INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

The Normal Kernel Coupler: An adaptive Markov Chain Monte Carlo method for efficiently sampling from multi-modal distributions

Gregory R. Warnes

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Washington

2000

Program Authorized to Offer Degree: Department of Biostatistics

UMI Number: 9995450

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature Cara R Wang

Date October 23, 2000

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

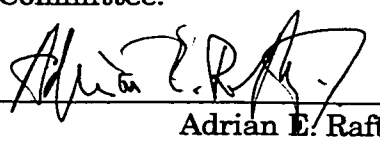
Gregory R. Warnes

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

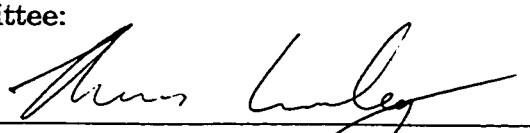
examining committee have been made.

Chair of Supervisory Committee:

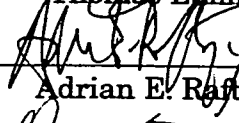


Adrian E. Raftery

Reading Committee:



Thomas Lumley



Adrian E. Raftery

Anthony J. Rossini

Date: October 23, 2000

University of Washington

Abstract

The Normal Kernel Coupler: An adaptive Markov Chain Monte Carlo method for efficiently sampling from multi-modal distributions

by Gregory R. Warnes

Chair of Supervisory Committee

Professor Adrian E. Raftery
Statistics

The Normal Kernel Coupler (NKC) is an adaptive Markov Chain Monte Carlo (MCMC) method which maintains a set of current state vectors. At each iteration one state vector is updated using a density estimate formed by applying a normal kernel to the full set of states.

We give proofs showing that this sampler is ergodic (irreducible, Harris recurrent and aperiodic) for any continuous distribution on d-dimensional Euclidean space. We also show that the NKC outperforms standard MCMC methods on a variety of uni-modal and bimodal problems in low to moderate dimensions.

Further, we address practical issues in using the NKC by giving direction for the selection of various parameters and by providing a run-length diagnostic. Using these we give a systematic method for initializing the NKC, selecting the kernel variance,

and determining the number of MCMC iterations.

We demonstrate the utility of the NKC on a problem of current interest in cancer genetics which has two distinct and dissimilar modes and show that the results are consistent with current scientific understanding.

Finally, we introduce Hydra, a software library for MCMC. We show how to use Hydra to implement both a variable-at-a-time Metropolis sampler and the NKC for our example problem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Except the Lord build the house,

they labour in vain that build it

—Psalm 127

I wish to thank my parents, Roger and LenNae Warnes. From my father I inherited an interest many fields. This enabled my course of study at the intersection of Computer Science, Mathematical Statistics, and Biology. My mother blessed me in many ways, including endowing me with the gift of self assurance and a conviction that I could accomplish anything I set my mind to.

I thank my wife Becky for her consideration and support, for her willingness to share in the sacrifices required to complete my studies, and for the inspiration and example she provides. I also thank Becky's parents, Larry and Janet Pedersen, for constant friendship and for help in a time of need. I appreciate my cat, Little Man, for the comfort provided by his fondness for a warm lap and for the reminder to recreate provided by his not-so-subtle requests to play.

Special thanks to Adrian Raftery for serving as my dissertation advisor and committee chair. I appreciate the many hours he invested working with me over the last

three years. His advice, both technical and strategic, contributed greatly to the success of my research. I have learned a great deal and am honored to have worked with him.

I thank my other reading committee members, Thomas Lumley and Anthony Rossini. Each has contributed friendship as well as useful advice over a period that began long before the reading committee was formed. I appreciate the time and expertise contributed by the other members of my dissertation committee, Elizabeth Thompson, Larry Snyder, and Scott Davis.

I'm indebted to Mary Emond for mentoring me during the first years of the PhD program and for introducing me to the Seattle Barretts Esophagus research group. I thank the members of the Barretts group, in particular, Brian Reid, Carissa Sanchez, Patricia Galipeau, Michael Barrett, David Wong, Peter Rabinovich, and David Cowan. Each of them provided invaluable help as well as friendship. My involvement with the Barretts group stoked my interest in cellular and molecular biology and provided understanding and background that enhanced the quality of my education.

I also thank Mary Emond and Manisha Desai for sharing their work on using hierarchical Bayesian models to study the rate of LOH in Barretts Esophagus. They provided both the data and model used as the applied example in this text.

I thank John Chambers and Duncan Temple-Lang for assistance in developing the HYDRA software libray. An initial version was developed as part of the the `Omegahat` project during an internship at Bell Labs during the summer of 1999.

This research was supported in part by NIH/NIAID Grant no. 5 T32 AI07450-09 NIH Grant no. 1 PO1 CA76466, NIH Grant no. 1 PO1 CA76466, and ONR Grant no. N00014-96-1-0192.

Chapter 1

INTRODUCTION

In this dissertation we introduce the Normal Kernel Coupler (NKC), an adaptive Markov Chain Monte Carlo (MCMC) technique intended to efficiently sample from multi-modal distributions. In contrast to standard MCMC methods, which maintain a single current state vector, the NKC maintains a *set* of current state vectors. At each iteration one state vector is updated via a Metropolis-Hastings update scheme using a proposal distribution formed by applying a normal kernel to the full set of state vectors. The flexibility and robustness of the kernel density estimate allow the NKC to efficiently sample from unimodal and multi-modal distributions defined on a moderate number of dimensions.

1.1 Markov Chain Monte Carlo

Markov Chain Monte Carlo, introduced by Metropolis et al. (1953), is a computational method for generating samples from arbitrarily complicated distributions (Geman & Geman, 1984; Hastings, 1970). Its particular strength is the ability to generate samples for distributions for which the density is only known up to a constant factor.

MCMC has found recent popularity as a tool for fitting Bayesian statistical models

(see, for example, Gilks et al., 1996). Fitting these models requires estimating the distribution of the parameters (considered random) conditional on the observed data. The density of this "posterior" distribution is obtained by applying Bayes' rule to the likelihood of the data given the parameters, $\mathcal{L}(\mathbf{D}|\theta)$, and a prior distribution $\pi(\theta)$ on the parameters:

$$\pi(\theta|\mathbf{D}) = \frac{\mathcal{L}(\mathbf{D}|\theta)\pi(\theta)}{\int \mathcal{L}(\mathbf{D}|\theta)\pi(\theta)d\theta} = \frac{p(\theta)}{\int p(\theta)d\theta}$$

The formulation of the Metropolis-Hastings algorithm allows the creation of MCMC samplers for the full posterior distribution $\pi(\theta|\mathbf{D})$ using only the numerator $p(\theta) = \mathcal{L}(\mathbf{D}|\theta)\pi(\theta)$. This removes the need to perform the often high-dimensional and analytically intractable integration in the denominator.

A properly constructed MCMC sampler will generate a sequence of points, $X_1, X_2, ...X_t$, which converges in distribution to the posterior. Once a sufficient sample has been generated the expected value of any function f can then be estimated by computing the corresponding sample path average, $\widehat{\mathbf{E}[f]} = \frac{1}{N}\sum_{t=1}^{N} f(X_t)$.

The most general MCMC technique is the Metropolis-Hasting algorithm (Hastings, 1970), which includes all of the common MCMC techniques as special cases. The basic formulation is simple. Given a target distribution Π and an initial starting location X_0, each iteration of the sampler consists of four steps:

1. **Propose** a new state Y using a proposal distribution $Q(X_t)$, which may depend on the current state X_t:

$$Y \leftarrow Q(X_t)$$

2. **Compute** the Metropolis-Hastings acceptance probability

$$\begin{aligned}\alpha(X_t, Y) &= \min\left\{1, \frac{\pi(Y)\, q(X_t|Y)}{\pi(X)\, q(Y|X_t)}\right\} \\ &= \min\left\{1, \frac{p(Y)\, q(X_t|Y)}{p(X)\, q(Y|X_t)}\right\}\end{aligned}$$

where π is a density corresponding to the target distribution Π, $p(x) \propto \pi(x)$ is an unnormalized density, and $q(Y|X_t)$ is the conditional density for Y under $\mathcal{Q}(X_t)$.

3. **Accept** the proposed point Y with probability $\alpha(X_t, Y)$ and set

$$X_{t+1} \leftarrow Y,$$

otherwise reject the proposed point and set

$$X_{t+1} \leftarrow X_t.$$

4. **Increment** time: $t \leftarrow (t+1)$.

There is considerable latitude in the selection of the proposal distribution, $\mathcal{Q}$. The sampler's convergence rate depends on the exact relationship between $\mathcal{Q}$ and Π, as well as on the choice of initial state X_0. Tierney (1996) provides an overview of the properties of Π and $\mathcal{Q}$ that guarantee eventual convergence to the target distribution, govern the speed of this convergence, and determine the efficiency with which the target distribution is explored once convergence has been attained.

Two essential properties for an MCMC sampler are irreducibility and recurrence. Loosely speaking, irreducibility means that the sampler can reach any region in the target distribution with positive probability from any other such region in finite time, while recurrence means that starting from any such region, the sampler will visit every other such region infinitely often. Presence of both properties is sufficient to ensure that the MCMC sampler converges to the target distribution and that sample path averages converge to their expected values.

