection sampling, we use the fact that

$$h(x_0, x_1|y_1) \leq M \cdot q(x_0, x_1)$$

where $M = 1\Big/\sqrt{2\pi\sigma_Y^2}$ and, again, $q(x_0, x_1) := \pi(x_1|x_0)\pi(x_0)$.

If $z_1 = 0$, we wish to draw from

$$\pi(x_0, x_1|z_1) = \pi(x_0, x_1|y_1 < 0) \propto P_\pi(y_1 < 0|x_1)\pi(x_1|x_0)\pi(x_0)$$

where $P_\pi(y_1 < 0|x_1) = \int_{-\infty}^0 \pi(y_1|x_1)\, dy_1$. We then have the unnormalized target density

$$h(x_0, x_1|z_1) = \Phi(-x_1/\sigma_Y) \cdot \pi(x_1|x_0)\pi(x_0) \tag{2.15}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function for the standard normal distribution.

Taking $q(x_0, x_1) := \pi(x_1|x_0)\pi(x_0)$, we can implement the SIR algorithm to draw from (2.15) using $w(x_{0:1}) = \Phi(-x_1/\sigma_Y)$ and we can implement rejection sampling using the fact that $\Phi(-x_1/\sigma_Y) \leq M := 1$.

For general n, we use $q(x_n|x_{n-1}) = \pi(x_n|x_{n-1})$ for both algorithms. The incremental weight for the SIR algorithm is

$$\alpha(x_{1:n}) = \begin{cases} \pi_{Y|X}(z_n|x_n) & , \text{ if } z_n > 0 \\ \\ \Phi(-x_n/\sigma_Y) & , \text{ if } z_n = 0. \end{cases}$$

Here, $\pi_{Y|X}(\cdot|\cdot)$ is the conditional density for Y_n given X_n.

For the WRS algorithm with window length w, we define M in Step 1 of Algorithm 2 as

$$M = \prod_{i=1}^{w} \left\{ \pi_{Y|X}(z_i|x_i) \mathbb{1}[z_i > 0] + \Phi(-x_i/\sigma_Y) \mathbb{1}[z_i = 0] \right\},$$

and M in Step 2 as

$$M = \prod_{i=m}^{m+w-1} \left\{ \pi_{Y|X}(z_i|x_i) \mathbb{1}[z_i > 0] + \Phi(-x_i/\sigma_Y) \mathbb{1}[z_i = 0] \right\}.$$

Since the underlying hidden Markov model is an AR(1) model with autocorrelation function $\rho(h) = \phi^h$, it is not surprising that, for $\phi = 0.99$, we would need w to be large for the WRS algorithm. As in previous examples, we produced $100,000$ independent draws directly from the 11-dimensional distribution $\pi(x_{0:10}|z_{1:10})$ using 11-dimensional, non-windowed, rejection sampling and then used windowed rejection sampling with increasing values of w until we matched marginal means. Figure 2.11 shows the matching means, as well as the matching means from the SIR algorithm. The turning point in this graph of means corresponds to the fact that z_3 was the only data point in $z_{1:10}$ that was positive. Both the WRS and SIR algorithms were quite slow for this example (on the order of 45 minutes) but were still of comparable speeds.

In Figure 2.12 we see that the marginal means for the SIR and WRS algorithms continue to coincide even for large n. Due to the Markov nature of the model, the window length determined with smaller n can still be used.

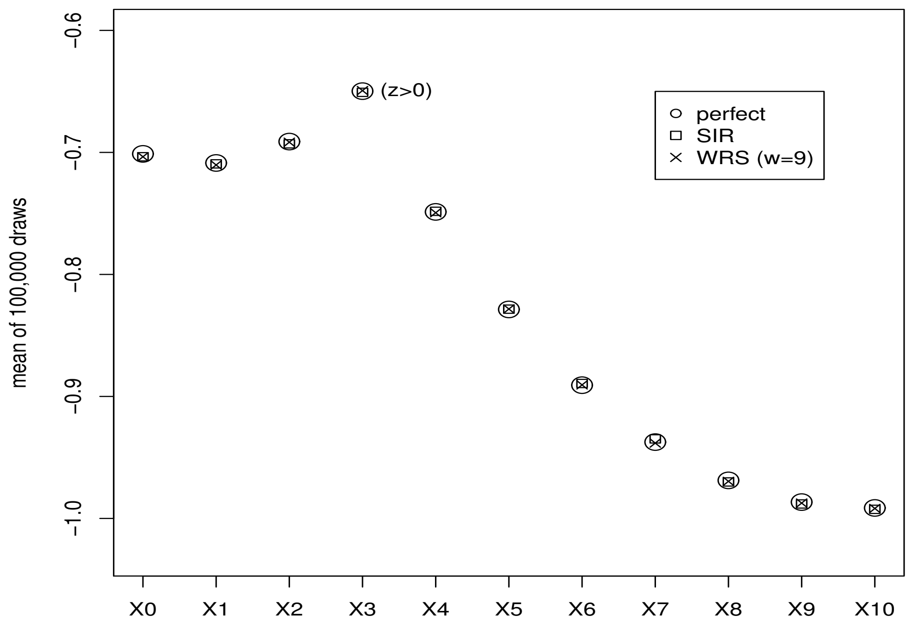


Figure 2.11: Means for marginal components in 100,000 draws from $\pi(x_{0:10}|z_{1:10})$ in the 2 Layer HMM using perfect 11-dimensional rejection sampling, the SIR algorithm, and the WRS algorithm with $w = 9$.

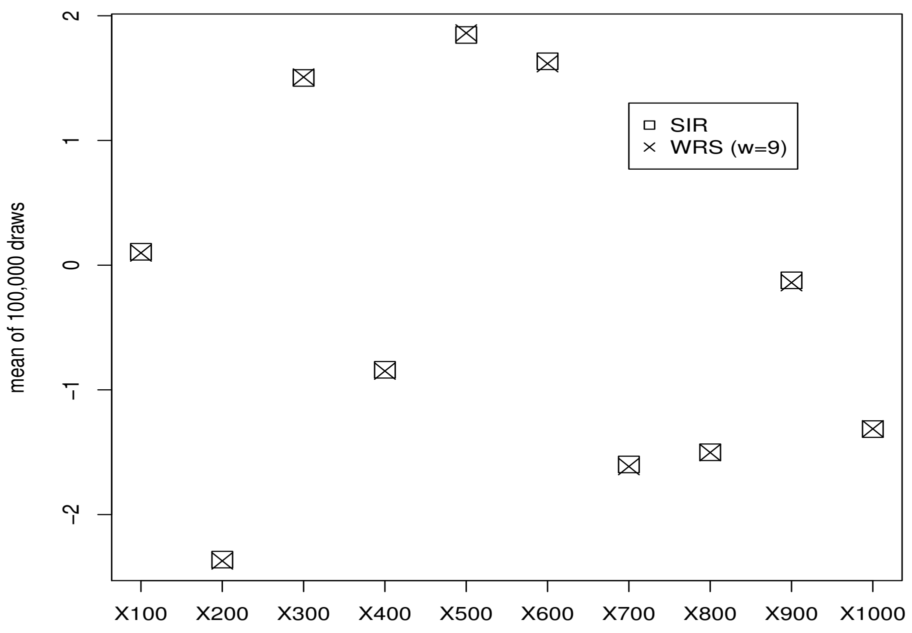


Figure 2.12: Means for certain marginal components in 100,000 draws from $\pi(x_{0:1000}|z_{1:1000})$ in the 2 Layer HMM using the SIR algorithm and the WRS algorithm with $w = 9$.

Chapter 3

Particle Methods for the Kac Model

In [35], Kac introduced an N particle stochastic process model for a spatially homogeneous gas that evolves according to a specific collision process. He proved that, as $N \to \infty$, the marginal probability density function (p.d.f.) for a single particle at time t approaches the solution of a simplified version of the famed Boltzmann equation. The N-dimensional probability density function for all particles is described by the **Kac master equation** which can be derived as a random walk with transitions occurring at times of a Poisson process. In this chapter we take advantage of properties of the Poisson process, including process "thinning" and the conditional uniformity of event times, to efficiently simulate values from the velocity density of any single particle at any given time $t > 0$. We compare our results to those from the familiar Nanbu ([44]), Nanbu-Babovsky ([4]), and DSMC (direct simulation Monte Carlo) ([6]) algorithms. Additionally, we introduce an "ε-perfect sampling algorithm" that uses "coupling from the past" techniques to simulate or sample from the velocity distribution at time $t = \infty$ up to a user defined accuracy ε that can be taken so small as to be essentially "perfect" within the accuracy available by a given machine precision.

In Section 3.1, we review the derivation of the N particle Kac master equation and show its relationship to the Poisson process. In Section 3.2 we very briefly review the probability density function of any one given particle as a marginal density of the joint probability density function solution to the Kac master equation. In Section 3.3 we review three existing simulation methods for the marginal density, introduce our new exact Poisson method, and make an empirical comparison between all methods. In Section 3.4, we briefly review the idea of perfect simulation (perfect sampling) and describe a novel "ε-perfect" algorithm that applies to the Kac model.

3.1 The Kac Master Equation

Consider N one-dimensional particles, each with mass m, with velocities $v_1, v_2, \dots, v_N$. Define $\vec{v} = (v_1, v_2, \dots, v_N)$. The total kinetic energy of the ensemble of particles is

$$E = \frac{m}{2} \sum_{i=1}^{N} v_i^2.$$

For simplicity, we assume that $m = 2$. Thus, we have $E = \sum_{i=1}^{N} v_i^2$ or, equivalently, $|\vec{v}| = \sqrt{E}$.

If we assume that kinetic energy is conserved as these velocities evolve through particle collisions, points $\vec{v}$ with always be found on the sphere $S^{N-1}(\sqrt{E})$. We now describe a particular evolution of velocities, proposed by Kac [35] in two stages as in [8]. The first stage is to describe as a particular discrete time random walk known as the "Kac walk" and the second stage is to modify the random walk so that it is taking place in continuous time.

3.1.1 The Kac Walk

The Kac Walk:

(1) Randomly select a pair of indices (i, j) from $\{1, 2, \dots, N\}$. There are $\begin{pmatrix} N \\ 2 \end{pmatrix}$ ways to do this, so a particular pair (i, j) is selected with probability $\left[\begin{pmatrix} N \\ 2 \end{pmatrix}\right]^{-1}$.

(2) Randomly select a **scattering angle**[1] θ from a density $\rho(\theta)$ on $[0, 2\pi)$. (Often θ is assumed to be uniformly distributed on $[0, 2\pi)$. For an upcoming derivation, it will be convenient to define $\rho(\theta)$ on $[-\pi, \pi)$ and to assume that it is symmetric about 0.)

[1] θ is referred to as a scattering or collision angle as an analogue to the 2- and 3-dimensional cases where the collisions are parameterized by the angle between pairs of particles.

(3) Update the velocities for particles i and j through the rotation

$$(v_i, v_j) \to (v_i', v_j') := (v_i \cos\theta + v_j \sin\theta, -v_i \sin\theta + v_j \cos\theta).$$

Return to Step 1.

It is easy to see that total kinetic energy is conserved by the Kac walk (though momentum is not) and therefore that these three steps create a random walk on $S^{N-1}(\sqrt{E})$. We will now derive the probability density function, f_k for the vector of N velocities after k "collisions" or iterations of the Kac walk.

Let $g : S^{N-1}(\sqrt{E}) \to \Re$ be any continuous function. Define an operator Q_N on such functions as the conditional expectation

$$Q_N g(\vec{v}) := \mathsf{E}[g(\vec{V}_{k+1})|\vec{V}_k = \vec{v}]$$

where $\vec{V}_k$ is the random vector of velocities at the kth iteration of the Kac walk.

Conditioning on the selected angle and particles, this becomes

$$\begin{aligned}
Q_N g(\vec{v}) &= \mathsf{E}[g(\vec{V}_{k+1})|\vec{V}_k = \vec{v}] \\
&= \int_{-\pi}^{\pi} \sum_{i<j} \mathsf{E}\left[g(\vec{V}_{k+1}) \middle| \vec{V}_k = \vec{v}, \begin{array}{c} \text{particles } i \text{ and } j \\ \text{are selected} \end{array}, \begin{array}{c} \text{angle } \theta \text{ is} \\ \text{selected} \end{array} \right] \cdot \left[\binom{N}{2} \right]^{-1} \rho(\theta)\, d\theta \\
&= \int_{-\pi}^{\pi} \sum_{i<j} g(v_1, \ldots, v_{i-1}, v_i', v_{i+1}, \ldots, v_{j-1}, v_j', v_{j+1}, \ldots, v_N) \cdot \left[\binom{N}{2} \right]^{-1} \rho(\theta)\, d\theta \\
&= \int_{-\pi}^{\pi} \sum_{i<j} g(R_{ij}(\theta)\, \vec{v}) \cdot \left[\binom{N}{2} \right]^{-1} \rho(\theta)\, d\theta
\end{aligned}$$

where $R_{ij}(\theta)$ is the $N \times N$ rotation matrix that is the identity matrix altered to have $\cos\theta$ as the (i,i) entry, $\sin\theta$ as the (i,j) entry, $-\sin\theta$ as the (j,i) entry, and $\cos\theta$ as the (j,j) entry.

Now, define random vectors $\vec{V}_0 \sim f_0$ and $\vec{V}_1 \sim f_1$. Note that

$$\mathsf{E}[g(\vec{V}_1)] = \int_S g(\vec{v})\, f_1(\vec{v})\, d\vec{v},$$

where S is shorthand for $S^{N-1}(\sqrt{E})$. Also note that

$$\begin{aligned} \mathsf{E}[g(\vec{V}_1)] &= \mathsf{E}[\mathsf{E}[g(\vec{V}_1)|\vec{V}_0]] \\ &= \textstyle\int_S Q_N g(\vec{v}) f_0(\vec{v})\, d\vec{v} \end{aligned}$$

Thus, we have that

$$\int_S g(\vec{v})\, f_1(\vec{v})\, d\vec{v} = \int_S Q_N g(\vec{v}) f_0(\vec{v})\, d\vec{v}. \tag{3.1}$$

It is straightforward to verify that Q_N is a self-adjoint operator in the sense that

$$\int_S Q_N g(\vec{v}) f(\vec{v})\, d\vec{v} = \int_S g(\vec{v}) Q_N f(\vec{v})\, d\vec{v} \tag{3.2}$$

for all continuous g and for all N dimensional densities f. Therefore, combining (3.1) and (3.2), we have

$$\int_S g(\vec{v}) f_1(\vec{v})\, d\vec{v} = \int_S g(\vec{v})\, Q_N f_0(\vec{v})\, d\vec{v}. \tag{3.3}$$

Since (3.3) holds for all $g : S^{N-1}(\sqrt{E}) \to \Re$, we have that $f_1 = Q_N f_0$. Similarly, the probability density function for the velocities after k collisions is given by

$$f_k = Q_N^k f_0. \tag{3.4}$$

3.1.2 Time and the Poisson Process

As of now, we have a p.d.f. for the N-dimensional vector of velocities evolving at the discrete times of a random walk. We now will allow the velocities to evolve as an N-dimensional continuous time Markov process. The p.d.f. will now be denoted by $f(\vec{v}, t)$.