1.1.1 Standard MCMC Samplers

A variety of Metropolis-Hastings samplers have been suggested in the literature. While the simplest Metropolis-Hastings sampler is the independence sampler (Tierney, 1994), the most commonly applied samplers are the Gibbs sampler (Geman & Geman, 1984), the random-walk Metropolis sampler (Metropolis et al., 1953), and the componentwise Metropolis sampler (Metropolis et al., 1953).

1.1.2 The Independence Sampler

An independence sampler is formed using a proposal distribution that does not depend on the current state of the MCMC chain. Instead, a fixed proposal is selected which is similar to the unknown target distribution. The performance of the independence sampler depends on how well the proposal distribution mimics the target distribution. To guarantee convergence, this proposal must have positive density wherever the target density is positive. In practice, it can be difficult to construct a proposal that ensures efficient sampling.

1.1.3 Gibbs Sampling

At each iteration, the Gibbs sampler generates values for a subset of the parameters using thier posterior distribution given all of the others. With this proposal distribution, the Metropolis-Hastings acceptance probability reduces to the constant one, ie $\alpha(X_t, Y) = 1$ for all X_t, Y. Consequently, all proposed values are accepted.

The Gibbs sampler is popular in domains, such as spatial statistics, where the conditional distributions have simple forms and permit easy sampling. In other domains, the required conditional distributions can be tedious to derive and may not allow easy sample generation.

The Gibbs sampler is often constructed so that it only proposes changes to a single parameter at each step. This structure can make the Gibbs sampler inefficient when the distribution has high correlation between parameters. In this case a large number of "stair-step" moves is required to traverse even a short distance through the distribution. In addition, single-component moves make it difficult for the sampler to transition between modes separated by regions of low probability. In the extreme case where regions are separated by areas of zero probability, the Gibbs sampler can be completely unable to move between the regions, making it reducible and preventing convergence.

1.1.4 Random-walk Metropolis

The random-walk Metropolis sampler is very simple. In its standard formulation, it generates candidate points using a normal distribution centered at the current state,

$$Y \leftarrow \mathrm{N}_d(X_t,\ \mathbf{V}).$$

Since this proposal is symmetric in X and Y, the Metropolis-Hastings acceptance function reduces to $\alpha(X,Y) = \min(1, p(Y)/p(X))$. When used with an appropriate covariance matrix, $\mathbf{V}$, the random-walk Metropolis sampler can be quite efficient for low and moderate dimensional problems.

Gelman et al. (1995) showed that for multivariate normal target distributions the optimal covariance matrix is approximately $(2.38^2/d)\mathrm{Var}(\Pi)$, where $\mathrm{Var}(\Pi)$ is the true variance of the target distribution. Unfortunately, $\mathrm{Var}(\Pi)$ is usually unknown and difficult to estimate. As a consequence, the random-walk Metropolis sampler is often constructed by running several initial "tuning" simulations to determine an effective value for $\mathbf{V}$. This hand tuning can be time consuming and fault-prone, especially when the initial variance guess is poor so that the sampler converges slowly to the target distribution.

1.1.5 The componentwise Metropolis sampler

The componentwise Metropolis sampler is a random-walk Metropolis sampler that updates a single parameter at each iteration using a univariate normal proposal centered at the current value. Like the componentwise Gibbs sampler, the componen-

twise Metropolis sampler can be very inefficient for distributions with highly correlated parameters. It is however, easier to apply to high-dimensional problems than the random-walk Metropolis sampler. As with the random-walk Metropolis sampler, the efficiency of the componentwise sampler depends on the proposal variances, with the proper values usually unknown.

1.2 The Problem

While the standard MCMC techniques are effective for posterior distributions which have a single dominant mode, they are unsuitable for distributions with modes that are separated by low probability areas. Such distributions commonly arise in the context of mixture models and variance component models.

To see why standard MCMC techniques have problems with multi-modal posteriors, consider sampling from a bimodal posterior distribution formed from a mixture of two multivariate normal distributions, one centered at (0,0, .., 0) and the other at (9, 9, .., 9), each with unit covariance. Two-dimensional likelihood contours for this distribution are shown in Figure 1.1.

Both the Gibbs sampler and the variable-at-a-time Metropolis sampler only generate moves parallel to a subset of the coordinate axes. Since the two modes in this example are not aligned along any of the coordinate axes, neither sampler can generate a proposal that moves between the modes in a single step (see figures 1.2(a) and 1.2(b)). Since the probability density is non-zero over the region between the modes, these methods do allow transition to the other mode via a sequence of moves. However,

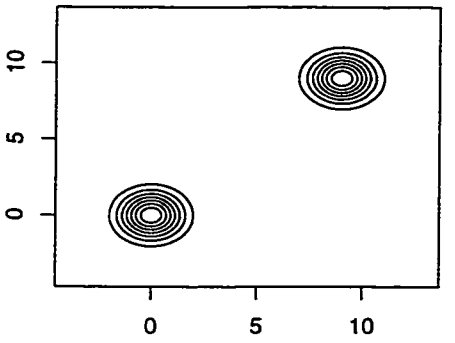
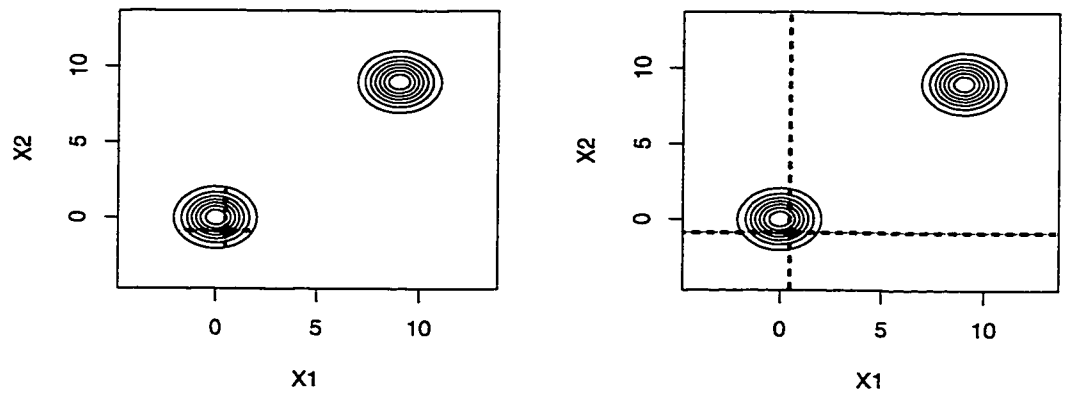


Figure 1.1: Density contours for a mixture of two unit-variance bivariate normals

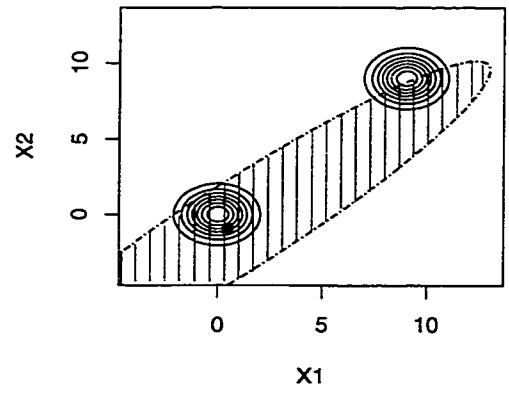
the probability of such a sequence is so small that such transitions are extremely rare in practice. As a consequence, these samplers will remain in whichever mode is located first for a very long period before the second mode is visited. While it is possible, in this case, to transform the parameters so that the modes lie along the axes, finding a suitable transformation can be difficult or impossible in practice.

In contrast, the standard random-walk Metropolis sampler, which proposes updates for all parameters at each step, can generate moves between the modes. Unfortunately, when this sampler is constructed using a proposal variance which is sufficiently large to permit single-step moves between the modes, it will generate candidate points spread over a large region most of which does not overlap with the target density (see Figure 1.2(c)). This will result in a large fraction of candidate points being rejected, reducing the acceptance rate and the efficiency of the sampler. Conversely, a proposal variance which allows efficient estimation within the current mode will convey a high acceptance rate but will also render moves between the modes extremely rare. The conflict between the need to reduce the proposal variance to allow a reasonable accep-



(a) Gibbs Sampler

(b) Variable-at-a-time Metropolis



(c) Random-walk Metropolis

Figure 1.2: Possible moves for the Gibbs, variable-at-a-time Metropolis, and (multivariate) random-walk Metropolis samplers for the point represented by the large dot.

tance probability and the need to maintain a large variance to enable moves between the modes becomes more pronounced as the modes become more separated and as the number of dimensions increases.

As illustrated by this simple example, the Gibbs sampler, componentwise Metropolis sampler, and random-walk Metropolis samplers are not suitable for sampling from posterior distributions with non-overlapping modes. This has lead to the search for MCMC techniques which are appropriate for such distributions.

1.3 Previous Work

Three research directions relate to our work. The first is the development of MCMC techniques designed to improve simulation from multi-modal distributions. The second is the development of adaptive MCMC methods. The third is the development of adaptive importance sampling methods based on kernel density estimation.

1.3.1 Samplers for Multi-Modal Distributions

Gelman & Rubin (1992) recommended creating a custom independence proposal from a mixture of multivariate normal distributions. They suggest locating the peaks of posterior modes by using numerical maximization starting from many different starting locations. The proposal distribution is then constructed as a mixture of normal distributions, with one normal centered at each peak with variance set to the inverse information matrix calculated at the peak.