Kac proposed in [35] that collisions happen in the N-particle ensemble at times of a Poisson process. Specifically, it is assumed that the expected number of particles that collide in a vanishingly small time step Δt is $\lambda N \Delta t$ for some $\lambda > 0$. This means that the expected number of collisions between pairs of particles is $\lambda N \Delta t/2$ and so the assumption is that collisions happen according to a

Poisson process with rate $\lambda N/2$. Among other things, this implies that, as $\Delta t \searrow 0$, the probability of exactly one collision in any time interval $(t, t+\Delta t)$ is $\lambda N \Delta t/2 + o(\Delta t)$, the probability of no collisions is $1 - \lambda N \Delta t/2 + o(\Delta t)$, and the probability of two or more is $o(\Delta t)$. (Here, $o(\Delta t)$ represents a vanishing function $g(\Delta t)$ such that $g(\Delta t)/\Delta t \to 0$ as $\Delta t \searrow 0$.)

Given the velocity density $f(\vec{v}, t)$, at time t, the density at time $t+\Delta t$ as $\Delta t \searrow 0$ is, informally, the mixture density

$$\begin{aligned} f(\vec{v}, t+\Delta t) &= p \cdot (\,\text{previous density after having gone through a collision}\,) \\ &\quad +(1-p) \cdot (\,\text{previous density}\,) \end{aligned}$$

where p is the probability of a collision in the ensemble during the time interval $(t, t+\Delta t)$ as $\Delta t \searrow 0$.

Formally, this is written as

$$\begin{aligned} f(\vec{v}, t+\Delta t) &= (\lambda N \Delta t/2 + o(\Delta t)) \cdot Q_N f(\vec{v}, t) + (1 - \lambda N \Delta t/2 + o(\Delta t)) \cdot f(\vec{v}, t) \\ &= \tfrac{\lambda N \Delta t}{2} Q_N f(\vec{v}, t) + f(\vec{v}, t) - \tfrac{\lambda N \Delta t}{2} f(\vec{v}, t) + o(\Delta t) \end{aligned}$$

which implies that

$$\frac{f(\vec{v}, t+\Delta t) - f(\vec{v}, t)}{\Delta t} = \frac{\lambda N [Q_N - I]}{2} f(\vec{v}, t) + \frac{o(\Delta t)}{\Delta t}$$

where I is the identity operator, and for $g : S^{N-1}(\sqrt{E}) \times \Re \to \Re$,

$$Q_N g(\vec{v}, t) := \mathsf{E}[g(\vec{V}_{k+1}, t) | \vec{V}_k = \vec{v}].$$

Letting $\Delta t \searrow 0$, $\beta = \lambda/2$, and specifying an initial condition, we get the **Kac Master Equation**:

$$\begin{aligned} \tfrac{d}{dt} f(\vec{v}, t) &= \beta N [Q_N - I] f(\vec{v}, t) \\ f(\vec{v}, 0) &= f_0(\vec{v}) \end{aligned}$$

which has solution

$$f(\vec{v},t) = e^{\beta N[Q_N - I]t} f_0 = e^{-\beta N t} \sum_{i=0}^{\infty} \frac{(\beta N t)^i}{i!} Q_N^i f_0$$
$$= \mathsf{E} Q_N^{Z_N} f_0,$$

where $Z_N \sim Poisson(\beta N)$.

3.2 The One Dimensional Kac Equation

For the following derivations and the simulations in Section 3.3.4, we assume that the scattering angle density, $\rho(\theta)$, is the uniform density on the interval $(0, 2\pi)$. We write $\theta \sim \text{unif}(0, 2\pi)$. Consider the marginal density for particle 1,

$$f_N(v_1, t) = \int_{S^{N-2}(\sqrt{E - v_1^2})} f(\vec{v}, t)\, d\vec{v}_{-1} \tag{3.5}$$

where $\vec{v}_{-1} := (v_2, v_3, \ldots, v_N)$. We will make the typical assumption that $E = N$ since the total kinetic energy grows with respect to the number of particles in the system.

Kac [35] showed, under some assumptions[2] on the initial density $f(v, 0)$, that this single particle marginal density converges, as $N \to \infty$, to the solution of a one-dimensional spatially homogeneous Boltzmann-type equation:

$$\frac{df}{dt}(v,t) = \frac{\lambda}{2\pi} \int_{-\infty}^{\infty} \int_0^{2\pi} [f(v',t) f(w',t) - f(v,t) f(w,t)]\, d\theta\, dw, \tag{3.6}$$

where

$$v' = v \cos\theta + w \sin\theta \qquad \text{and} \qquad w' = -v \sin\theta + w \cos\theta.$$

Equation (3.6) is known as the **Kac Equation** or the **Kac-Boltzmann Equation**.

Note that (3.6) may be rewritten as

$$\frac{\partial f}{\partial t} = \frac{1}{\varepsilon} [P(f,f) - f] \tag{3.7}$$

[2] Let ϕ be a one dimensional density. Informally, an n-dimensional density ϕ_n, symmetric in its arguments, is "ϕ-chaotic" if, as $n \to \infty$, any k-dimensional marginal of ϕ_n acts as a product of k copies of ϕ. In [35], Kac proved that if $f^{(N)} = f(\vec{v}, t)$ solves the Kac Master equation, then $f^{(N)}$ is f-chaotic, where f solves the Kac-Boltzmann equation. Thus, we can consider any univariate marginal of $f(\vec{v}, t)$ as $N \to \infty$ as a solution to (3.6). For a more formal explanation, please see [8].

for $\varepsilon = 1/\lambda$, and $P(f,f)$ defined as the **bilinear collision operator**

$$P(f,f) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{0}^{2\pi} f(v',t)f(w',t)\,d\theta\,dw.$$

If the initial distribution $f_0(v) := f(v,0)$ on $(-\infty,\infty)$ is taken to be $f_0(v) = \frac{2}{\sqrt{\pi}}v^2e^{-v^2}$, and if $\lambda = \sqrt{\pi}/2$, the solution to (3.6) is known [37] to be

$$f(v,t) = \frac{1}{\sqrt{\pi}}\left[\frac{3}{2}(1-C(t))\sqrt{C(t)} + (3C(t)-1)(C(t))^{3/2}v^2\right]e^{-C(t)v^2} \tag{3.8}$$

where

$$C(t) = [3 - 2\exp(-\sqrt{\pi}t/16)]^{-1}.$$

It is easy to see that

$$\lim_{t\to\infty} f(v,t) = \frac{1}{\sqrt{3\pi}}e^{-\frac{1}{3}v^2} \qquad \text{for } -\infty < v < \infty. \tag{3.9}$$

We will use (3.8) and (3.9) to test several Monte Carlo algorithms.

3.3 Simulation Methods for $f(v,t)$

In Sections 3.3.1 and 3.3.2 we review existing methods for drawing (simulating) velocity values from the single particle marginal distribution $f_N(v,t)$ for finite t. It has been shown ([6]) that these types of particle methods can be useful for modeling rarefied gas flows, but in 2- or 3-dimensions, computational constraints require the number of particles to be quite small. In Section 3.3.3 we propose a new method that takes advantage of properties of the Poisson collision process. Our goal is to have a method which is accurate and efficient enough to better approximate $f(v,t)$ with manageable values of N.

3.3.1 The Nanbu and Nanbu-Babovsky Algorithms

In [44], Nanbu proposes an algorithm to draw from $f_N(v,t)$ based on the probabilistic interpretation of colliding particles and on the discretization

$$f_{n+1} = (1-\lambda\Delta t)\,f_n + \lambda\Delta t P(f_n,f_n) \tag{3.10}$$

of (3.7). Here, $f_n = f_n(v) = f(v, t_n)$, $t_0 = 0$, and $t_n = t_0 + n\Delta t$ for some small increment $\Delta t > 0$. The interpretation of (3.10) is that, if Δt is small enough so that $0 \leq \lambda\Delta t \leq 1$, "particle 1" (or any fixed given particle)

- fails to undergo a collision and does not have any velocity update with probability $1 - \lambda\Delta t$, or
- experiences a collision with probability $\lambda\Delta t$ and has a new velocity generated according to the collision process described by $P(f, f)$.

For the following algorithm descriptions, we assume that Δt divides into t evenly.

Nanbu's Algorithm

For $i = 1, 2, \ldots, N$, let $v_i^{(n)}$ denote the velocity for particle i at time step $n = 0, 1, 2, \ldots$.

(1) Sample initial velocities, $v_1^{(0)}, v_2^{(0)}, \ldots, v_N^{(0)}$, for all N particles independent and identically distributed (i.i.d.) from $f_0(v)$. Set $s = 0$.

(2) For $i = 1, 2, \ldots, N$,

- With probability $1 - \lambda\Delta t$ do nothing to the velocity of the ith particle.
  Set $v_i^{(n+1)} = v_i^{(n)}$.
- With probability $\lambda\Delta t$,
  - ∗ Randomly select a second particle from among the remaining particles in $\{1, 2, \ldots, N\}$.
  - ∗ Draw an angle value $\theta \sim \rho(\theta)$.
  - ∗ Set $v_i^{(n+1)} = v_i^{(n)} \cos\theta + v_j^{(n)} \sin\theta$

(3) Set $s = s + \Delta t$ and $n = n + 1$. If $s < t$, return to Step 2.

As kinetic energy is not conserved in Nanbu's Algorithm, Babovsky [4] modified the algorithm to update both v_i and v_j as particles i and j collide. Additionally, at each time step, the number of collisions processed is taken to be equal to the expected number of collision pairs in an interval of length Δt. As the expected number of collisions is $\lambda N \Delta t$, the expected number of collision pairs is $\lambda N \Delta t/2$. This is then probabilistically rounded using the following function.

$$Round[x] = \begin{cases} \lfloor x \rfloor & , \text{ with probability } \lfloor x \rfloor + 1 - x \\ \lfloor x \rfloor + 1 & , \text{ with probability } x - \lfloor x \rfloor \end{cases}$$

It is easy to see that, for any given x, the expected value of $Round[x]$ is x.

Nanbu-Babovsky Algorithm

For $i = 1, 2, \ldots, N$, let $v_i^{(n)}$ denote the velocity for particle i at time step $n = 0, 1, 2, \ldots$.

(1) Sample initial velocities, $v_1^{(0)}, v_2^{(0)}, \ldots, v_N^{(0)}$, for all N particles i.i.d. from $f_0(v)$. Set $s = 0$.

(2) For $i = 1, 2, \ldots, N$,

- Select $m = Round[\lambda N \Delta t/2]$.
- Randomly select m pairs, (i, j) of particle indices, without replacement, from among all possible pairs.
- For each selected pair (i, j), draw an angle value $\theta \sim \rho(\theta)$ and set

$$\begin{aligned} v_i^{(n+1)} &= v_i^{(n)} \cos\theta + v_j^{(n)} \sin\theta \\ v_j^{(n+1)} &= -v_i^{(n)} \sin\theta + v_j^{(n)} \cos\theta. \end{aligned}$$

- For the remaining unselected indices k, set $v_k^{(n+1)} = v_k^{(n)}$.

(3) Set $s = s + \Delta t$ and $n = n + 1$. If $s < t$, return to Step 2.

Both the Nanbu and the Nanbu-Babovsky algorithms can be shown to converge to the solution of the discretized Kac equation given by (3.10). Nanbu's algorithms allows each particle to potentially undergo a collision only once per time step and the Nanbu-Babovsky scheme allows each pair of particles to potentially undergo a collision once per time step.

3.3.2 Bird's DSMC Algorithm

The Direct Simulation Monte Carlo (DSMC) approach due to Bird [6] was first proposed in the late 60's and has been shown ([51]) to converge to the solution of (3.7). It differs from the Nanbu-Babovsky algorithm in that any particular collision pair may be selected multiple times in the time interval of length Δt. Since there are $\lambda N \Delta t/2$ collisions expected in that interval, the average time between collisions is taken to be $\Delta t_c := \Delta t/[\lambda N \Delta t/2] = 2/(\lambda N)$.

Bird's DSMC Algorithm

(1) Sample initial velocities, $v_1^{(0)}, v_2^{(0)}, \ldots, v_N^{(0)}$, for all N particles i.i.d. from $f_0(v)$. Set $s = 0$.

(2) Process one collision as follows.

- Select a pair (i, j) of particles at random from among all possible pairs.
- For each selected pair (i, j), draw an angle value $\theta \sim \rho(\theta)$ and set

$$\begin{aligned} v_i^{(n+1)} &= v_i^{(n)} \cos\theta + v_j^{(n)} \sin\theta \\ v_j^{(n+1)} &= -v_i^{(n)} \sin\theta + v_j^{(n)} \cos\theta. \end{aligned}$$

- For the remaining unselected indices k, set $v_k^{(n+1)} = v_k^{(n)}$.

(3) Set $s = s + \Delta t_c$ and $n = n + 1$. If $s < t$, return to Step 2.

Bird's DSMC techniques comprise a much wider class of extremely useful and popular algorithms than the one detailed here. As it stands, the one given here is essentially the Nanbu-Babovsky algorithm where pairs are selected with replacement.

3.3.3 Exact Simulation of $f_N(v, t)$ Based on Properties of the Poisson process

In this section, we simulate exactly from $f_N(v, t)$ using the actual Poisson numbers of collisions between important time points as opposed to the expected numbers of collisions. We do not partition the interval $(0, t)$, as in the previously described algorithms but instead focus on collision times for the fixed "particle 1".

Recall that the N-particle ensemble is assumed to undergo collisions at times of a Poisson process with rate λN. As each collision involves 2 particles, this means that pairs of particles are colliding according to a Poisson process with rate $\lambda N/2$. The probability that particle 1 is involved in any one of these collisions is $(N-1)/\binom{N}{2} = 2/N$. So, the particle 1 collision process is a thinned Poisson process with rate $(\lambda N/2)\cdot 2/N = \lambda$. Therefore, we begin by simulating a Poisson rate λt number of collisions for particle 1 in the time interval $(0,t)$.

Though intercollision times for particle 1 are exponential with rate λt, once the number of collisions during $(0,t)$ is fixed, the times of those collisions are uniformly distributed over the interval $(0,t)$. So, rather than divide up the interval $(0,t)$ into Δt increments, we consider only the times when particle 1 is involved in a collision by first simulating $K \sim Poisson(\lambda t)$, then simulating independent $U_1, U_2, \ldots, U_K$ from the uniform distribution over $(0,t)$ and finally considering the ordered time points $T_1, T_2, \ldots, T_K$ corresponding to the ordered U_i. (For example, $T_1 = \min(U_1, U_2, \ldots, U_K)$ and $T_K = \max(U_1, U_2, \ldots, U_K)$.)

To update the velocity for particle 1 at time T_1, we must choose, at random, a particle from the remaining $N-1$ particles for collision. As this "sub-ensemble" of particles has been evolving over the time interval $(0,T_1)$, we will process $K_1 \sim Poisson(\lambda(N-1)T_1/2)$ collisions involving only these particles without any regard for the times of these collisions and then select one at random for collision with particle 1. Continuing in this manner, we get the following algorithm.

Exact Poisson Algorithm

(1) Sample initial velocities, $v_1^{(0)}, v_2^{(0)}, \ldots, v_N^{(0)}$, for all N particles i.i.d. from $f_0(v)$.

(2) Simulate $K \sim Poisson(\lambda t)$. This is the number of collisions involving particle 1 during $[0,t]$. Suppose that $K = k$.

(3) Simulate $U_1, U_2, \ldots, U_k \overset{iid}{\sim} \text{unif}(0,t)$ collision times for particle 1.

Denote the ordered $U_1, U_2, \ldots, U_k$ as $T_1, T_2, \ldots, T_k$ where $T_1 = \min(U_1, U_2, \ldots, U_k)$ and $T_k = \max(U_1, U_2, \ldots, U_k)$.

(4) Let $T_0 = 0$.

For $i = 1, 2, \ldots, k$,

- Simulate $K_i \sim Poisson(\lambda(N-1)(T_i - T_{i-1})/2)$. This is the number of "non particle 1" collisions happening during the time interval $[T_{i-1}, T_i]$. Suppose that $K_i = k_i$.
- Process k_i collisions for the $(N-1)$ particle ensemble as follows.

  For $j = 1, 2, \ldots, k_i$

  - Select a pair (r, s) of particles at random from among all possible pairs of particles with labels in $\{2, 3, \ldots, N\}$.
  - For each selected pair (r, s), draw an angle value $\theta \sim \rho(\theta)$ and set new velocities as

    $$\begin{aligned} v_r &= v_r \cos\theta + v_s \sin\theta \\ v_s &= -v_r \sin\theta + v_s \cos\theta \end{aligned}$$

    where v_r and v_s on the right-hand side of these equations are the pre-collision velocities for particles r and s.
- Randomly select one particle from $\{2, 3, \ldots, N\}$ and process a collision with particle 1 by assigning new velocities

  $$\begin{aligned} v_r &= v_r \cos\theta + v_s \sin\theta \\ v_s &= -v_r \sin\theta + v_s \cos\theta \end{aligned}$$

  where r is the selected index in $\{2, 3, \ldots, N\}$ and v_1 and v_r on the right-hand side of these equations are the pre-collision velocities for particles 1 and r.

Note that, unlike the Nanbu, Nanbu-Babovsky, and Bird algorithms, we are not using any sort of time counter or partitioning of the time interval $[0, t]$. Furthermore, we are actually processing a Poisson number of collisions, as assumed by the derivation of the Kac equation, in any given time interval rather than simply the expected number of collisions under the Poisson assumption as in Bird's algorithm. Finally, we note that we are saving on collisions by not propagating the system in the time interval $[T_K, t]$ as we are trying to sample the velocity of particle 1 at time t and this is unchanged in this interval. This gives an expected savings of $(N/2)(1 - e^{-\lambda t})$ collisions per simulation repetition.

3.3.4 Simulation Results

In order to assess the performance of the different algorithms, we simulated repeated draws from the velocity distribution of the first particle, v_1, for various values of N, and compared the results to the exact solution (for $N \to \infty$) given by (3.8). Specifically, for each algorithm we simulated N i.i.d. velocities and propagated the ensemble forward according to the algorithms described in Sections 3.3.1, 3.3.2, and 3.3.3, collecting the ultimate values for v_1 at various times t. In all cases, we were able to achieve reasonable accuracy for a surprisingly small number of particles. However, since the accuracy of the Nanbu and Nanbu-Babovsky algorithms both depend on how finely the time interval is partitioned, and since the Nanbu-Babovsky sampling with replacement step is relatively inefficient, we will focus our comparisons mainly on the Nanbu, Bird, and Poisson algorithms.

We will express the accuracy of our numerical results using an estimate of the **total variation norm** (TVN) between the true and estimated distributions. The total variation norm between two probability measures π_1 and π_2 is defined to be

$$||\pi_1 - \pi_2|| = \frac{1}{2} \sup_{A \in \mathcal{B}} |\pi_1(A) - \pi_2(A)|,$$

where $\mathcal{B}$ is the set of Borel sets in the appropriate space. If π_1 and π_2 are discrete measures, then the total variation norm can be written simply as

$$||\pi_1 - \pi_2|| = \frac{1}{2} \sum_{x} |\pi_1(x) - \pi_2(x)|. \tag{3.11}$$

In this chapter, we discretized both the true and simulated densities by a natural histogram binning procedure in order to use (3.11) to give an estimated TVN between the target density and our simulation results. As this estimate depends on the bin widths used, the actual values of our estimated TVN are not as interesting as the relative values. We will keep our bin widths constant throughout.

For example, for $t = 2$, using Nanbu's algorithm with $N = 5$ produced distributions for v_1 shown in Figure 3.1. In both cases the "target density" given by (3.8) is shown as a solid curve. It is no surprise that the smaller simulation time step gives better results, but to illustrate the

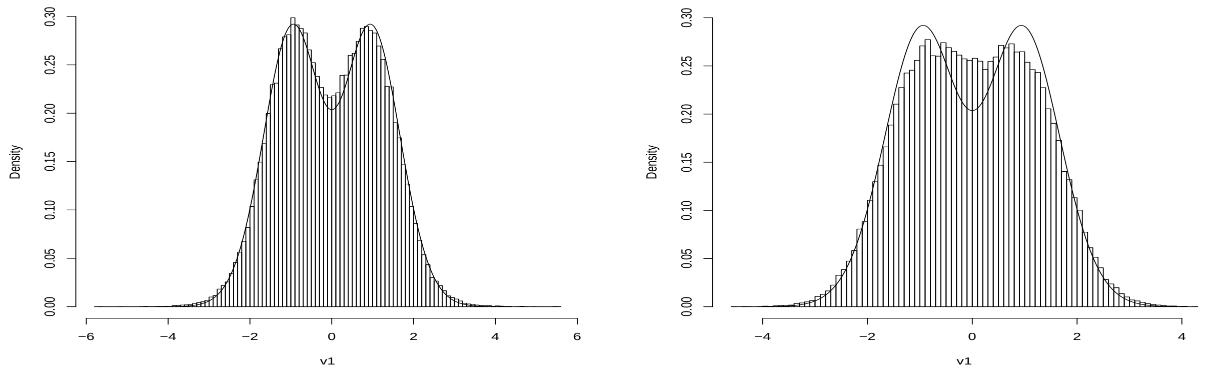


Figure 3.1: Histograms of 100,000 values of v_1 at $t = 2$ with $N = 5$ using Nanbu's Algorithm for $\Delta t = 0.01$ (left) and $\Delta t = 1.0$ (right). Curve is exact solution for $N = \infty$ given by (3.8).

TVN error, the estimated values of TVN for the left and right graphs are 0.0187 and 0.0482, respectively. Figure 3.2 shows estimated TVN, for several values of N, as the simulation time step size decreases. Each point on the graph was created by averaging 100 values of TVN, where each estimated distribution was created from 100,000 realizations of v_1. The height of the horizontal line just below 0.01 shows the average TVN for 100 samples of size 100,000 for Bird's DSMC algorithm (which doesn't depend on Δt) for $N = 50$. This was indistinguishable to several decimal places from similarly estimated TVN for larger N, for the Poisson algorithm for $N \geq 50$, and for a baseline estimated TVN for samples of size 100,000 from the standard normal distribution. (Recall that we do not expect the estimated TVN to necessarily approach zero even in the case of exact simulation from the normal distribution due to the histogram binning discretization used in its calculation.) Figure 3.3 shows the average TVN values (again averaging 100 values each computed from a sample of size 100,000) for the three algorithms as a function of the number of particles in the simulations. Here we can see that the Poisson algorithm consistently performs better than both the Nanbu and Bird algorithms for small N. That is, less particles are needed in order to approximate (3.8) when using the Poisson algorithm. Distributions for v_1, estimated by the DSMC and Poisson algorithms are shown in Figure 3.4. While they are similar both visually and by the TVN, the Poisson algorithm appears to be performing slightly better in the tails, as shown in Figure 3.5 for the case of the upper tails.

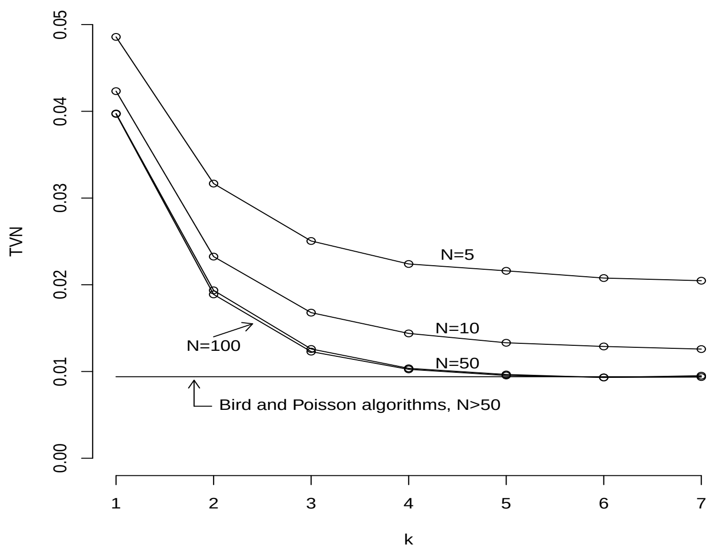


Figure 3.2: Estimated TVN for Nanbu's Algorithm as a function of k for $\Delta t = 2/2^k$

3.4 An ε-Perfect Simulation of $f(v, \infty)$

"Perfect sampling" (or "perfect simulation") is the name applied to a class of Markov chain Monte Carlo (MCMC) algorithms which enable one to draw values exactly from the stationary (limiting) distribution π of a Markov chain. This is in contrast to the more typical MCMC algorithms in which transitions are simulated for "a long time" after which output is collected giving only approximate draws from π.

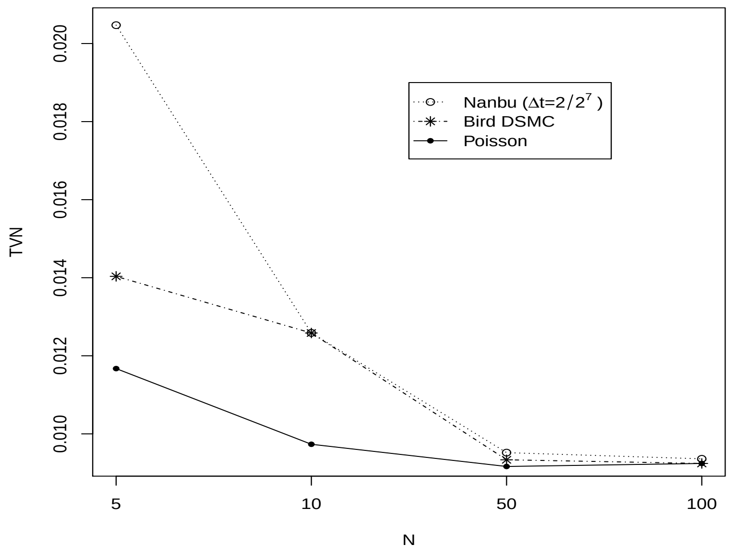


Figure 3.3: Estimated TVN for the Nanbu, Bird, and Poisson algorithms as a function of N

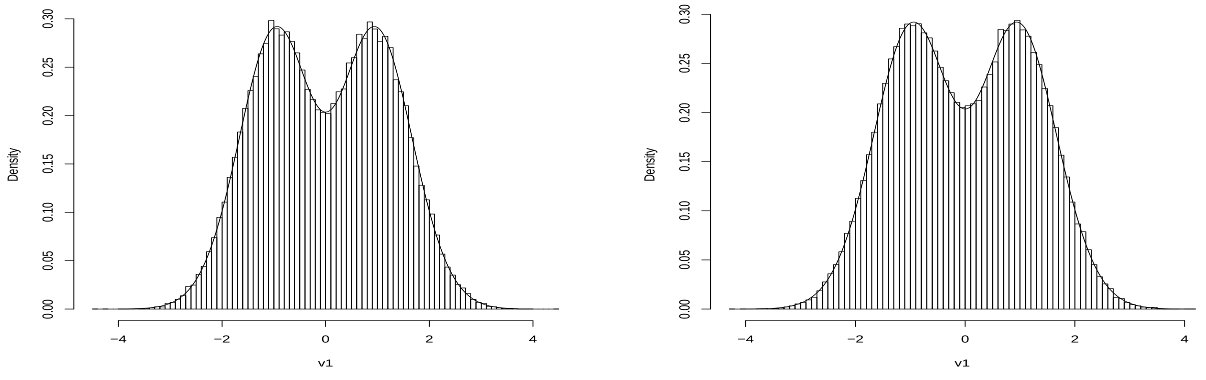


Figure 3.4: Histograms of 100,000 Values of v_1 at $t = 2$ with $N = 50$ for the DSMC algorithm (left) and the Poisson algorithm (right).