Geyer (1991) introduced Metropolis-Coupled MCMC, which uses a set of concurrent

MCMC samplers operating on a set of successively smoother distributions. Coupling these samplers by occasionally swapping current states allows the sampler operating on the density of interest to "inherit" the increased mobility possible in the smoother distributions. Mariani & Parisi (1992) and later Geyer & Thompson (1994) introduced Simulated Tempering, a related method which allows a single sampler to follow a random walk jointly on the parameter space and the set of smoothed distributions rather than using a set of concurrent samplers. Neal (1996) extended this work by replacing the random walk over the smoothed distributions with a systematic scan. This ensures that the distribution of interest is sampled more frequently.

Another approach was recently introduced by Tjielmeland & Hegstad (2000). They provide a method for integrating numerical maximization into a random-walk Metropolis sampler to permit sampling from multiple posterior modes. When combined with a standard MCMC technique this method promises to enable better mobility between the modes.

1.3.2 Adaptive MCMC Samplers

Adaptive MCMC samplers use information generated by the current MCMC run to improve the efficiency of the MCMC simulation. While adapting the proposal distribution based on previous states can destroy the Markov property of the sampler and prevent convergence to the desired target distribution, three adaptive methods which converge properly have been proposed: finite stage adaptation, adaptation at regeneration times, and Adaptive Metropolis Sampling.

Gelfand & Sahu (1994) introduced finite stage adaptation. In this approach, adaptation occurs only during an initial period of fixed length. After the adaptation stage is complete, an estimation stage is run. During the estimation stage, the MCMC sampler does no adaptation, so the standard MCMC convergence proofs apply. This method is a formalization of the "pre-simulation tuning" that has been practiced for some time.

Gilks et al. (1998) describe adaptation at regeneration times of the MCMC sampler. A regeneration time occurs when the MCMC sampler enters a region A, called an "atom", with the property that the future history of the chain is conditionally independent of the past. IE, given $X_t \in A$, $X_0, X_1, \ldots, X_{t-1}$ is conditionally independent of $X_{t+1}, X_{t+2}, \ldots$ At such regeneration times, the sampler can be modified using the past history without disturbing convergence.

Gilks et al. (1994a) introduced Adaptive Direction Sampling (ADS). Instead of a single current state, ADS maintains a set of state vectors. At each iteration one component is updated by generating a new point from the conditional distribution along a line passing through the current value. The orientation of the line is determined by sampling two of the remaining state vectors and computing the orientation of the line connecting them. Adaptive Metropolis Sampling (Gilks & Roberts, 1996) generalizes ADS to allow more general proposal distributions.

1.3.3 Adaptive importance samplers

Adaptive methods have also been proposed for importance sampling. Notably, West (1993) introduced an iterative method where samples from a previous iteration are

used to construct an importance weighted kernel density estimate. This density estimate is then used as the proposal distribution for the next set of samples. Givens & Raftery (1996), Oehlert (1998), and Zhang (1996) provide extensions.

1.4 Our Approach

We are interested in obtaining a MCMC method which efficiently samples from both unimodal and multi-modal posterior distributions that are continuous on d-dimensional Euclidean space and that requires minimal tuning and pre-simulation effort. To allow the widest applicability, we desire a method that does not require the posterior density to be available in an analytical form. One consequence of this requirement is that the Hessian is not used.

We approach this problem by combining the AMS framework described by Gilks & Roberts (1996) with kernel density estimation to form the Normal Kernel Coupler (NKC). The NKC maintains a current set where each component has the same target distribution. At each iteration, one component of the current set is updated by generating a candidate state from the density formed by applying a normal kernel to the elements of the current set. In Chapter 2 we show that the flexibility of the kernel density estimate allows the NKC to efficiently sample from both unimodal and multi-modal distributions with a moderate number of parameters.

Our work is a fusion of the three research areas mentioned previously. First, we have the goal of efficient sampling from multi-modal distributions. Second, we use the AMS framework to form an adaptive sampler. Third, we borrow West's idea of using

kernel density estimate to provide the proposal distribution for our sampler.

1.5 Organization of the Text

In this introductory chapter, we have provided an overview of Markov Chain Monte Carlo, described the most commonly applied MCMC techniques, and have shown why these methods are not suitable multi-modal posterior distributions. Next, we reviewed related work on MCMC samplers for multi-modal distributions, adaptive MCMC samplers, and adaptive importance samplers. Finally, we described our approach to developing a MCMC sampler suitable for multi-modal distributions.

In Chapter 2, we describe the NKC and prove that it is ergodic (irreducible, Harris recurrent and aperiodic) for any continuous distribution on d-dimeionsal Euclidean space. We then use a simulation study to demonstrate the superiority of the NKC over standard methods for a variety of examples in 4 and 20 dimensions. Next, we demonstrate the utility of the NKC by applying it to a problem in cancer genetics which has two distinct and very different modes. We finish the chapter with a discussion of the strengths and weaknesses of the NKC and suggest avenues for future research.

In Chapter 3 we focus on issues in implementing and using NKC. We give directions for selecting various algorithm parameters and develop a run-length diagnostic. We then provide an iterative method for initializing the NKC, selecting the kernel variance, and determining the number of MCMC iterations. Next we give a detailed example by using our iterative method to apply the NKC to the problem introduced in Chapter 2.

Chapter 4 describes the HYDRA MCMC library which allows custom MCMC samplers, including NKC, to be created with a minimum of programming. We first review the design philosophy and tools used to create HYDRA. We then demonstrate HYDRA by creating first a random-walk Metropolis sampler, and then a NKC, for the model introduced in Chapter 2.

In Chapter 5, we summarize our results, review the strengths and weaknesses of the NKC, and provide directions for future research.

Chapter 2

THE NORMAL KERNEL COUPLER: AN ADAPTIVE MCMC METHOD FOR EFFICIENTLY SAMPLING FROM MULTI-MODAL DISTRIBUTIONS

In this chapter, we present the Normal Kernel Coupler (NKC). We show that NKC is efficient for low and moderate dimensional problems with a small number of modes, give an applied example from cancer genetics, and suggest approaches for extending NKC to higher dimensional problems.

2.1 Multi-State Samplers

Multi-state samplers differ from standard MCMC methods by maintaining a set of current states, $X_t^{(1)}, X_t^{(2)}, \ldots, X_t^{(c)}$, called the current set. Each element $X_t^{(i)}$ of the current set is a vector defined on the original parameter space. The elements of the current set may share a common target distribution or may each have a different target distribution. Multi-state samplers update one or more elements of the current set at each iteration by applying the Metropolis-Hastings algorithm with a proposal distribution that (potentially) depends on the entire current set. This dependence allows the proposal to adapt to the target distribution using information stored in the current set.

2.1.1 Previous Work

A variety of sequential and parallel MCMC methods can be formulated using the multi-state framework. Published examples include Metropolis-Coupled MCMC (Geyer, 1991) and Adaptive Direction Sampling (ADS) (Gilks et al., 1994a). Other variants of multi-state sampling include Adaptive Metropolis Sampling (Gilks & Roberts, 1996) and "parallel chains" (Tierney & Mira, 1999).

Metropolis-Coupled MCMC is a multi-state sampler where each component of the current set has a different target distribution. The target distributions for the components are successively smoother versions of the distribution of interest. Most of the time Metropolis-Coupled MCMC updates each element of the current set independently using a Gibbs sampler. On the smoothest distributions, the Gibbs sampler can quickly move between high-probability regions even when they are separated by areas of low (but non-zero) posterior probability. With some predetermined frequently, a move "swapping" the values of adjacent elements is performed. This swapping allows the component operating on the rougher distributions (including the distribution of interest) to make large jumps that would otherwise be extremely improbable.

Adaptive Direction Sampling (ADS) uses a current-set where each element of the current set has the same target distribution. At each iteration one component is updated by generating a new point from the conditional distribution along a line passing through the current value. The orientation of the line is determined by sampling two of the remaining state vectors and computing the orientation of the line connecting them. This allows ADS to adapt to the orientation of the target distribution as it is

explored.

Despite their apparent complexity, standard MCMC theory can be applied to multi-state samplers by considering the current-set as a single large parameter vector and determining the corresponding "overall" proposal and target distributions. For Adaptive Direction Sampling, the target distribution for the "concatenated" state vector is the product of C copies of the original target distribution, where C is the number of components of the current set. For Metropolis-Coupled MCMC, the target distribution is a product of the individual "smoothed" distributions. With corresponding definitions for the proposal distributions, it becomes clear that both multi-state samplers preserve the Markov property. Further, standard convergence theorems become easy to apply.

2.1.2 Related Work

Recently, Haario et al. (1999) introduced the Adaptive Proposal (AP) method. AP adapts the variance of a random-walk Metropolis sampler using a fixed number of sequentially recorded recent states. When generating a proposal for state X_{t+1} the variance is computed using states $X_t, X_{t-1}, \ldots, X_{t-k}$. By considering the set of states as a vector, it becomes clear that AP is not a proper Metropolis-Hastings sampler because the update step replaces the value of X_{t-k} with X_{t+1}, but computes the acceptance probability as if X_t was being replaced. Consequently, standard Metropolis-Hastings convergence theorems do not apply.