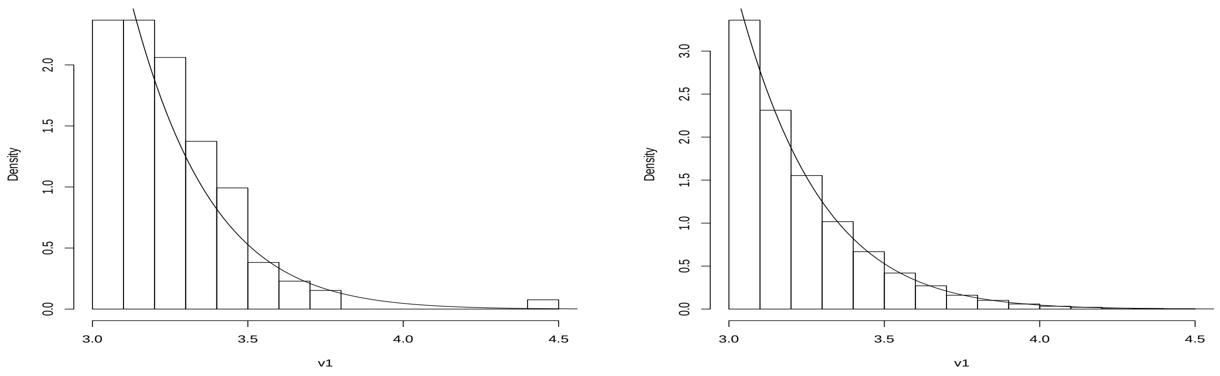


Figure 3.5: Histograms of 100,000 upper tail values of v_1 at $t = 2$ with $N = 1000$ using Bird's DSMC algorithm (left) and the Poisson algorithm (right).

3.4.1 Perfect Simulation

The essential idea of most of these approaches is to find a random epoch $-T$ in the past such that, if one constructs sample paths (according to a transition density $p(x, y)$ that has stationary/limiting distribution π) from every point in the state space starting at $-T$, all paths will have met or "coupled" by time zero. The common value of the paths at time zero is a draw from π. Intuitively, it is clear why this result holds with such a random time T. Consider a chain starting anywhere in the state space at time $-\infty$. At time $-T$ it must pick *some* value x, and from then on it follows the trajectory from that value. Since it arrives at the same place at time zero, no matter what value x is picked at time $-T$, the value returned by the algorithm at time zero is the tail end of a path that has run for an infinitely long time and is therefore a "perfect" draw from π.

Typically, perfect sampling is carried out using **stochastically dominating processes** that bound or "sandwich" the process of interest for all time as depicted in Figure 3.6. At time $-T$, these bounding processes are at the extremes of the state space. At time $-b$, they have coupled together to form a single path that is followed forward to time 0. If such coupling is not achieved before time 0, T is increased and the bounding processes are resimulated using the same random inputs whenever they pass through previously visited time points. Note that when the bounding processes couple at some random time $-b$, all possible sample paths of the process of interest have necessarily coupled at some earlier random time $-a$. Note further that a path started anywhere in the state space at time $-T$ will be at the same place at time 0. This value at time 0 is a perfect draw from the stationary distribution of the Markov chain. Time $-T$ is known as a **backward coupling time**.

Forward coupling of sample paths does not yield the same result for reasons that we will not go into here. For more information on perfect sampling algorithms, we refer the reader to [9].

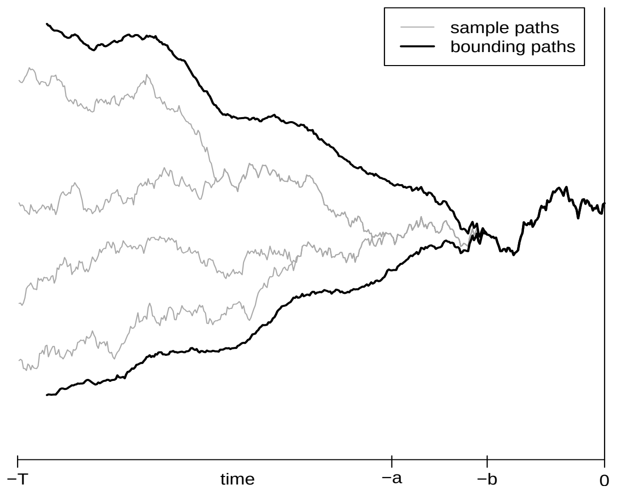


Figure 3.6: Depiction of stochastically dominating (bounding) processes in perfect simulation.

3.4.2 ε-Perfect Simulation for the Kac Model

We say that an algorithm is "ε-perfect" if the distance between the bounding processes depicted in Figure 3.6 is monotonically decreasing and if this distance is less than ε at time 0. In this Section we will describe an ε-perfect algorithm for sampling from $f(v, \infty)$ that is interesting mainly as a proof of concept for the approach that we hope will eventually be adapted to and useful for more interesting models.

Consider, for the purpose of illustration, the case where we have only $N = 3$ particles. Simulating i.i.d. velocities v_1, v_2, and v_3 from $f_0(v)$ gives a kinetic energy $E = v_1^2 + v_2^2 + v_3^2$. As we propagate these 3 velocities forward in time, we create a realization of a 3-dimensional Markov chain that lives on the surface of a 3-dimensional sphere with radius $\sqrt{E}$. For the simple model considered in this chapter, we can, without loss of generality, consider the velocity vector as a random walk on the sphere restricted to the first octant where $v_1, v_2, v_3 > 0$ and at the end of our simulation attach a negative sign to each velocity independently of the others with probability $1/2$.

For the purpose of achieving a coupling of sample paths walking on the sphere, we note that $v_i \cos\theta + v_j \sin\theta$ can be written as

$$v_i \cos\theta + v_j \sin\theta = A\,\sin(\theta + \varphi)$$

for some A and φ. In the case where $v_j = 0$, we may take $A = v_i$ and $\varphi = \pi/2$. In the case that $v_j \neq 0$, we may take $A = \sqrt{v_i^2 + v_j^2}$ and $\varphi = \tan^{-1}(v_i/v_j)$. Suppose that the current position of the random walk is (v_1, v_2, v_3) and that the particles randomly selected for updating are indexed by $(i, j) = (1, 2)$. Note then that $A = \sqrt{E - v_3^2}$ and, since v_2 is nonzero with probability 1, the new velocity for particle 1 is given by

$$v_1' = \sqrt{E - v_3^2}\ \sin(\theta + \varphi) \tag{3.12}$$

where $\theta \sim \text{unif}(0, 2\pi)$ and $\varphi = \tan^{-1}(v_1/v_2)$. Since $\theta + \varphi$ is uniformly distributed over an interval of length 2π, $\theta + \varphi(\mod 2\pi) \sim \text{unif}(0, 2\pi)$, and the distribution of v_1' in (3.12) is the same as that of $\sqrt{E - v_3^2}\ \sin(\theta)$. Thus, we will update v_1 as

$$v_1' = \sqrt{E - v_3^2}\ \sin(\theta) \tag{3.13}$$

where $\theta \sim \text{unif}(0, 2\pi)$ and then v_2 by setting

$$v_2' = \sqrt{E - v_3^2 - (v_1')^2}. \tag{3.14}$$

Four different values for (v_1, v_2, v_3) are visualized as points on the sphere in the first octant in Figure 3.7, as well as an angle θ drawn uniformly over $(0, 2\pi)$. All four points, **and indeed all points on the entire surface**, will map onto the depicted arc.

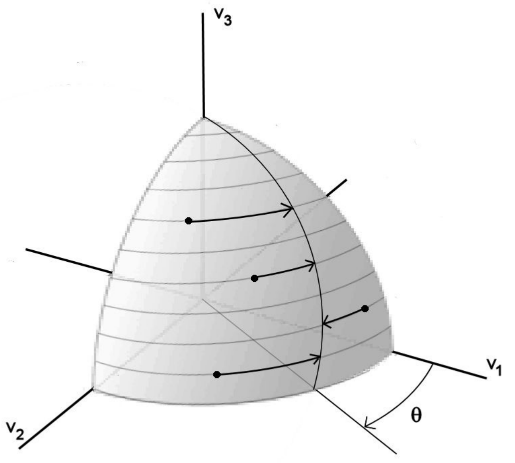
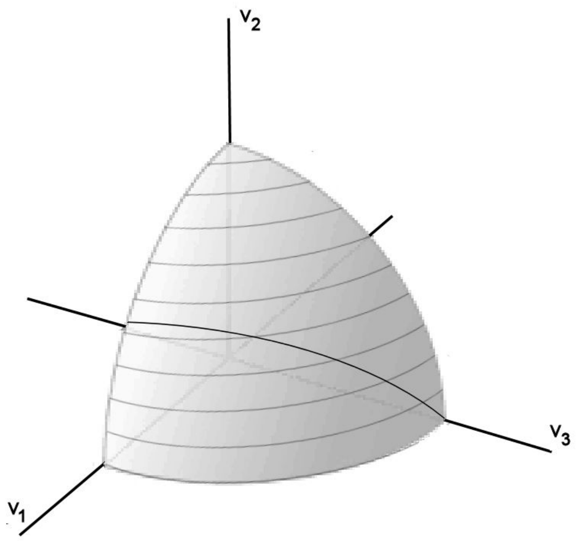


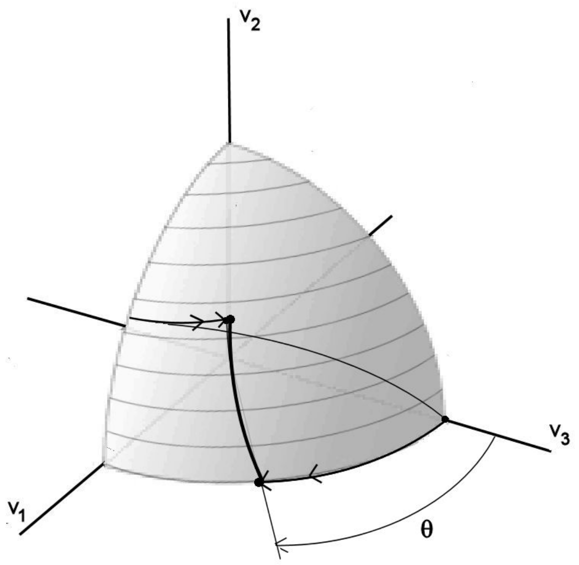
Figure 3.7: Visualization of sample paths of the "Perfect Kac Walk" on a sphere in the first octant.

Suppose now that the next particles randomly selected for updating are indexed by $(i, j) = (3, 1)$. Given a new angle θ drawn uniformly over $(0, 2\pi)$, the points on the arc from Figure 3.7 will map to the smaller arc segment depicted in Figure 3.8.

It is easy to see that if the same indices (in the same or different order) are selected twice in a row, the arc will maintain its length and move "laterally". However, if at least one index is different from those in the previous step, the arc segment will, necessarily, shrink in length. In other words, we will be mapping all points on the sphere closer and closer together. Thus, the algorithm is closing in on a single point on the surface. Formally, consider N particles with all



(a) Previous arc with rotated axes



(b) New angle and arc mapped to smaller arc

Figure 3.8: Arc from Figure 3.7 mapped to shorter arc.

positive velocities. Suppose two realizations of the velocities are given by $\vec{u}$ and $\vec{v}$, both living in the first orthant. Applying a collision between particles i and j, we get

$$\begin{aligned}
\|R_{ij}(\theta)\vec{u} - R_{ij}(\theta)\vec{v}\|_2^2 &= \Big[(u_1 - v_1)^2 + \cdots + \left(\sqrt{u_i^2 + u_j^2}\sin\theta - \sqrt{v_i^2 + v_j^2}\sin\theta\right)^2 + \cdots \\
&\quad + \left(\sqrt{u_i^2 + u_j^2}\cos\theta - \sqrt{v_i^2 + v_j^2}\cos\theta\right)^2 + \cdots + (u_N - v_N)^2\Big] \\
&= \Big[(u_1 - v_1)^2 + \cdots + \left(u_i^2 + u_j^2 - 2\sqrt{u_i^2 + u_j^2}\sqrt{v_i^2 + v_j^2} + v_i^2 + v_j^2\right)\sin^2\theta + \cdots \\
&\quad + \left(u_i^2 + u_j^2 - 2\sqrt{u_i^2 + u_j^2}\sqrt{v_i^2 + v_j^2} + v_i^2 + v_j^2\right)\cos^2\theta + \cdots + (u_N - v_N)^2\Big] \\
&\le \Big[(u_1 - v_1)^2 + \cdots + \left(u_i^2 + u_j^2 - 2(u_iv_i + u_jv_j) + v_i^2 + v_j^2\right)\sin^2\theta + \cdots \\
&\quad + \left(u_i^2 + u_j^2 - 2(u_iv_i + u_jv_j) + v_i^2 + v_j^2\right)\cos^2\theta + \cdots + (u_N - v_N)^2\Big] \\
&= \Big[(u_1 - v_1)^2 + \cdots + \left((u_i - v_i)^2 + (u_j - v_j)^2\right)\sin^2\theta + \cdots \\
&\quad + \left((u_i - v_i)^2 + (u_j - v_j)^2\right)\cos^2\theta + \cdots + (u_N - v_N)^2\Big] \\
&= \Big[(u_1 - v_1)^2 + \cdots + (u_i - v_i)^2 + \cdots + (u_j - v_j)^2 + \cdots + (u_N - v_N)^2\Big] \\
&= \|\vec{u} - \vec{v}\|_2^2,
\end{aligned}$$

where equality only occurs when $(u_i, u_j) = c(v_i, v_j)$, for some constant c. This will occur anytime that the same pair of particles is chosen multiple times in a row, but assuming that a new pair is chosen, the coupled collision acts as a contraction on the portion of the sphere living in the first orthant.

To formalize the ε-perfect sampling algorithm for general N, we will follow the N corner points $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_N$, where $\vec{c}_i = \sqrt{E}\,\vec{e}_i$ and $\vec{e}_i$ is the ith standard basis vector for $\mathbb{R}^N$.

ε-Perfect Algorithm

Set an error tolerance $\varepsilon > 0$ and set $n = 1$.

(1) Simulate and store $\theta_{-n} \sim \text{unif}(0, \pi/2)$.