Instead, Haario, Saksman, and Tamminen show that AP converges to a distribution that is related to, but different from, the specified target distribution. They note that

this has the potential to introduce considerable bias in estimated quantities. Thus, practical use of this sampler requires not only diagnosing convergence—already a difficult task—but also determination of whether the stationary distribution is sufficiently similar to the desired target for it to be useful for the problem at hand.

2.2 The Normal Kernel Coupler (NKC)

The Normal Kernel Coupler (NKC) is a multi-state sampler where each component has the same target distribution. At each iteration, a new value is proposed for one component state using a kernel density estimate constructed from the entire current set. Since the kernel density estimate makes very few assumptions about the form of the target distribution, the Normal Kernel Coupler's efficiency is largely independent of the number and location of modes. When properly constructed, the NKC efficiently samples from both unimodal and multi-modal target distributions, even when the distribution is oddly shaped or the modes are well separated.

2.2.1 The Algorithm

Let $\pi(X)$, defined on $X \in \Re^d$ (d-dimensional Euclidean space), be the density of the distribution of interest. Let $p(X)$ be a function which is proportional to $\pi(X)$. Let $X_t^{(\cdot)} = (X_t^{(1)}, \ldots, X_t^{(C)})$, be a vector of component states where each $X_t^{(i)} \in \Re^d$. Note that subscripts index time, while parenthesized superscripts index component states.

We will use $\mathrm{N}_d(\mu, \Xi)$ to represent a d-variate normal distribution with mean vector μ and covariance matrix Ξ. In a slight abuse of notation we will use $\mathrm{N}_d(v \mid \mu, \Xi)$

to represent the *density* at v of a d-variate normal with mean vector μ and covariance matrix Ξ.

The Normal Kernel Coupler iterates through six step a variant of the Metropolis-Hastings update cycle:

1. **Select** a component state, $X_t^{(i)}$, to update.

2. **Propose** a new state $Y^{(i)}$ for component i using a normal kernel density estimate by randomly selecting a source component, $X_t^{(u)}$, where

$$u \sim \text{Discrete Unifiorm}(1, .., C)$$

and then generating a value from a normal centered at $X_t^{(u)}$:

$$Y^{(i)}|u \sim \mathrm{N}_d\left(X_t^{(u)},\ h^2\mathbf{V}\right),$$

so that the density of $Y^{(i)}|X_t^{(\cdot)}$ is

$$\begin{aligned} q_k(Y^{(i)}|X_t^{(\cdot)}) &= q_k(Y^{(i)}|X^{(i)}, X^{(-i)}) \\ &= \frac{1}{C}\sum_{j\neq i}^{C}\mathrm{N}_d\left(Y^{(i)}|X_t^{(j)},\ h^2\mathbf{V}\right) + \frac{1}{C}\mathrm{N}_d\left(Y^{(i)}|X_t^{(i)},\ h^2\mathbf{V}\right) \end{aligned}$$

where h^2 is a bandwidth tuning constant and $\mathbf{V}$ determines the shape and scale of the normal kernel.

3. **Compute** the Metropolis-Hastings acceptance probability

$$\begin{aligned}\alpha(X_t^{(i)}, Y^{(i)}|X_t^{(-i)}) &= \min\left\{1, \frac{\pi(Y^{(i)})\, q_k(X_t^{(i)}|Y^{(i)}, X_t^{(-i)})}{\pi(X^{(i)})\, q_k(Y^{(i)}|X_t^{(i)}, X_t^{(-i)})}\right\} \\ &= \min\left\{1, \frac{p(Y^{(i)})\, q_k(X_t^{(i)}|Y^{(i)}, X_t^{(-i)})}{p(X^{(i)})\, q_k(Y^{(i)}|X_t^{(i)}, X_t^{(-i)})}\right\}.\end{aligned}$$

4. **Accept** the proposed point $Y^{(i)}$ and set

$$X_{t+1}^{(i)} \leftarrow Y^{(i)}$$

with probability $\alpha(X_t^{(i)}, Y^{(i)}|X_t^{(-i)})$, otherwise,

Reject the proposed point and set

$$X_{t+1}^{(i)} \leftarrow X_t^{(i)}.$$

5. **Copy** the remaining states

$$X_{t+1}^{(-i)} \leftarrow X_t^{(-i)}$$

6. **Increment** time: $t \leftarrow (t+1)$.

2.3 Convergence

A concern with adaptive MCMC methods is the possibility that such methods may fail to converge to the desired stationary distribution. Fortunately, the NKC lends itself to

a straightforward proof that it is ergodic (irreducible, Harris recurrent and aperiodic), which is sufficient to ensure convergence.

Define the joint state as the vector formed by concatenating each of the component states, $X^{(\cdot)} = (X^{(1)}, .., X^{(C)})$, the joint target distribution by

$$\pi_*(X^{(\cdot)}) = \prod_{i=1}^{C} \pi(X^{(i)})$$

and the joint proposal density by

$$q_*(Y^{(\cdot)}|X^{(\cdot)}) = q_k(Y^{(c_t)}|X^{(\cdot)}) \prod_{i=1}^{C} \delta(Y^{(i)} = X^{(i)})$$

where $\delta(\cdot)$ is the indicator function that takes value 1 when its argument is true and 0 otherwise, and c_t cycles through a permutation of the integers $1, \ldots, C$. With these definitions, the NKC is seen to be a variable-at-a-time Metropolis-Hastings sampler for the joint target where each component state, $X^{(i)}$, is considered a parameter.

Theorem 1 *If π is a continuous distribution on $\Re^d$, $h^2 > 0$, and $\mathbf{V}$ is positive definite, then the NKC constructed for π using h^2 and $\mathbf{V}$ is ergodic (π_*-irreducible, Harris-recurrent, and aperiodic) with unique invariant distribution π_*.*

Proof 1

1. $\pi_*^{(\cdot)}$-irreducible

   The transition kernel P for the NKC from a point x to a set A is

$$P(x, A) = \int_A q_*(x, y)\alpha(x, y)\mu(dy),$$

and the n-step transition kernel is defined recursively by

$$P^n(x, A) = \int P(x, dy) P^{(n-1)}(y, A)$$

for $n \geq 2$ where $P(x, dy)$ is the probability of moving to a small measurable subset $dy \subset S$ given that the move starts at x.

We need to show that for all $x \in \Re^{d \times C}$ and $A \subset \Re^{d \times C}$ there is a value of n for which

$$P^n(x, A) > 0 \text{ for whenever } \pi_*(A) > 0. \tag{2.1}$$

First, note that the conditions on $h^2\mathbf{V}$ guarantee that

$$q_k(Y^{(i)}|X^{(\cdot)}) = \sum_{i=1}^{C} \mathrm{N}_d\big(Y^{(i)}|X^{(j)},\ h^2\mathbf{V}\big) > 0 \qquad \text{for all } Y^{(i)} \in \Re^d \text{ and } X^{(\cdot)} \in \Re^{d \times C}. \tag{2.2}$$

Consequently,

$$\alpha(X^{(i)}, Y^{(i)}|X^{(-i)}) = \min\left\{1, \frac{p(Y^{(i)})\ q_k(Y^{(i)} \to X_t^{(i)}|X_t^{(-i)})}{p(X^{(i)})\ q_k(X_t^{(i)} \to Y^{(i)}|X_t^{(-i)})}\right\} > 0 \tag{2.3}$$

whenever $p(Y^{(i)}) \propto \pi(Y^{(i)}) > 0$.

Let $P(X^{(i)}, Y^{(i)}|X^{(\cdot)})$ be the transition kernel for a single step of the NKC which updates component i conditional the set of current states. Without loss of generality, assume that the permutation defining the order of component updates is $1, 2, .., C$. Now, the transition probability for one complete scan of the C compo-

nent states is

$$
\begin{aligned}
P^C(X^{(\cdot)}, Y^{(\cdot)}) &= \prod_{i=1}^{C} P(X^{(i)}, Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) && (2.4)\\
&= \prod_{i=1}^{C} \{ q_k(Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) \times && (2.5)\\
&\qquad \alpha(X^{(i)}, Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) \} && (2.6)\\
&> 0 \quad \text{for all } X^{(\cdot)}, Y^{(\cdot)} \in \Re^{d \times C} \text{ whenever } \pi_*(X^{(\cdot)}) > 0 && (2.7)
\end{aligned}
$$

by (2.2) and (2.3). Since this holds for any $Y^{(\cdot)} \in A \subset \Re^{d \times C}$ the NKC is irreducible for π_*.

2. Harris recurrent

Chan & Geyer (1994) give sufficient conditions for an irreducible variable-at-a-time Metropolis-Hastings sampler to be Harris recurrent:

Theorem 2 *A variable-at-a-time Metropolis-Hastings algorithm on R^d with proposal distributions that are absolutely continuous with respect to Lebesgue measure is Harris recurrent if all of the conditional samplers (including the unconditional sampler which conditions on the empty set of variables $I = \emptyset$) are irreducible for any values of the fixed variables.*

The essence of the Chan-Geyer condition is that the sampler is Harris recurrent if updates on any subset of the components $I = \{i_1, .., i_k\}$, conditional on the remaining components $I^- = \{i_{k+1}, .., i_C\}$, are irreducible for any fixed values of the components I^-. See Chan & Geyer (1994) for the proof.