Randomly select a pair of distinct indices from $\{1, 2, \ldots, N\}$ and store as a 2×1 vector $\vec{P}_{-n}$

with entries denoted as $\vec{P}_{-n}(1)$ and $\vec{P}_{-n}(2)$. (Order is important. For example $\vec{P}_{-n} = (3,5)$ is different from $\vec{P}_{-n} = (5,3)$.)

Set $\vec{c}_i = \sqrt{E}\,\vec{e}_i$ for $i = 1, 2, \ldots, N$.

Let $t = n$.

(2) Let $\vec{c}_i(j)$ denote the jth element of $\vec{c}_i$.

For $i = 1, 2, \ldots, N$, let $e = [c_i(P_{-t}(1))]^2 + [c_i(P_{-t}(2))]^2$ and set

$$\vec{c}_i(P_{-t}(1)) = \sqrt{e}\ \sin\theta_{-t}$$

and

$$\vec{c}_i(P_{-t}(2)) = \sqrt{e - [\vec{c}_i(P_{-t}(1))]^2}$$

If $t = 1$, go to Step 3. Otherwise, let $t = t - 1$ and return to the beginning of Step 2.

(3) Compute the Euclidean distance between all pairs of corner points. Let D be the maximum of these distances. If $D < \varepsilon$, compute the average corner point

$$\vec{c} := \frac{1}{N}\sum_{i=1}^{n} \vec{c}_i,$$

multiply each coordinate of $\vec{c}$ by -1 with probability $1/2$ independent of all other coordinates, output the new $\vec{c}$, and exit the algorithm. Otherwise, set $n = n + 1$ and return to Step 1.

In practice, it is more efficient, in the search for a backward coupling time, to step back further than one step at a time. It is generally advised to let $t = 2^n$ in, what is in this case, Step 1 of the ε-perfect algorithm. For simplicity, we use the Euclidean distance between corner points as a surrogate for arc length distance on the surface of the sphere since convergence to zero of the former implies convergence to zero of the latter.

In Figure 3.9 we show the results for 100,000 draws of v_1 using the ε-perfect algorithm with $N = 50$ and $\varepsilon = 10^{-6}$ along with the curve given by (3.9). For this simulation run, the mean backward coupling time was 948.2 with a minimum of 422 and a maximum of 1811.

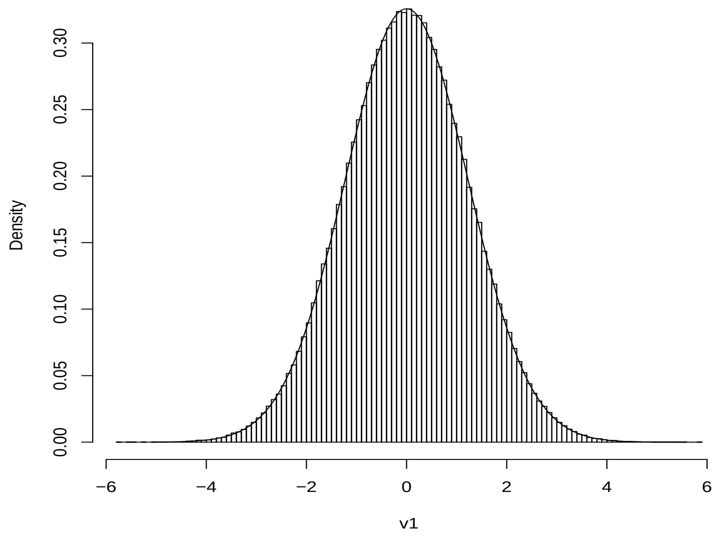


Figure 3.9: Histogram of 100, 000 draws of v_1 using the ε-perfect algorithm with $N = 50$ and $\varepsilon = 10^{-6}$. The solid curve represents the true solution.

Chapter 4

Uniform Sampling of DAGs

4.1 Introduction

Directed acyclic graphs (DAGs) are collections of nodes and directed edges such that there are no directed paths from a node to itself. DAGs are often used to model the flow of information and form the underlying structure of Bayesian Networks [48]. As such, testing algorithms for learning Bayesian Networks and making inferences on structural properties of DAGs leads to a problem which is very simple to state but much more difficult to solve: How can a DAG be uniformly selected from the set of all DAGs with n nodes? Direct enumeration is not possible since the number of DAGs grows super-exponentially as a function of the number of nodes. Table 4.1 lists the number of DAGs for 1-9 nodes.

Recent work ([32],[38],[42]) on this problem has relied mostly on Markov chains with unknown mixing times or asymptotic approximations of recursive formulas for counting DAGs. This chapter

# of nodes	# of DAGs
1	1
2	3
3	25
4	543
5	29,281
6	3,781,503
7	1,138,779,265
8	783,702,329,343
9	1,213,442,454,842,881

Table 4.1: Number of Directed Acyclic Graphs with 1-9 nodes

explores some perfect sampling techniques to sample uniformly from the set of DAGs on a given number of nodes.

4.2 Forward Markov Chain

Melancon et al [42] proposed a very simple Markov chain whose stationary distribution is uniform over the set of DAGs. The chain goes as follows

(1) Start with any initial DAG, X_0 and let $t = 0$.

(2) Select a pair of nodes i and j.

- If edge $i \to j$ is in X_t, set $X_{t+1} = X_t \setminus \{i \to j\}$.
- If edge $i \to j$ is not in X_t and adding it leaves the graph acyclic, set $X_{t+1} = X_t \cup \{i \to j\}$.
- Otherwise, set $X_{t+1} = X_t$.

(3) Set $t = t + 1$ and go back to step (2).

It is easy to check that this leads to an irreducible aperiodic Markov chain, and that the corresponding transition matrix is symmetric, which guarantees the desired uniform stationary distribution. Melancon et al further argue that the algorithm can produce an approximately uniform sample in $O(n^3 \log n)$ time.

4.3 Perfect Sampling Chains

The general concept of perfect sampling was reviewed in Section 3.4.1. Here, we apply this idea in two separate ways. The first is through constructing a regeneration set that facilitates coalescence of coupled chains. The second is a more direct application of stochastic domination processes applied to the forward chain discussed previously.

4.3.1 Regenerative DAG Coupler

Let $\mathbb{G}_n$ be the set of DAGs on n nodes. Denote $\tilde{\mathbb{G}}_n \subset \mathbb{G}_n$ the set consisting of DAGs with either no edges or $\binom{n}{2}$ edges (which is the maximum possible number of edges). Note that $n!$ unique DAGs with $\binom{n}{2}$ edges can be constructed by first choosing an ordering of the nodes, and then for every node, placing edges to every node that follows it in the ordering. Also, since the adjacency matrix of any DAG with $\binom{n}{2}$ edges is a permutation of the fully filled in lower triangular matrix, there are at most $n!$ such DAGs. $\tilde{\mathbb{G}}_n$ thus contains $n! + 1$ elements. We want to construct a Markov chain that will uniformly sample from $\mathbb{G}_n$, but to do so we will work with an extended state space of $(\mathbb{G}_n \times \{T\}) \cup (\tilde{\mathbb{G}}_n \times \{R\})$. This extended state space corresponds to target states T, and regeneration states R.

For ease of notation, we will enumerate this extended state space in the following way

- Let 1 represent the empty graph in $\mathbb{G}_n \times \{T\}$.
- Let $|\mathbb{G}_n| - n! + 1, |\mathbb{G}_n| - n! + 2, \ldots, |\mathbb{G}_n|$ represent the graphs with maximal number of edges in $\mathbb{G}_n \times \{T\}$.
- Fill in the values in between 1 and $|\mathbb{G}_n| - n! + 1$ in order of increasing number of edges.
- Let $|\mathbb{G}| + 1$ be the empty graph in $\tilde{\mathbb{G}}_n \times \{R\}$.
- The remaining $n!$ values correspond to the graphs with maximal number of edges in $\tilde{\mathbb{G}}_n \times \{R\}$.

With this enumeration, we are attempting to draw from the distribution

$$\pi = [p_T \pi_T \mid p_R \pi_R]$$

$$\pi_T \propto 1$$

using a transition matrix of the form

$$M = \begin{bmatrix} M_{T,T} & M_{T,R} \\ M_{R,T} & M_{R,R} \end{bmatrix}$$

with sizes

$$\begin{aligned} M_{T,T}: &\quad |\mathbb{G}_n| \times |\mathbb{G}_n| \\ M_{T,R}: &\quad |\mathbb{G}_n| \times (n!+1) \\ M_{R,T}: &\quad (n!+1) \times |\mathbb{G}| \\ M_{R,R}: &\quad (n!+1) \times (n!+1) \end{aligned}$$

Since we want to force coalescence through a regeneration, we choose $M_{T,R}$ to be of the form

$$M_{T,R} = c_{T,R} \begin{bmatrix} p_e & p_f & \cdots & p_f \\ \vdots & \vdots & & \vdots \\ p_e & p_f & \cdots & p_f \end{bmatrix}$$

where p_e is the probability of regenerating at the empty graph, and p_f is the probability of regenerating at a full graph. These should satisfy

$$p_e + (n!)p_f = 1.$$

The constant $c_{T,R}$ represents the unconditional probability of moving to a regenerative state. Furthermore, we can let $M_{R,R}$ be of the form

$$M_{R,R} = c_{R,R} \begin{bmatrix} p_e & p_f & \cdots & p_f \\ \vdots & \vdots & & \vdots \\ p_e & p_f & \cdots & p_f \end{bmatrix}$$

since moving from one regenerative state to another should be itself again be a regeneration. The constant $c_{R,R}$ then represents the probability of staying in the regeneration set, $\tilde{\mathbb{G}}_n \times \{R\}$.

The choices for the blocks $M_{T,T}$ and $M_{R,T}$ are less constrained. They need to be constructed in a way that allows for every state to be reached, while still being easy to simulate. The simplest choice for $M_{R,T}$ is then

$$M_{R,T} = c_{R,T} \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & & 0 \\ \vdots & \vdots & & \vdots & & \ddots & \\ 0 & 0 & \cdots & 0 & 0 & & 1 \end{bmatrix}$$

where, with probability $c_{R,T}$, the chain moves to the target set and the graph remains the same.

Instead of trying to define $M_{T,T}$, we examine the requirements there are for the coefficients $c_{i,j}$ and p_i, $i, j \in \{T, R\}$.

$$c_{T,T} + c_{T,R} = 1 \tag{4.1}$$

$$c_{R,T} + c_{R,R} = 1 \tag{4.2}$$

$$p_T + p_R = 1 \tag{4.3}$$

Stationarity requires that

$$\begin{aligned} p_T \pi_T &= p_T c_{T,T} \pi_T M_{T,T} + p_R c_{R,T} \pi_R C_{R,T} \\ &= p_T c_{T,T} \pi_T M_{T,T} + p_R c_{R,T} [\begin{array}{cccccc} p_e & 0 & \cdots & 0 & p_f & \cdots & p_f \end{array}] \end{aligned} \tag{4.4}$$

and

$$\begin{aligned} p_R \pi_R &= p_T c_{T,R} \pi_R M_{T,R} + p_R c_{R,R} \pi_R M_{R,R} \\ &= p_T c_{T,R} [\begin{array}{cccc} p_e & p_f & \cdots & p_f \end{array}] + p_R c_{R,R} [\begin{array}{cccc} p_e & p_f & \cdots & p_f \end{array}] \\ &= (p_T c_{T,R} + p_R c_{R,R}) \pi_R \end{aligned} \tag{4.5}$$

Combining together (4.2), (4.3), and (4.4), we get

$$p_R = \frac{c_{T,R}}{c_{T,R} + c_{R,T}} \tag{4.6}$$

and then using (4.6) along with (4.1) and (4.3) we have

$$p_T = \frac{c_{R,T}}{c_{T,R} + c_{R,T}} \tag{4.7}$$

Now, substituting (4.6) and (4.7) into (4.5) and requiring non-negativity of probabilities, we get

$$c_{R,T} \leq \frac{\min(1/p_e, 1/p_f)}{|\mathbb{G}_n|} \tag{4.8}$$

which is maximized when $p_e = p_f = \frac{1}{n!+1}$. Thus, the best we can hope for is to have a regeneration probability of $\frac{n!+1}{|\mathbb{G}_n|}$, which leads to an average regeneration time of at least $\frac{|\mathbb{G}_n|}{n!+1}$.

4.3.2 DAG Coupling from the Past

Instead of forcing coalescence to take place through regenerations, we can design our Markov chain to be able to be coupled in a way to allow natural coalescence to take place. For this, some modifications to the original forward chain are needed. One of the biggest issues with using the chain as described earlier is the fact that two instances of the chain may differ in the existence of one edge. The chain whose graph has the edge will remove it if selected, while the chain whose graph does not have the edge will attempt to add that edge. The only way the two chains will eventually agree on that edge is if the edge becomes invalid for the chain that does not contain the edge, which is a strong requirement.

In order to avoid this alternating behavior, we modify the chain to first select the type of attempted move, along with the pair of edges. This leads to the following algorithm

(1) Start with any initial DAG, X_0 and let $t = 0$.

(2) Select a pair of nodes i and j.

(3) Generate a uniform draw $U \sim \text{unif}(0, 1)$.

- If $U \le \frac{1}{2}$,
    - ∗ If edge $i \to j$ is in X_t, set $X_{t+1} = X_t \setminus \{i \to j\}$.
    - ∗ Otherwise, set $X_{t+1} = X_t$.
- If $U > \frac{1}{2}$,
    - ∗ If edge $i \to j$ is not in X_t and adding it leaves the graph acyclic, set $X_{t+1} = X_t \cup \{i \to j\}$.
    - ∗ Otherwise, set $X_{t+1} = X_t$.

(4) Set $t = t + 1$ and go back to step (2).

The second modification comes from the fact that edge additions and deletions are not the only possible moves. Allowing edge reversals is known ([26]) to improve mixing at least in the context of Bayesian Networks. Incorporating these edge reversals, as well as making some minor tweaks, we get the forward chain

(1) Start with any initial DAG, X_0 and let $t = 0$.

(2) Select an ordered pair of nodes (i, j).

(3) Generate a uniform draw $U \sim \text{unif}(0, 1)$.

- If $U \leq \frac{1}{3}$,
  - ∗ If edge $i \to j$ is in X_t, set $X_{t+1} = X_t \setminus \{i \to j\}$.
  - ∗ If edge $j \to i$ is in X_t, set $X_{t+1} = X_t \setminus \{j \to i\}$.
  - ∗ Otherwise, set $X_{t+1} = X_t$.
- If $\frac{1}{3} < U \leq \frac{2}{3}$,
  - ∗ If edge $i \to j$ is in X_t and the removal of it allows for the addition of edge $j \to i$, then set $X_{t+1} = (X_t \setminus \{i \to j\}) \cup \{j \to i\}$.
  - ∗ If edge $i \to j$ is not in X_t and adding edge $\{j \to i\}$ leaves the graph acyclic, set $X_{t+1} = X_t \cup \{j \to i\}$.
  - ∗ Otherwise, set $X_{t+1} = X_t$.
- If $\frac{2}{3} < U$,
  - ∗ If edge $j \to i$ is in X_t and the removal of it allows for the addition of edge $i \to j$, then set $X_{t+1} = (X_t \setminus \{j \to i\}) \cup \{i \to j\}$.
  - ∗ If edge $j \to i$ is not in X_t and adding edge $\{i \to j\}$ leaves the graph acyclic, set $X_{t+1} = X_t \cup \{i \to j\}$.
  - ∗ Otherwise, set $X_{t+1} = X_t$.

(4) Set $t = t + 1$ and return to step 2.

The last piece is then to couple together multiple instances of the chain from all possible starting graphs, and check for coalescence. Obviously, we cannot enumerate and check every single possible graph, so instead, we will keep track of whether an edge is in every chain, possibly some of the chains, or none of the chains. This is done through introducing a **mark**, $M_{i,j} \in \{0, ?, 1\}$, to every edge $\{i \to j\}$. A mark of $M_{i,j} = 1$ means that every instance of the chain contains the edge $\{i \to j\}$, a mark of $M_{i,j} = 0$ means that none of the chains contain the edge, and a mark of $M_{i,j} =?$ means that some of the chains may possibly contain the edge. With these marks, we can then easily check to see if all of the chains have coalesced. If all of the marks are either 0 or 1, then the graph of all of the chains is the same. Putting this into the coupling from the past framework, our new DAG CFTP Algorithm is as follows.

(1) Set $T = 1$.

(2) Initialize the marks $M_{i,j} =?$ for all $i \neq j$.

(3) Sample an ordered pair of indices $(i, j)_{-T}$.

(4) Sample $U_{-T} \sim \text{unif}(0, 1)$.

(5) For $t = T, T - 1, \ldots, 1$,

- If $U_{-t} \leq 1/3$,
    - ∗ Set $M_{i_{-t},j_{-t}} = 0$ and $M_{j_{-t},i_{-t}} = 0$.
- If $1/3 < U_{-t} \leq 2/3$,
    - ∗ If the DAG consisting of edges with $M_{k,\ell} = 1$ prohibits adding edge $j \to i$ or reversing edge $i \to j$, do nothing.
    - ∗ If one or more edges with $M_{k,\ell} =?$ prohibits adding edge $j \to i$ or reversing edge $i \to j$, set $M_{i,j} =?$ and $M_{j,i} =?$.
    - ∗ Otherwise, set $M_{i,j} = 0$ and $M_{j,i} = 1$.
- If $2/3 < U_{-t}$,
    - ∗ If the DAG consisting of edges with $M_{k,\ell} = 1$ prohibits adding edge $i \to j$ or reversing edge $j \to i$, do nothing.

- $*$ If one or more edges with $M_{k,\ell} =?$ prohibits adding edge $i \to j$ or reversing edge $j \to i$, set $M_{j,i} =?$ and $M_{i,j} =?$.
- $*$ Otherwise, set $M_{j,i} = 0$ and $M_{i,j} = 1$.

(6) If $M_{i,j} \neq ?$ for all $i \neq j$, then all the chains have coalesced, and M corresponds to the adjacency matrix of uniformly sampled DAG.

Otherwise, set $T = T + 1$ and return to Step (2).

4.4 Results

We first want to verify that our DAG CFTP Algorithm is truly returning a uniform sample, as it theoretically should. With $n = 3$ nodes, there are only 25 DAGs which makes enumeration possible. So we start with generating uniform DAGs with 3 nodes. A histogram of 1,000,000 draws is shown in Figure 4.1.

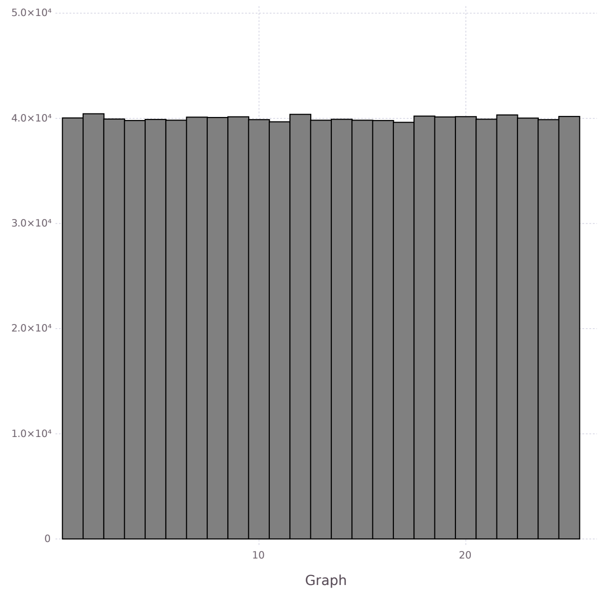


Figure 4.1: Histogram of 1,000,000 draws from the uniform distribution on the set of DAGs with 3 nodes. Graphs are ordered in order of increasing number of edges.

It is then important to look at how the algorithm scales as the number of nodes increases. We ran the DAG CFTP for $n = 3, 4, \ldots, 9$ nodes, producing 1,000 uniform samples for each. The backward coupling time (bct) was also returned for each sample, allowing the average backward coupling time to be estimated. Figure 4.2 shows the average backward coupling time along with the theoretical average regeneration time from the Regenerative DAG Coupler.

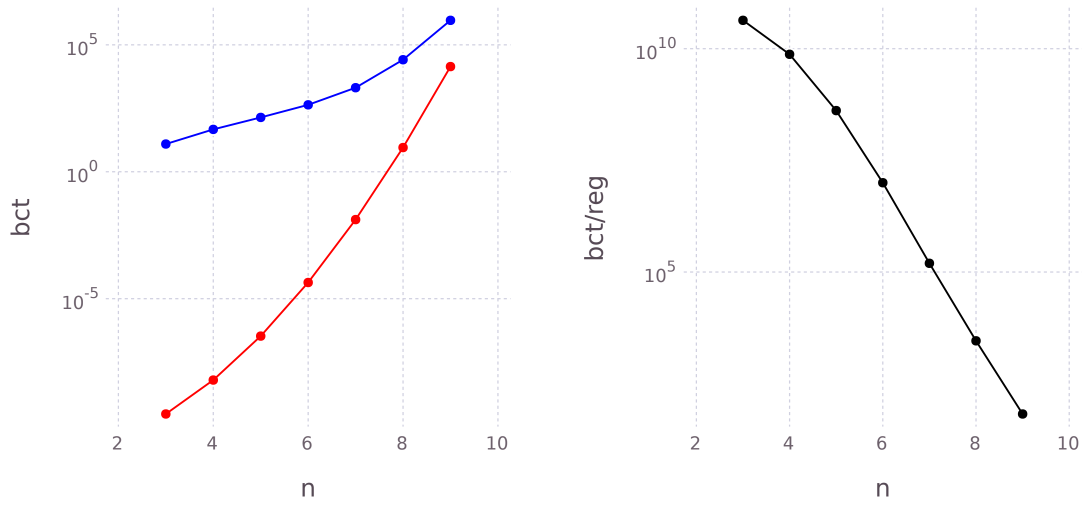


Figure 4.2: (Left) Empirical average backward coupling time in blue and theoretical regeneration time in red as a function of the number of nodes. (Right) Ratio of empirical coupling time to theoretical regeneration time

The ratio of the average backward coupling time over the regeneration time appears to be approaching an exponential (linear in log scale). Using the last several data points, we can approximate the average backward coupling time to be

$$\text{bct} \underset{\sim}{\propto} \frac{|\mathbb{G}_n|}{n! + 1} e^{-3.943n}$$

Although the DAG CFTP Algorithm performs better than the theoretical Regenerative DAG Coupler, it still scales at a rate which makes the algorithm computationally infeasible for reasonable numbers of nodes. Approximate methods are still required for larger numbers of nodes as is of practical interest.

Chapter 5

Bayesian Network Structure Inference

Bayesian networks (Pearl [48]) are convenient graphical expressions for high dimensional probability distributions representing complex relationships between a large number of random variables. A Bayesian network is a **directed acyclic graph** consisting of nodes which represent random variables and arrows which correspond to probabilistic dependencies between them.

There has been a great deal of interest in recent years on the NP-hard problem of learning the structure (placement of directed edges) of Bayesian networks from data ([10],[16],[23],[24],[29],[31],[33],[34],[45]). Much of this has been driven by the study of genetic regulatory networks in molecular biology due to advances in technology and, specifically, microarray techniques that allow scientists to rapidly measure expression levels of genes in cells. As an integral part of machine learning, Bayesian networks have also been used for pattern recognition, language processing including speech recognition, and credit risk analysis.

Structure learning typically involves defining a network score function and is then, in theory, a straightforward optimization problem. In practice, however, it is quite a different story as the number of possible networks to be scored and compared grows super-exponentially with the number of nodes. Simple greedy hill climbing with random restarts is understandably inefficient yet surprisingly hard to beat. There have been many deterministic and stochastic alternatives proposed in recent years such as dynamic programming ([36],[47],[46]), genetic ([15],[39],[40]), tempering ([5]), and Metropolis-Hastings ([20],[23],[41]) algorithms. There have been different approaches to the task, including order space sampling and the scoring of graph "features" rather than graphs themselves. Several of these methods have offered some improvement over greedy hill climbing but can be difficult to implement. Deterministic methods tend to get stuck in local maxima and probabilistic methods tend to suffer from slow mixing.

In this chapter we consider a new stochastic algorithm in which the appearance and disappearance of edges are modeled as a birth and death process. We compare our algorithm with the popular Metropolis-Hastings algorithm and give empirical evidence that ours has better mixing properties.

5.1 Bayesian Networks

Bayesian networks are graphical representations of the relationships between random variables from high dimensional probability distributions. A Bayesian Network on N nodes is a *directed acyclic graph* (DAG) where the nodes (vertices), labeled $1, 2, \ldots, N$, correspond to random variables $X_1, X_2, \ldots, X_N$ and directed edges correspond to probabilistic dependencies between them. We say that node i is a **parent** of node j and node j is a **child** of node i if there exists a directed edge from i to j, (in which case we write $i \to j$), and use the notation pa_j to denote the collection of all parents of node j. We will refer to nodes and their associated random variables interchangeably. Thus, pa_j may also represent the collection of parent random variables for X_j. Rigorously, a Bayesian network consists of a DAG and a set of conditional densities $\{P(X_i|pa_i)\}_{i=1}^N$ along with the assumption that the joint density for the N random variables can be written as the product

$$P(X_1, X_2, \ldots, X_N) = \prod_{i=1}^{N} P(X_i|pa_i).$$

In other words, all nodes are conditionally independent given their parents.

In the problem of structure inference, the DAG is not explicitly known. A set of observations $D = (D_1, D_2, \ldots, D_M)$ is given, where each D_i is an N-tuple realization of $(X_1, X_2, \ldots, X_N)$. The goal is then to recover the "best" edge structure of the underlying Bayesian Network, which may be measured in many ways. For example, one may consider the best DAG as the one that maximizes the posterior probability

$$P(G|D) \propto P(D|G)P(G) \tag{5.1}$$

over G. Indeed, (5.1) is important for many measures of "best DAGs" and it is the goal of this chapter to simulate DAGs efficiently from this distribution.

Throughout this chapter, we will use the common assumption that the data D come from a multinomial distribution, allowing us to analytically integrate out parameters to obtain a score which is proportional to $P(G|D)$.

5.2 Sampling Methods

In this section, we discuss two sampling methods for $P(G|D)$. First, in Section 5.2.1, we review a Metropolis-Hastings based algorithm for structure learning. Then, in Section 5.2.3, we introduce a new algorithm for sampling from $P(G|D)$ based on a continuous time Markov process. The goal is to reduce the number of wasted computational steps that can come from many rejections in Metropolis-Hastings. Instead of stepping through a discrete time chain with the same state repeated many times before a move to a new state, our goal is to find when the next transition will occur, and jump to that transition. This idea is illustrated in Figure 5.1. The circles represent a realization of a discrete time Markov chain with transitions rarely occuring. The red line depicts a continuous time process with the same transitions. The length of the line between transitions represents the waiting time between jumps. By knowing the waiting times and the resulting transitions, we can potentially remove wasted computational time of frequent rejections.

5.2.1 Structure Metropolis-Hastings

In [41], Madigan et al. construct a Metropolis-Hastings Algorithm for walking through the space of BN structures, which was later improved upon by Grzegorczyk and Husmeier in [26]. The basis of the algorithm is the construction of a proposal density based on three possible types of moves: adding a single edge, removing a single edge, or reversing the direction of a single edge. Of course, only moves which leave the graph acyclic are allowed in the proposal. Thankfully, the

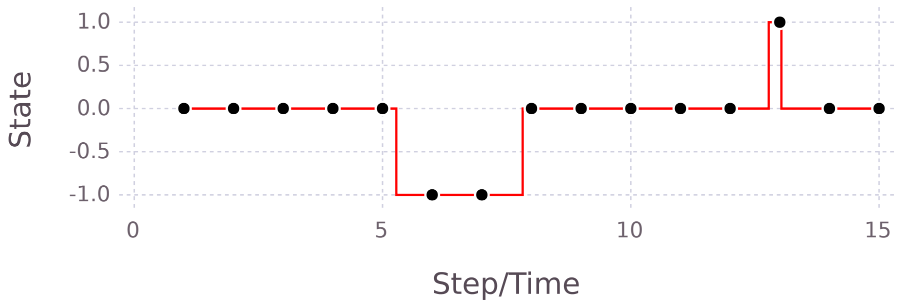


Figure 5.1: An example realization of a Markov chain with many rejections, with the state at each step given by circles, and a continuous time process with the same transitions given as a red line.

possible moves are quite easy to calculate ([22]). The possible edge additions are given by the zeros of the matrix

$$\left[(I-A)^{-1}\right]^T + A$$

where A is the adjacency matrix of the current graph, and I is the identity matrix. The possible edge reversals are given by the ones of the matrix

$$(I-A)^{-1} \bullet A$$

where $A \bullet B$ is component-wise multiplication between A and B. The Structure Metropolis-Hastings Algorithm then goes as follows.