For the NKC, the Chain-Geyer condition is easily verified. Consider the k-step transition kernel for the conditional sampler with $C-k$ components held fixed:

$$\begin{aligned} P^k(X^{(I)}, Y^{(I)} | X^{(I^-)}) &= \prod_{i=1}^{k} P(X^{(i)}, Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,k)}), X^{(-I)}) \\ &= \prod_{i=1}^{k} P(X^{(i)}, Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) \\ &= \prod_{i=1}^{k} \{ q_k(Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) \times \\ &\qquad \alpha(X^{(i)}, Y^{(i)} | (Y^{(1,..i-1)}, X^{(i,..,C)})) \} \\ &> 0 \quad \text{for all } X^{(I)}, Y^{(I)} \in \Re^{d \times k} \text{ and } X^{(-I)} \in \Re^{d \times C-k} \\ &\qquad \text{whenever } \pi_*(X^{(\cdot)}) > 0. \end{aligned}$$

Consequently, each of the conditional samplers is irreducible, the requirements of theorem 2 are met, and the NKC is Harris recurrent. Since the chain has invariant distribution π_* by construction, the conditions to Tierney (1996) Theorem 4.2 and 4.3 are met. Consequently, the NKC has π_* as its unique invariant distribution, it converges to this distribution, and sample path averages computed using NKC converge to the true expectation under π_*.

3. Aperiodic

Recall that an m-cycle for an irreducible chain with transition kernel P is a collection $\{E_0, E_1, \ldots, E_{m-1}\}$ of disjoint sets such that

$$P(x, E_j) = 1 \text{ for } j = i+1 \mod m \text{ and for all } x \in E_i.$$

Define the support S of π_* as $S = \{x \in \Re^{d \times C} : \pi_*(x) > 0\}$. Without loss of generality, we can assume that $E_i \subset S$ for $i = 0, \ldots, m-1$. Now, if an m-cycle is present, then

$$P^t(x, E_k) = 1 \text{ for } k = i + t \mod m \text{ and for all } x \in E_i.$$

Consider $t = C$. We have previously shown that $P^C(x, A) > 0$ for all $x \in \Re^{d \times C}$ and $A \in S$. Consequently $P^C(x, E_k) = 1$ for all $X \in E_i$ can only hold if $m = 1$ so that $E_0 = E_k = S$. Since the largest cycle length is 1, the NKC is aperiodic.

Corollary 3 *The distribution of the sequence of values taken by each component state* $\mathbf{X^{(i)}} = \{X_t^{(i)}, t = 0, 1, ..\}$ *converges to* π.

Proof 3

Since the vector of component states $X^{(\cdot)}$ is converging to π_*, which factorizes to π for each component $X^{(i)}$, the distributions of the individual components converge to π.

2.4 Simulations

We performed a simulation study to compare the efficiency of the NKC to the efficiency of three standard MCMC methods; a custom independence proposal, a variable-at-a-time Metropolis sampler, and a random-walk Metropolis sampler using a multivariate normal proposal.

Target	Distribution
OneMode	$\mathrm{N}_d\big((0,0,..0),\ \mathbf{I}\big)$
Narrow	$\mathrm{N}_d\big((0,0,..0),\ \mathrm{AR}_1(0.95)\big)$
TwoMode	$\frac{1}{2}\mathrm{N}_d\big((0,0,..0),\ \mathbf{I}\big) + \frac{1}{2}\mathrm{N}_d\big((9,9,..,9),\ \mathbf{I}\big)$
BigAndSmall	$\frac{1}{2}\mathrm{N}_d\big((0,0,..0),\ \mathbf{I}\big) + \frac{1}{2}\mathrm{N}_d\big((9,9,..,9,\ \frac{1}{16}\mathbf{I}\big)$
HeavyAndLight	$\frac{1}{8}\mathrm{N}_d\big((0,0,..0),\ \mathbf{I}\big) + \frac{7}{8}\mathrm{N}_d\big((9,9,..,9),\ \frac{1}{16}\mathbf{I}\big)$
Banana	$\frac{1}{2}\mathrm{N}_d\big((-1.5,1.5,..1.5),\ \mathrm{AR}_1(-0.95)\big) + \frac{1}{2}\mathrm{N}_d\big((1.5,1.5,..,1.5),\ \mathrm{AR}_1(0.95)\big)$
TwoNarrow	$\frac{1}{2}\mathrm{N}_d\big((0,0,..,0),\ \mathrm{AR}_1(-0.95)\big) + \frac{1}{2}\mathrm{N}_d\big((9,9,..,9),\ \mathrm{AR}_1(0.95)\big)$

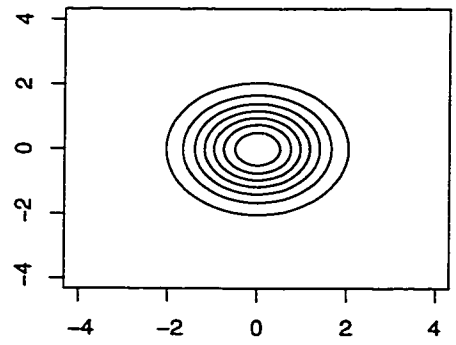
Figure 2.1: Densities for the target distributions used in the simulation study. I is the Identity Matrix. $\mathrm{AR}_1(\rho)$ is an AR-1 covariance matrix with correlation parameter ρ (see table 2.1).

2.4.1 Target Distributions

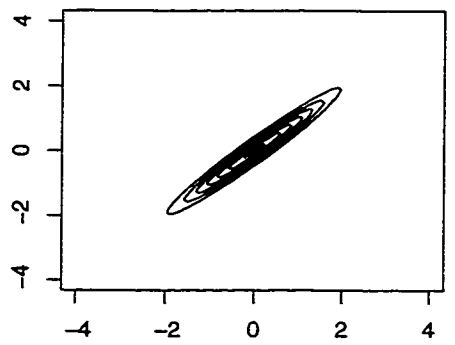
For the simulations, we constructed a set of seven test distributions which abstract different characteristics of posterior densities encountered in practice. Each distribution was given a descriptive title. These are "OneMode", "Narrow", "TwoMode", "BigAndSmall", "HeavyAndLight", "Banana", and "TwoNarrow". Formulae and contour plots for these distributions are given in tables 2.1 and 2.2 respectively.

OneMode This target distribution, a spherical d-variate normal with identity covariance matrix, represents an ideal target distribution for which all MCMC methods should perform well.

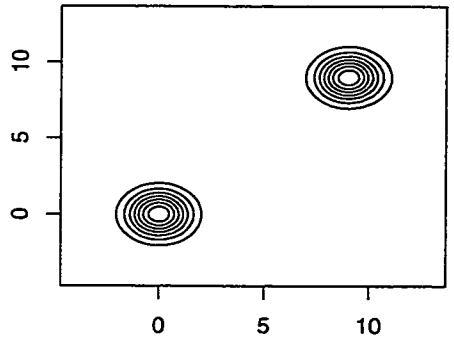
Narrow The second sampler is a d-variate normal with an AR-1 style covariance matrix (see table 2.1) with with correlation coefficient $\rho = 0.95$. It represents a more



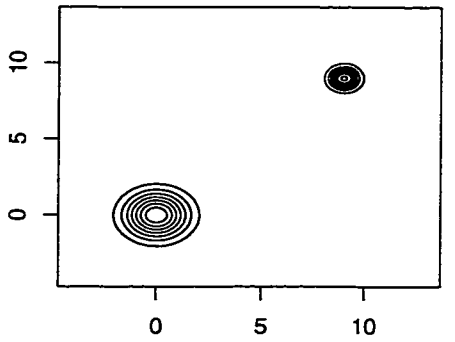
(a) OneMode



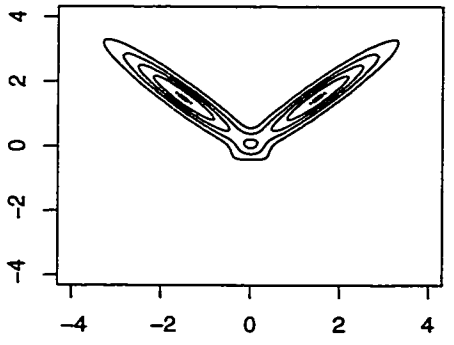
(b) Narrow



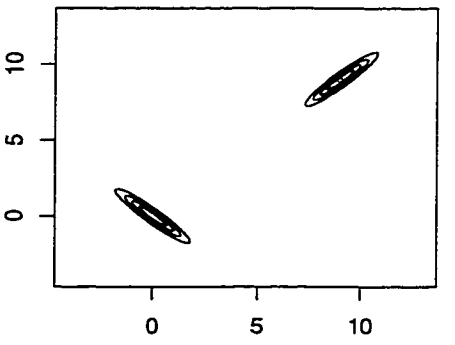
(c) TwoMode



(d) BigAndSmall (equal weight) and HeavyAndLight (weight=$\frac{1}{8}$, $\frac{7}{8}$)



(e) Banana



(f) TwoNarrow

Figure 2.2: Contour plots of the target distributions used in the simulation study

realistic target distribution which is approximately normal but which has highly correlated parameters.