(1) Start with an arbitrary initial DAG, G_0, and set $n = 0$.

(2) Choose a proposal graph G' uniformly from all graphs obtainable by one of the following

- Adding a single non-cycle causing edge to G_n.
- Removing a single edge from G_n.
- Performing a non-cycle causing reversal of a single edge in G_n.

(3) Draw $U \sim \text{unif}(0,1)$

(4) If

$$U \leq \frac{P(G'|D)(\#\text{ of moves from } G')}{P(G_n|D)(\#\text{ of moves from } G_n)},$$

set $G_{n+1} = G'$. Otherwise, set $G_{n+1} = G_n$.

(5) Set $n = n+1$ and return to step (2).

5.2.2 Jump Processes

Let $(\Omega, \mathcal{F})$ be a state space consisting of a non-empty set Ω and a sigma algebra $\mathcal{F}$ on Ω. A *jump process* on $(\Omega, \mathcal{F})$ is a continuous time stochastic process that is characterized as follows. Assume the process is in some some state $x \in \Omega$.

- The waiting time until the next jump follows an exponential distribution with rate (or intensity) $\lambda(x)$ and is independent of the past history.

- The probability that the jump lands the process in $F \in \mathcal{F}$ is given by a transition kernel $K(x, F)$.

It is known (e.g. [21], [49]) that there exists a $Q_t : \Omega \times \mathcal{F} \to \mathbb{R}^+$ so that $Q_t(x, F)$ is the probability that at time t the process is in F given that the process was in state x at time 0. Such Q_t are defined as the solution to Kolmogorov's backward equation

$$\frac{\partial}{\partial t} Q_t(x, F) = -\lambda(x) Q_t(x, F) + \lambda(x) \int_\Omega Q_t(y, F) K(x, dy).$$

Furthermore, let $Q_t^{(n)}(x, F)$ be the probability of a transition from x to F with at most n jumps. If $\lambda(x)$ is bounded then

$$Q_t^{(\infty)}(x, F) := \lim_{n \to \infty} Q_t^{(n)}(x, F)$$

is the unique minimal solution to Kolmogorov's forward equation

$$\frac{\partial}{\partial t} Q_t(x, F) = -\int_F \lambda(z) Q_t(x, dz) + \int_\Omega \lambda(z) K(z, F) Q_t(x, dz).$$

(It is "minimal" in the sense that if $R_t(x, F)$ is any other nonnegative solution, then $R_t(x, F) \geq Q_t^{(\infty)}(x, F)$ for all $t \geq 0$, $x \in \Omega$, and $F \in \mathcal{F}$.)

For a distribution, π, to be invariant under such a process, π must satisfy the detailed balance conditions

$$\pi(x)\lambda(x)K(x, dy)d\mu(x) = \pi(y)\lambda(y)K(y, dx)d\mu(y)$$

with respect to some density μ.

Preston [49] extended this jump process to the trans-dimensional case where jumps from states in Ω_n can move the process to a one dimension higher state, living in Ω_{n+1}, with (birth) rate $\lambda_b(x)$ or to a one dimension lower state, living in Ω_{n-1}, with (death) rate $\lambda_d(x)$. Associated with these birth and death rates are birth and death kernels, K_b and K_d. The total jump rate and the transition kernel are then given by

$$\lambda(x) = \lambda_b(x) + \lambda_d(x)$$

$$K(x, F) = \frac{\lambda_b(x)}{\lambda(x)} K_b(x, F) + \frac{\lambda_d(x)}{\lambda(x)} K_d(x, F).$$

For a configuration x with n points to move to a configuration x' with $n+1$ points (or vice versa), the detailed balance conditions simplify ([49], [50]) to requiring balance, with respect to π, of the birth and death rate of a single point in the configuration. That is, we require that

$$\pi(x) b(x, x' \setminus x) = \pi(x') d(x', x' \setminus x)$$

where $b(x, x' \setminus x)$ is the birth rate of the single point $x' \setminus x$ given that the current configuration of points is x, and $d(x', x' \setminus x)$ is the death rate of the single point $x' \setminus x$ given that the current configuration of points is x'. These relate to the total birth and death rates in that

$$\lambda_b(x) = \sum b(x, x' \setminus x)$$

$$\lambda_d(x') = \sum d(x', x' \setminus x)$$

where the birth sum is taken over all states x' that consist of configuration x with the addition of a single point and the death sum is taken over all states x that consist of configuration x' with a single point deleted.

5.2.3 A Birth and Death Process on Edges of a BN

To construct a jump process for BN structure inference, our goal is to construct a birth and death process acting on edges of a BN which has invariant distribution $P(G|D)$.

The relevant state space is $(\mathcal{G}, 2^{\mathcal{G}})$, where $\mathcal{G}$ is the set of all DAGs with N nodes and $2^{\mathcal{G}}$ is the power set of $\mathcal{G}$. We define the disjoint sets $\mathcal{G}_k$, $k = 0, \ldots, \frac{N(N-1)}{2}$, to be the set of DAGs with exactly k edges. Our jump process will then jump between the $\mathcal{G}_k$ for adjacent values of k.

For $G \in \mathcal{G}_k$, denote the graph with the addition of the edge from node i to node j by $(G \cup \{i \to j\}) \in \mathcal{G}_{k+1}$, and the graph with the removal of the edge from i to j by $(G \setminus \{i \to j\}) \in \mathcal{G}_{k-1}$.

Detailed balance then requires that, for every edge $i \to j$ that is a valid, i.e. non-cycle causing, addition,

$$P(G|D)\, b(G, \{i \to j\}) = P(G \cup \{i \to j\}|D)\, d(G \cup \{i \to j\}, \{i \to j\}).$$

It is convenient to let

$$d(G \cup \{i \to j\}, \{i \to j\}) = 1$$

so that

$$\begin{aligned} b(G, \{i \to j\}) &= \tfrac{P(G \cup \{i \to j\}|D)}{P(G|D)} \\ &= \tfrac{P(D|G \cup \{i \to j\}) P(G \cup \{i \to j\})}{P(D|G)P(G)} \end{aligned}$$

If we let Δ_j denote the M-dimensional vector of observations of X_j in the data set D, this birth rate may be rewritten as

$$b(G, \{i \to j\}) = \frac{P(\Delta_j|\Delta_{pa'_j}, G \cup \{i \to j\}) P(G \cup \{i \to j\})}{P(\Delta_j|\Delta_{pa_j}, G)P(G)}.$$

Here, Δ_{pa_j} is the $M \times k$ matrix of data points for the k parents of node j in G ($\Delta_{pa_j} = \emptyset$ if $k = 0$) and $\Delta_{pa'_j}$ is the $M \times (k+1)$ matrix of data points for the k parents of node j in $G \cup \{i \to j\}$.

The transition rates are then given by

$$\lambda_b(G) = \sum_{\text{valid } i \to j} b(G, \{i \to j\}),$$

$$\lambda_d(G) = \sum_{\{i \to j\} \in G} 1.$$

With this, we can easily construct a way to simulate from this process in the following way.

(1) Start with an arbitrary initial DAG, G

(2) Compute the birth rates $b(G, \{i \to j\})$ for all possible valid $i \to j$ edge additions to G. Compute

$$\lambda_b(G) = \sum_{\text{valid } i \to j} b(G, \{i \to j\})$$

and

$$\lambda_d(G) = \sum_{\{i \to j\} \in G} 1.$$

(3) With probability $\lambda_d(G)/(\lambda_b(G)+\lambda_d(G))$, remove a randomly selected existing edge.

Otherwise, add valid edge $i \to j$ with probability $b(G, \{i \to j\})/\lambda_b(G)$.

(4) Return to step 2.

At first glance, it may seem like such an algorithm would be computationally expensive, as the required birth rates depend on computing the score for two different graphs. However, if we assume a modular score, the computation at each step is manageable. A modular score means we have

$$P(D|G) = \prod_{i=1}^{N} P(\Delta_i|\Delta_{pa_i}, G)$$

which leads to a birth rate of

$$\begin{aligned} b(g, \{i \to j\}) &= \frac{P(G \cup \{i \to j\}) \prod_{i=1}^{N} P(\Delta_i|\Delta_{pa'_i}, G \cup \{i \to j\})}{P(G) \prod_{i=1}^{N} P(\Delta_i|\Delta_{pa'_i}, G)} \\ &= \frac{P(G \cup \{i \to j\}) P(\Delta_j|\Delta_{pa'_j}, G \cup \{i \to j\})}{P(G) P(\Delta_j|\Delta_{pa_j}, G)} \end{aligned}$$

So, for each birth rate, we only need the ratio of the altered score to current score of a single node corresponding to the child end of the proposed edge.

Further computational relief comes from the fact that most of the birth rates can be stored from previous steps. After adding or removing an edge, the only birth rates that need to be recalculated are for edges pointing to nodes whose parent set has been altered, edges which were not previously valid, or edges which are no longer valid.

These realizations allow for a more complete and efficient algorithm

(1) Start with an initial DAG, G

(2) For all possible edge additions, $i \to j$, calculate the birth rate $b(G, \{i \to j\})$

(3) With probability $\lambda_d(G)/(\lambda_b(G)+\lambda_d(G))$, remove a randomly selected existing edge.

Otherwise, add a valid edge $k \to \ell$ with probability $b(G, \{k \to \ell\})$.

(4) If any of the following are true, update the birth rate $b(G, \{i \to j\})$.

- $j = \ell$
- Edge $i \to j$ was not a valid addition before the addition or removal of edge $k \to \ell$ but now is valid
- Edge $i \to j$ was a valid addition before the addition or removal of edge $k \to \ell$ but now is no longer valid

(5) Return to step 3.

5.3 Experimental Results

While the theory guarantees that our edge birth and death process will have the correct stationary distribution, in this Section we investigate the mixing time and whether or not Monte Carlo simulation of graphs from $P(G|D)$ using our method is feasible in practice. In order to attempt to avoid overfitting, i.e. introducing too many edges to fit the data, we will select our prior distribution $P(G)$ to give a log score proportional to the Akaine Information Criteria (AIC).

$$\log P(G|D) \propto \text{AIC}(G|D) = -2 \log P(D|G) + 2\,(\#\text{ parameters in G})$$

In the simulation study, we first consider a simple 4 node graph so that we may compare our results with exact computations. We generated 50 observations of X_1, X_2, X_3, X_4 as related by the DAG in Figure 5.2. Each node was allowed to take on values in $\{1, 2, 3, 4\}$, with equal probability.

We then ran both the standard Structure Metropolis-Hastings algorithm and our Edge Birth and Death Algorithm. Figure 5.3 shows two independent runs of each algorithm. While both algorithms tend to find high probability regions of the state space, our edge birth and death algorithm explores much more of the state space with far fewer number of steps.

One benefit of running Monte Carlo methods is they allow us to easily calculate the probabilities of specific graph features, e.g. edge probabilities. With the Edge birth and Death Algorithm, the edge probabilities correspond to the proportion of time that the graphs contain the given edge. With the same 4 node graph as before, we calculated the edge probabilities and compare them to

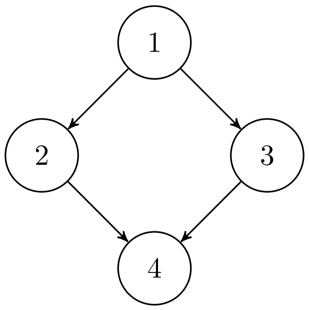


Figure 5.2: 4 Node Graph Used for Testing

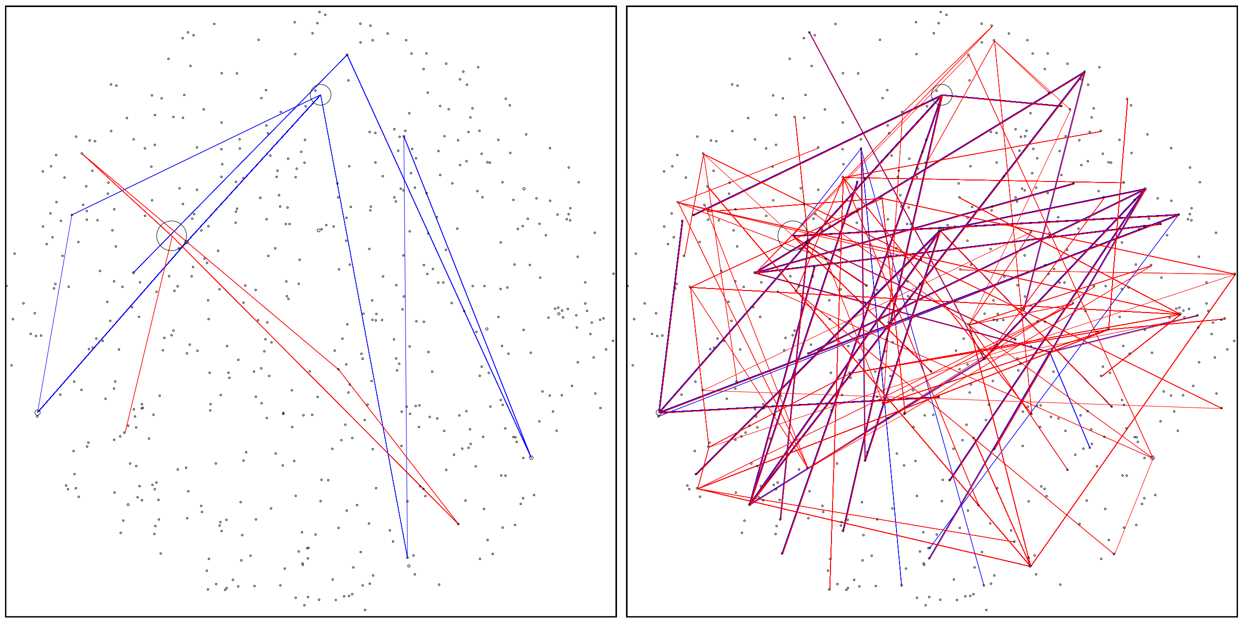


Figure 5.3: (Left) Paths of two (red and blue) independent Metropolis-Hastings chains of $20,000$ steps. (Right) Paths of two independent Edge Birth and Death Algorithm runs of $1,500$ jumps. The circles represent each of the 543 possible DAGs, with size proportional to the posterior probability of the graph.

the true edge probabilities both for $m = 100$ observations and $m = 500$ observations. The results are shown in Table 5.1. As the number of observations increases, the error in estimating edge probabilities decreases, as expected. Even with edges that are not being predicted entirely accurately, the error is balanced out between the probability of an edge and the reversed version of the edge. For example, with 50 observations, the probability of edge $1 \to 2$ existing was underestimated by 0.06, but the probability of edge $2 \to 1$ existing was overestimated by 0.06. This keeps the correct dependencies between random variables, but the causal directions may be less accurate with fewer observations.