Table 2.1: AR-1 style covariance matrix with correlation parameter ρ

$$\mathrm{AR}_1(\rho) = \begin{pmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{C-1} \\ \rho & 1 & \rho & \cdots & \rho^{C-2} \\ \rho^2 & \rho & 1 & \cdots & \rho^{C-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{C-1} & \rho^{C-2} & \rho^{C-3} & \cdots & 1 \end{pmatrix}$$

TwoMode The third distribution is composed of two equally weighted d-variate normals, each with an identity covariance matrix. The first is centered at $(0, 0, .., 0)$, while the second is centered at $(9, 9, 9, ..., 9)$. This target mimics the behavior of distributions with highly separated modes, where each mode is essentially identical and is well modeled by a multivariate normal.

BigAndSmall The fourth sampler is also formed using two equally weighted d-variate normals, one at $(0, 0, .., 0)$ and the other at $(9, 9, .., 9)$. However, the second mode now has a identity covariance matrix scaled down by a factor of $\frac{1}{16}$. This target exhibits different scales and different sized basins of attraction. These features can cause samplers to incorrectly assign extra mass to the larger mode.

HeavyAndLight The fifth sampler is constructed using the same normals as the previous sampler, "BigAndSmall", but the larger mode is now assigned only $\frac{1}{8}$th of

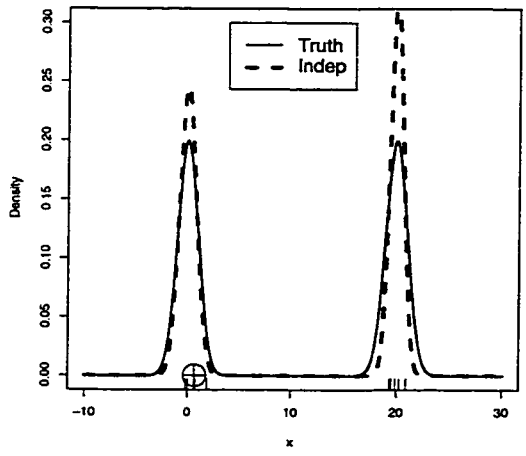
the mass. This makes it very likely that samplers will incorrectly assign too much mass to the larger mode.

Banana The sixth target distribution is also formed from two d-variate normals. These two normals are centered at $(1.5, 1.5, .., 1.5)$ and $(-1.5, 1.5, .., 1.5)$. The first normal has an AR-1 style covariance matrix with correlation coefficient $\rho = 0.95$. The second mode has an AR-1 structure with correlation coefficient $\rho = -0.95$. This creates two narrow normals aligned perpendicularly, with some overlap at $(0, 0, .., 0)$. This target mimics the complex topologies which can can seriously reduce the efficiency of many samplers.

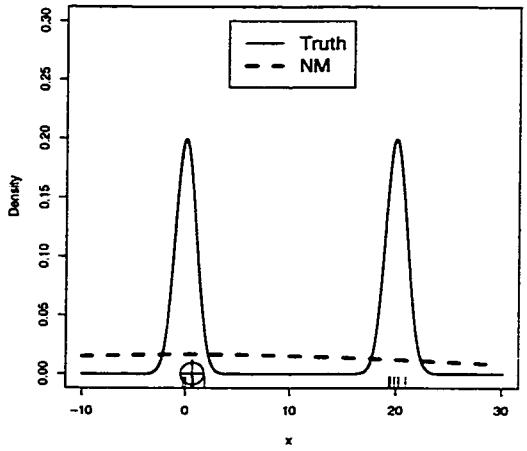
TwoNarrow The seventh target also has two perpendicular modes with AR-1 structure with $\rho = 0.95$ and $\rho = -0.95$ respectively. This time, the modes are quite separated, with one at $(0, 0, .., 0)$ and the other at $(9, 9, .., 9)$. This combines the problems of different covariance structure within modes with those of separated modes.

2.4.2 Samplers

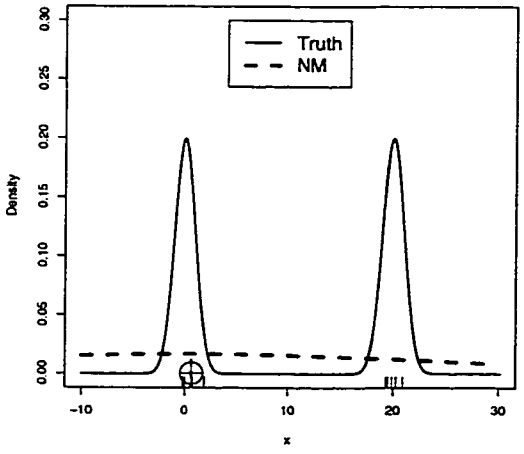
Four samplers were used for the simulations, a Metropolis-Hastings sampler using a custom independence proposal constructed from a mixture of normal distributions, a variable-at-a time Metropolis sampler using a normal proposal, a random walk Metropolis sampler using a multivariate normal proposal distribution, and Normal Kernel Coupler.



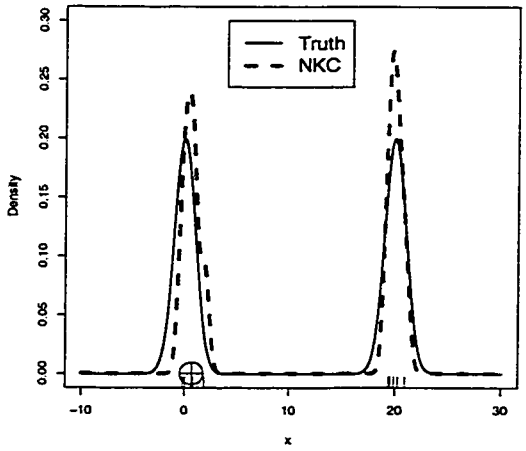
(a) Indep



(b) variable-at-a-time Metropolis



(c) random walk Metropolis



(d) NKC

Figure 2.3: Proposal distributions for the samplers used in the simulation study. Ten points (tick marks along the x-axis) were sampled from a bimodal posterior (solid curve). The broken curve shows the proposal distribution for one of the points, which is marked by the circle. The variance for the proposal distributions and the density estimate for the NKC were computed using the 10 points shown.

We used two versions of each sampler for the simulations. To capture the best possible performance of the individual samplers we constructed the first version using an appropriate function of the true covariance. The second sampler was constructed "adaptively" using a multi-stage tuning method method similar to the one proposed by Raftery & Lewis (1996).

Our multi-stage tuning method used a sequence of runs, each of length 1,000, to estimate the overall and per-mode variance. For the first run, the MCMC samplers were constructed using a preliminary variance estimate. The initial estimates of the variances of each mode was $0.5\,\mathbf{I}_d$. For unimodal distributions, this was also the the estimate of the overall variance. For bimodal distributions, the overall variance estimate was computed using the estimated the mode variance and known locations of the mode centers.

After each 1,000 iterations, new variance estimates were computed. These estimates were then used to construct the samplers for the next set of iterations. This iterative tuning process was performed for one third of the total number of iterations. EG, when the total number of iterations was 10,000, three runs of length 1,000 were used to tune the variance estimates. Once the tuning phase had been completed, the variance remained fixed for the remaining iterations.

At each stage, the overall variance was estimated by computing the variance matrix of the simulation output. For the bimodal targets, the variance of each mode was estimated by dividing the parameter space along the first dimension halfway between the modes. The sample values in each half-space were then used to estimate the cor-

responding mode variance.

The custom independence proposal was constructed from either a single multivariate normal distribution for unimodal targets, or from a mixture of two equally weighted multivariate normal distributions for bimodal targets. Each normal was centered at one of the posterior modes and was assigned either the true or estimated variance of the mode. Note that the independence proposal using the true variance of the modes simulates directly from the true target distribution in every case except the HeavyAndLight target. For the HeavyAndLight target the independence proposal gives equal weight to the two modes when the true distribution gives the smaller (in area) mode seven times more weight. Thus, except in this one case, this independence proposal gives the best possible performance for a sampler which maintains only a single current state.

The random-walk Metropolis sampler was constructed using a d-variate normal distribution. The variance was set to the known or estimated overall variance scaled by $\frac{2.38^2}{d}$, the optimal scaling factor derived by Gelman et al. (1995). The componentwise random-walk Metropolis sampler used a the univariate normal proposal with variance set to the true or estimated marginal variance of the parameter being updated scaled by 2.38^2.

The NKC was constructed as described in section 2.2.1 with the variance matrix $\mathbf{V}$ set to the average of the true or estimated variances of the individual modes. The scaling factor h^2 was set to $1.4\left(\frac{1}{C}\right)^{\frac{2}{4+d}}$.

2.4.3 Setup

Two sets of simulations were run. The first set used target distributions defined in four dimensions, and the second set used distributions defined in twenty dimensions. For both sets of simulations, the NKC used 200 component states. Twenty trials were performed for each combination of sampler, target distribution, and number of components.

The states of each sampler were randomly initialized to one of the modes of the target distribution plus a small random displacement: $X_0^{(c)} \sim N_d(\mu_i, 0.05\mathbf{I})$, where μ_i is the peak of the ith mode. This mimics the practice of initializing the MCMC samplers using modes located via a numerical maximization technique.

Since the primary cost of most MCMC simulations is the expense of evaluating the (unnormalized) density of the target distribution, the results are displayed in terms of the total number of likelihood evaluations rather than in execution time. For the four dimensional simulation, cumulative univariate means and quantiles were computed after 1,000, 2,000, 3,000, 4,000, 8,000, and 10,000 iterations. For the twenty dimensional simulation we computed cumulative univariate means and quantiles after 10,000, 20,000, 30,000, 40,000, 80,000, and 100,000 iterations.

For the samplers which used the true posterior variance, no attempt was made to exclude burn-in iterations since the samplers were started at posterior modes. For the multi-stage tuned samplers, the iterations during the tuning phase (the first 1/3 of the iterations) were discarded when computing later summary measures. Overall mean squared errors (MSEs) were computed from the univariate summary measures

by collapsing across dimensions and simulation runs. Relative efficiency was then calculated as the ratio of the MSE of the sampler of interest over the MSE of the independence proposal.

2.4.4 Results

Summaries of the four dimensional results are given in tables 2.2, 2.3 and 2.4. In addition, Figure 2.4 gives representative plots of the MSE for the estimated mean of each sampler as a function of the number of iterations. Summaries of the twenty dimensional results are given in tables 2.5, A.3 and A.4. Detailed results for each sampler, target, dimension combination are given in appendix A. In the tables and figures, we use the following acronyms: "Indep" for the independence sampler, "CNM" for the componentwise Metropolis sampler, "NM" for the random-walk Metropolis sampler, and "NKC" for the Normal Kernel Coupler.

Table 2.2: All four dimensional distributions: MSE at 10,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution except in one case (HeavyAndLight).

Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	11	121	423	0.95
CNM	True	125,000	314,000	157,900	0.17
NM	True	107,00	33,800	615	0.11
NKC	True	421	155	407	0.44
Indep	3-Stage	1,940	6,710	619	0.74
CNM	3-Stage	125,000	300,000	151,000	0.36
NM	3-Stage	122,000	354,000	181,000	0.54
NKC	3-Stage	176	245	390	0.47

Table 2.3: Unimodal four dimensional distributions: MSE at 10,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution.

Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	12	2,650	2,450	1.00
CNM	True	14,700	25,700	18,800	0.31
NM	True	840	5,800	5,180	0.21
NKC	True	46	1,320	1,550	0.55
Indep	3-Stage	497	3,320	5,590	0.71
CNM	3-Stage	19,600	29,300	32,300	0.31
NM	3-Stage	4,940	7,430	9,170	0.24
NKC	3-Stage	76	1,620	1,290	0.57

Table 2.4: Bimodal four dimensional distributions: MSE at 10,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution except in one case (HeavyAndLight).

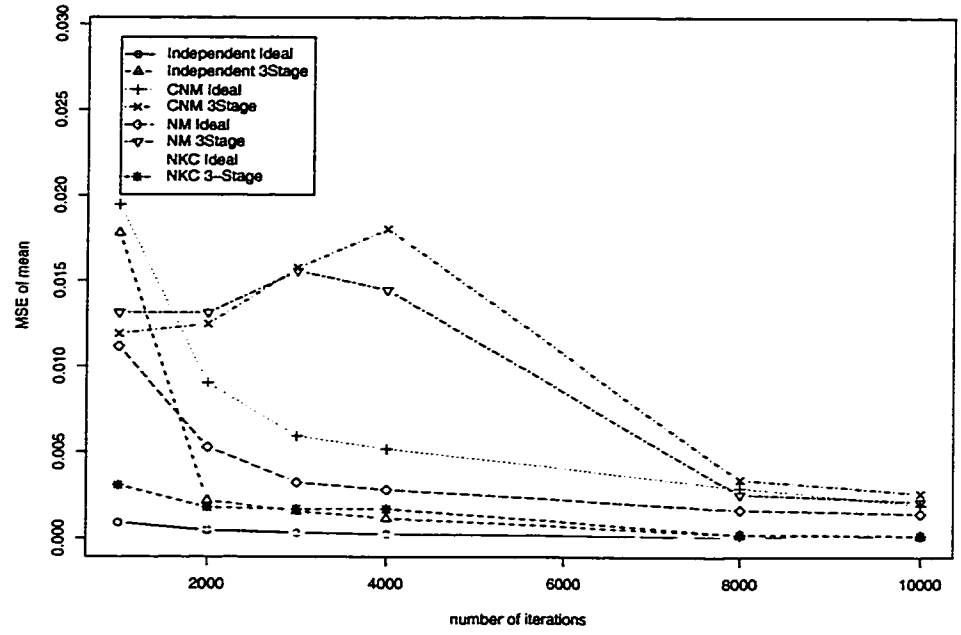
Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	187	138	5,570	0.91
CNM	True	218,000	5,470,000	2,730,000	0.07
NM	True	18,700	587,000	6,880	0.04
NKC	True	733	1,720	5,960	0.36
Indep	3-Stage	3,370	115,000	6,650	0.76
CNM	3-Stage	217,000	5,220,000	2,610,000	0.40
NM	3-Stage	213,000	6,190,000	3,160,000	0.77
NKC	3-Stage	303	3,080	5,860	0.39

Table 2.5: All twenty dimensional distributions: MSE at 100,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution except in one case (HeavyAndLight).

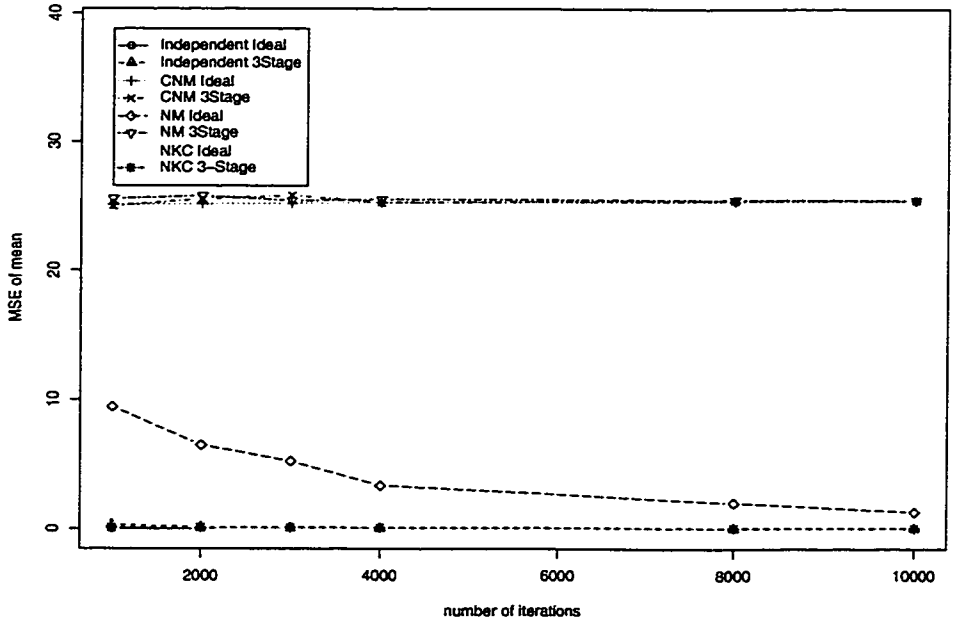
Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	10	1,370	4,520	0.95
CNM	True	1,240,000	3,120,000	1,580,000	0.17
NM	True	996,000	2,400,000	1,090,000	0.09
NKC	True	138,000	25,200	18,700	0.04
Indep	3-Stage	1,360,000	1,180,000	3,610,000	0.24
CNM	3-Stage	1,230,000	3,000,000	1,520,000	0.43
NM	3-Stage	1,240,000	2,960,000	1,600,000	0.31
NKC	3-Stage	190,000	281,000	20,000	0.03

The simulation results show that, as expected, the custom independence proposal gives the lowest MSE when the true covariance of the modes is known. When the true covariance is unknown and must be estimated, the NKC has the best overall performance. The performance benefit of the NKC is more apparent in twenty dimensions than in four and is especially clear for the twenty dimensional multi-modal distributions. Notably, the "tuned" NKC, is more efficient overall than either of the Metropolis samplers constructed using the true covariance of the target distribution. This result holds for both unimodal and bimodal distributions.

When the variance is easy to estimate, as is the case of targets with spherical modes in four dimensions, the independence proposal with the multi-stage tuning method performs at least as well as the NKC. Even in the cases where the tuned independence proposal performed better than the NKC, the performance loss by using the NKC was at most 50%.



(a) "OneMode" Spherical Normal



(b) "HeavyAndLight" One large normal with low probability, and one small normal with high probability

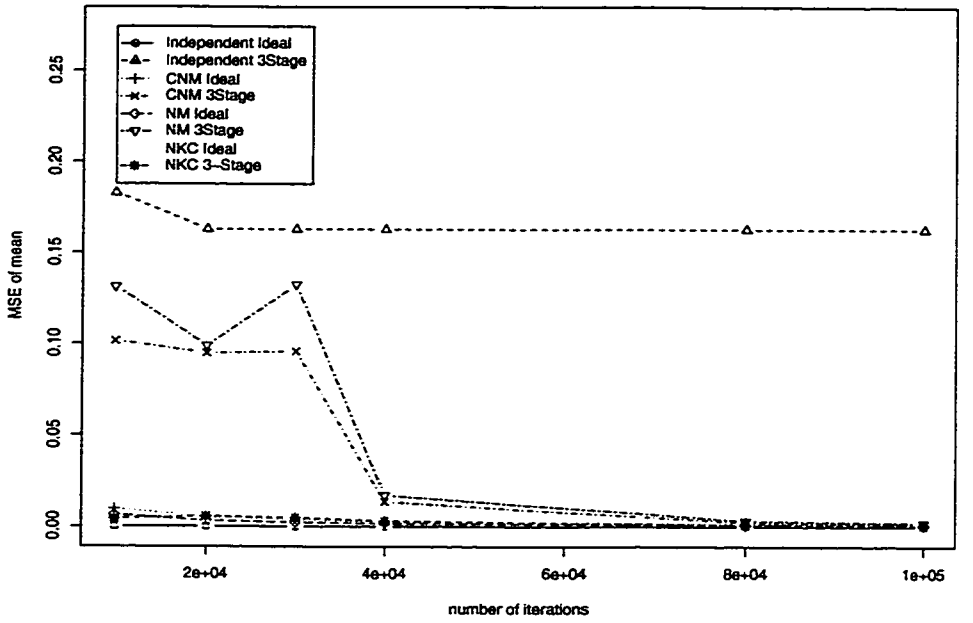
Figure 2.4: MSE for means from the four dimensional simulation. Two representative plots.