Node	1	2	3	4
1	–	-.06	+.02	-.05
2	+.06	–	.00	-.01
3	-.02	+.04	–	-.01
4	+.04	-.04	.00	–

Node	1	2	3	4
1	–	+.03	-.02	.00
2	-.03	–	.00	.00
3	+.02	.00	–	.00
4	.00	.00	.00	–

Table 5.1: Errors from estimated to exact probabilities of edges in a 4 node Bayesian network from one run of the Edge Birth and Death Algorithm for 100 observations (left) and 500 observations (right).

Next, we tested our edge birth and death algorithm on the "Alarm data set" compiled by Herskovits ([30]). This data set, often used as a benchmark for structure learning algorithms, consists of 1000 observations of 37 random variables, each taking on 4 possible values. We ran our edge birth and death algorithm for 10^5 time steps. For comparison, we ran the standard Metropolis-Hastings scheme for 2×10^6 steps which took approximately the same amount of CPU time. The AIC was recorded at each step and is plotted for one of these runs in Figure 5.4. As is well known, the MH scheme is prone to getting trapped in local minima, and takes many steps to escape these points whereas our edge birth and death algorithm appears to mix more easily.

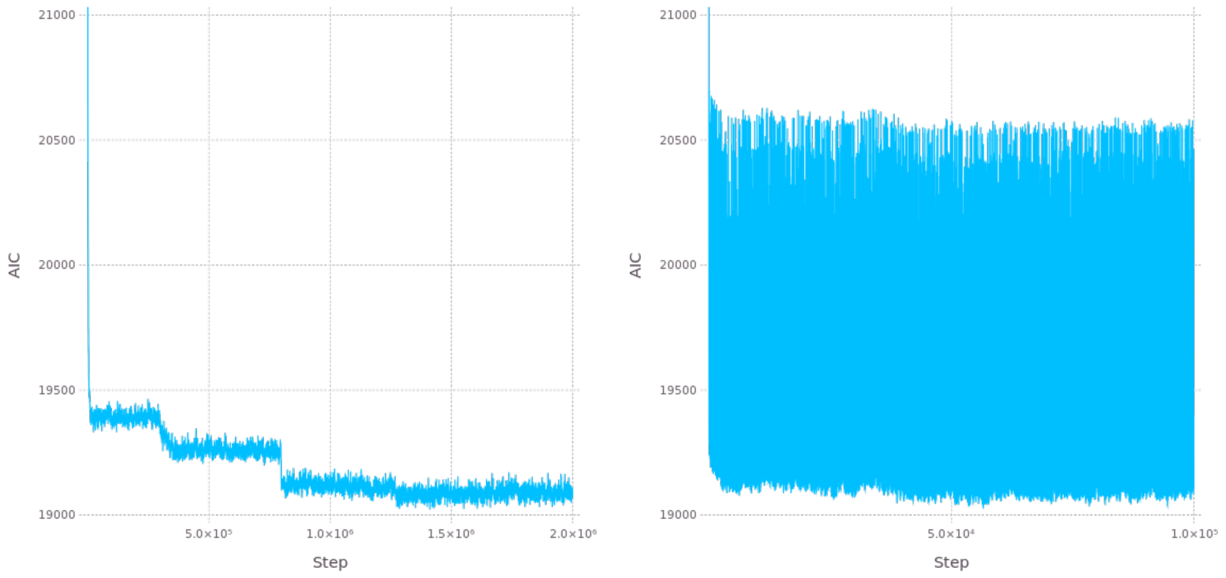


Figure 5.4: (Left) AIC score of a Metropolis-Hastings run of 2×10^6 steps. (Right) AIC score of an edge birth and death process run of 10^5 steps.

Chapter 6

Conclusion

In this thesis, we studied three separate applications for which Markov chain Monte Carlo methods have commonly been implemented. In each of these applications, we developed MCMC algorithms that exhibit enhanced efficiency and/or accuracy.

In Chapter 2, we developed a "windowed" rejection sampling algorithm to target the marginal distributions of the posterior of a hidden Markov model. The decaying influence of past samples was exploited in order to have small enough windows for the algorithm to be efficient while still retaining enough samples to have improved accuracy over the commonly used sequential importance sampling algorithms.

In Chapter 3, a derivation one-dimensional Kac model was reexamined, with emphasis placed on the underlying Poisson process. We then leveraged this Poisson process to create a more accurate sampling algorithm for particle collisions. This Poisson process was used further to construct a proof of concept ε-perfect algorithm for sampling the limiting distribution of particle velocities as time goes to infinity. This algorithm returns samples with a controlled level of sampling error, ε.

Chapters 4 and 5 were focused on directed acyclic graphs and bayesian networks. Chapter 4 explored the possibility of creating a perfect sampler for the uniform distribution over the set of DAGs on a given number of nodes. A theoretical Regenerative DAG Coupler was discussed with a known regeneration time. We then paired Coupling from the Past concepts with a modified version of a known Markov chain with uniform stationary distribution to create our DAG CFTP Algorithm, which functions as a perfect uniform sampling algorithm. The backward coupling time of the DAG CFTP Algorithm was explored empirically, and while it scales better than the size of the state space, and better than the Regenerative DAG Coupler, it is still prohibitively expensive to use. In Chapter 5, we presented a new algorithm for sampling from the posterior distribution of Bayesian Networks given data based on a birth and death process. This new Edge Birth and Death Algorithm allows for probabilistic inferences to be made about Bayesian Network structures while avoiding some of the downfalls of existing methods. In particular, our Edge Birth and Death Algorithm does not get trapped in local extrema as easily as structure MCMC and allows for less restrictive choices of priors over graphs compared to order MCMC. An open question related to this Edge Birth and Death Algorithm is whether it can be made perfect by applying similar constructions to perfect algorithms for spatial point processes.

Bibliography

[1] Christophe Andrieu and Arnaud Doucet. Particle filtering for partially observed gaussian state space models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 64(4):827–836, 2002.

[2] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72(3):269–342, 2010.

[3] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on signal processing, 50(2):174–188, 2002.

[4] Hans Babovsky and Reinhard Illner. A convergence proof for nanbu's simulation method for the full boltzmann equation. SIAM journal on numerical analysis, 26(1):45–65, 1989.

[5] Daniel James Barker, Steven M Hill, and Sach Mukherjee. Mc4: a tempering algorithm for large-sample network inference. In IAPR International Conference on Pattern Recognition in Bioinformatics, pages 431–442. Springer, 2010.

[6] GA Bird. Molecular gas dynamics and the direct simulation monte carlo of gas flows. Clarendon, Oxford, 508, 1994.

[7] EA Carlen, Maria C Carvalho, and Michael Loss. Kinetic theory and the kac master equation. Contemporary mathematics, 552:1–20, 2011.

[8] Eric A Carlen, Maria C Carvalho, J Le Roux, Michael Loss, and Cédric Villani. Entropy and chaos in the kac model. arXiv preprint arXiv:0808.3192, 2008.

[9] George Casella, Michael Lavine, and Christian P Robert. Explaining the perfect sampler. The American Statistician, 55(4):299–305, 2001.

[10] Ting Chen, Hongyu L He, George M Church, et al. Modeling gene expression with differential equations. In Pacific symposium on biocomputing, volume 4, page 4, 1999.

[11] Wen-shiang Chen, Bhavik R Bakshi, Prem K Goel, and Sridhar Ungarala. Bayesian estimation via sequential monte carlo sampling: unconstrained nonlinear dynamic systems. Industrial & engineering chemistry research, 43(14):4012–4025, 2004.

[12] Jem N Corcoran and Richard L Tweedie. Perfect sampling of ergodic harris chains. Annals of Applied Probability, pages 438–451, 2001.

[13] JN Corcoran and RL Tweedie. Perfect sampling from independent metropolis-hastings chains. Journal of statistical planning and inference, 104(2):297–314, 2002.

[14] Mary Kathryn Cowles and Bradley P Carlin. Markov chain monte carlo convergence diagnostics: a comparative review. Journal of the American Statistical Association, 91(434):883–904, 1996.

[15] Luis M de Campos and Juan F Huete. Approximating causal orderings for bayesian networks using genetic algorithms and simulated annealing. In Proceedings of the Eighth IPMU Conference, volume 1, pages 333–340, 2000.

[16] Adrian Dobra, Chris Hans, Beatrix Jones, Joseph R Nevins, Guang Yao, and Mike West. Sparse graphical models for exploring gene expression data. Journal of Multivariate Analysis, 90(1):196–212, 2004.

[17] Arnaud Doucet, Nando De Freitas, and NJ Gordon. Sequential monte carlo methods in practice. series statistics for engineering and information science, 2001.

[18] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. Statistics and computing, 10(3):197–208, 2000.

[19] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. Handbook of nonlinear filtering, 12(656-704):3, 2009.

[20] Byron Ellis and Wing Hung Wong. Learning causal bayesian network structures from experimental data. Journal of the American Statistical Association, 103(482):778–789, 2008.

[21] William Feller. An introduction to probability theory and its applications: volume I, volume 3. John Wiley & Sons London-New York-Sydney-Toronto, 1968.

[22] Nicandro Flores. Counting directed acyclic graphs and its application to Monte Carlo learning of Bayesian networks. ProQuest, 2007.

[23] Nir Friedman and Daphne Koller. Being bayesian about network structure. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence, pages 201–210. Morgan Kaufmann Publishers Inc., 2000.

[24] Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using bayesian networks to analyze expression data. Journal of computational biology, 7(3-4):601–620, 2000.

[25] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In IEE Proceedings F-Radar and Signal Processing, volume 140, pages 107–113. IET, 1993.

[26] Marco Grzegorczyk and Dirk Husmeier. Improving the structure mcmc sampler for bayesian networks by introducing a new edge reversal move. Machine Learning, 71(2-3):265–305, 2008.

[27] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. Biometrika, 57(1):97–109, 1970.

[28] Anton J Haug. Bayesian estimation and tracking: a practical guide. John Wiley & Sons, 2012.

[29] David Heckerman. A tutorial on learning with bayesian networks. In Learning in graphical models, pages 301–354. Springer, 1998.

[30] Edward Herskovits. Computer-based probabilistic-network construction. PhD thesis, Stanford University USA, 1991.

[31] Dirk Husmeier. Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic bayesian networks. Bioinformatics, 19(17):2271–2282, 2003.

[32] Jaime S Ide and Fabio G Cozman. Random generation of bayesian networks. In Brazilian symposium on artificial intelligence, pages 366–376. Springer, 2002.

[33] Seiya Imoto, Sunyong Kim, Takao Goto, Sachiyo Aburatani, Kousuke Tashiro, Satoru Kuhara, and Satoru Miyano. Bayesian network and nonparametric heteroscedastic regression for nonlinear modeling of genetic network. Journal of bioinformatics and computational biology, 1(02):231–252, 2003.

[34] E Jarvis, V Smith, K Wada, M Rivas, M McElroy, T Smulders, P Carninci, Y Hayashizaki, F Dietrich, X Wu, et al. A framework for integrating the songbird brain. Journal of Comparative Physiology A, 188(11-12):961–980, 2002.

[35] Mark Kac. Foundations of kinetic theory. In Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, volume 1955, pages 171–197, 1954.

[36] Mikko Koivisto and Kismat Sood. Exact bayesian structure discovery in bayesian networks. Journal of Machine Learning Research, 5(May):549–573, 2004.

[37] Max Krook and Tai Tsun Wu. Formation of maxwellian tails. Physical Review Letters, 36(19):1107, 1976.

[38] Jack Kuipers and Giusi Moffa. Uniform random generation of large acyclic digraphs. Statistics and Computing, 25(2):227–242, 2015.

[39] Pedro Larranaga, Cindy MH Kuijpers, Roberto H Murga, and Yosu Yurramendi. Learning bayesian network structures by searching for the best ordering with genetic algorithms. IEEE transactions on systems, man, and cybernetics-part A: systems and humans, 26(4):487–493, 1996.

[40] Pedro Larrañaga, Mikel Poza, Yosu Yurramendi, Roberto H. Murga, and Cindy M. H. Kuijpers. Structure learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. IEEE transactions on pattern analysis and machine intelligence, 18(9):912–926, 1996.

[41] David Madigan, Jeremy York, and Denis Allard. Bayesian graphical models for discrete data. International Statistical Review/Revue Internationale de Statistique, pages 215–232, 1995.

[42] Guy Melancon, Isabelle Dutour, and Mirelle Bousquet-Melou. Random generation of dags for graph drawing. 2000.

[43] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092, 1953.

[44] Kenichi Nanbu. Direct simulation scheme derived from the boltzmann equation. i. monocomponent gases. Journal of the Physical Society of Japan, 49(5):2042–2049, 1980.

[45] Irene M Ong, Jeremy D Glasner, and David Page. Modelling regulatory pathways in e. coli from time series expression profiles. Bioinformatics, 18(suppl 1):S241–S248, 2002.

[46] Sascha Ott, Seiya Imoto, and Satoru Miyano. Finding optimal models for small gene networks. In Pacific symposium on biocomputing, volume 9, pages 557–567. Citeseer, 2004.

[47] Sascha Ott and Satoru Miyano. Finding optimal gene networks using biological constraints. Genome Informatics, 14:124–133, 2003.

[48] Judea Pearl. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, 1988.

[49] Chris Preston. Spatial birth and death processes. Advances in applied probability, 7(3):465–466, 1975.

[50] Brian D Ripley. Modelling spatial patterns. Journal of the Royal Statistical Society. Series B (Methodological), pages 172–212, 1977.

[51] Wolfgang Wagner. A convergence proof for bird's direct simulation monte carlo method for the boltzmann equation. Journal of Statistical Physics, 66(3-4):1011–1044, 1992.