Table 2.6: Unimodal twenty dimensional distributions: MSE at 100,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution.

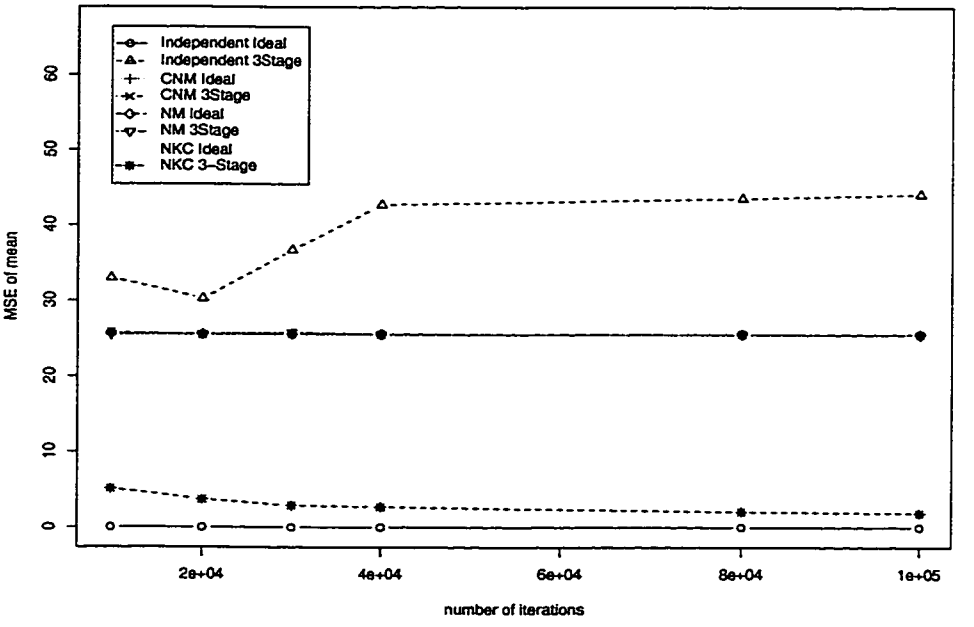
Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	1	3,180	3,150	1.00
CNM	True	18,300	44,200	28,600	0.30
NM	True	3,650	10,200	11,800	0.17
NKC	True	58	11,700	11,600	0.06
Indep	3-Stage	8,480	316,000	309,000	0.37
CNM	3-Stage	16,900	40,100	30,000	0.41
NM	3-Stage	5,710	47,300	39,000	0.30
NKC	3-Stage	1,880	17,700	14,300	0.05

Table 2.7: Bimodal twenty dimensional distributions: MSE at 100,000 iterations. MSE values have been multiplied by 1×10^5 and are accurate to within ±30%. Note that the "Indep" sampler with the "True" variance samples directly from the posterior distribution except in one case (HeavyAndLight).

Sampler	Variance Est.	Mean MSE	2.5% Quantile MSE	97.5% Quantile MSE	Accept Rate
Indep	True	16	13	5,550	0.91
CNM	True	2,160,000	5,430,000	2,740,000	0.07
NM	True	1,740,000	4,190,000	1,900,000	0.03
NKC	True	241,000	35,300	24,100	0.03
Indep	3-Stage	2,380,000	1,820,000	6,090,000	0.13
CNM	3-Stage	2,140,000	5,210,000	2,630,000	0.45
NM	3-Stage	2,160,000	5,150,000	2,770,000	0.31
NKC	3-Stage	331,000	478,000	24,200	0.02



(a) "OneMode" Spherical Normal



(b) "HeavyAndLight" One large normal with low probability, and one small normal with high probability

Figure 2.5: MSE for means from twenty dimensional simulation. Two representative plots.

The NKC seems to have the greatest performance benefit when the variance of the modes is difficult to estimate. The performance benefit of the NKC over the other tuned methods for more difficult problems was often 500% and sometimes several orders of magnitude.

Taken together, the simulation results show that the NKC outperforms the commonly applied random-walk samplers, across a range of problem types and dimensions even when these use the true posterior variance and the "optimal" scaling. The NKC also outperforms the custom independence sampler when the true mode variance is unknown.

2.5 Applied Example

To illustrate the use of the Normal Kernel Coupler we estimate the parameters of a hierarchical mixture model fit to data on the genetic instability of esophageal cancer.

2.5.1 The Problem

Cancer cells undergo a number of genetic changes during neoplastic progression, including loss of entire chromosome sections. When an individual patient has two different alleles for a particular gene, the loss of a chromosome section containing one allele by abnormal cells, termed "Loss of Heterozygosity" (LOH), can be detected using laboratory assays. Chromosome regions with high rates of LOH are hypothesized to contain genes which regulate cell behavior so that loss of these regions disables important cellular controls.

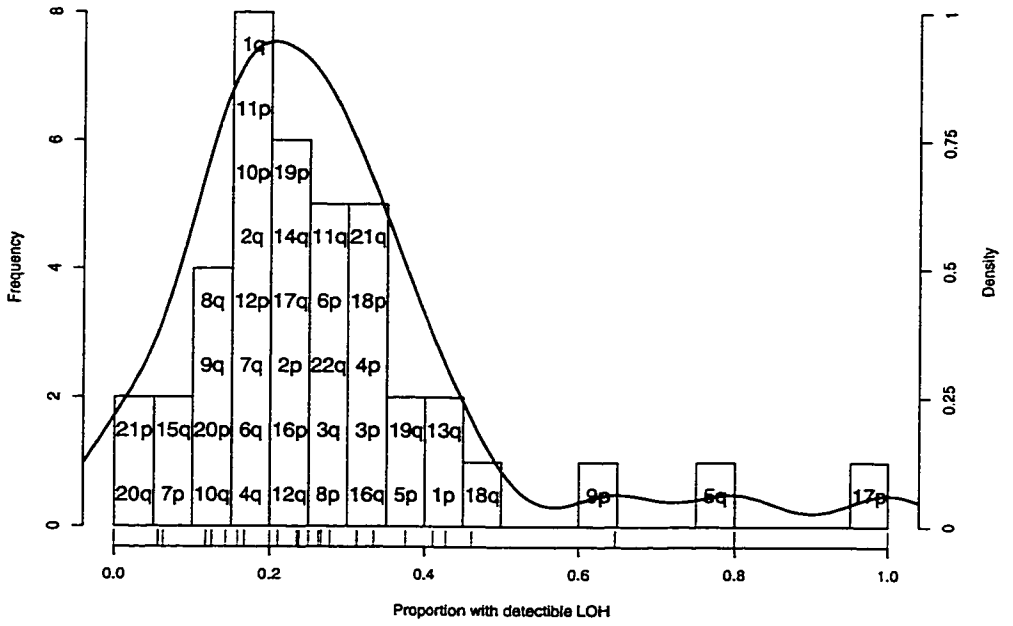


Figure 2.6: Histogram (bars and left axis) and kernel density estimate (curve and right axis) for the Barrett's LOH data. Text labels give the location of each chromosome arm.

The Seattle Barretts Esophagus research project (Barrett et al., 1996) has collected LOH rates from esophageal cancers for 40 regions, each on a distinct chromosome arm. The intent is to locate "Tumor Suppressor Genes" (TSGs), whose deactivation contributes to the development of esophageal cancer (Fearon, 1998; Klein, 1987). Chromosome regions with high rates of LOH ("systematic LOH") are hypothesized to contain TSGs, (Newton et al., 1998; Marshall, 1991). In addition to LOH of regions containing TSGs, there is also a high level of "background" LOH which is thought to be a consequence, rather than a cause, of neoplastic progression. A histogram of the relative frequency of LOH for the Barretts data is shown in table 2.6. The data itself is provided in Appendix B.

The immediate goal of this analysis is to determine the probability of LOH for both the "background" and TSG groups. This will enable the development of a simple discrimination method. Since the labeling of the two groups is unknown, we model the LOH frequency using mixture models, as described by Desai (2000). While several models have been considered, we will focus on a hierarchical Binomial-BetaBinomial mixture model:

$$\begin{aligned} X_i &\sim \eta \text{ Binomial}(N_i, \pi_1) \\ &\quad + (1-\eta) \text{ Beta-Binomial}(N_i, \pi_2, \gamma) \\ \eta &\sim \text{Unif}[0,1] \\ \pi_1 &\sim \text{Unif}[0,1] \\ \pi_2 &\sim \text{Unif}[0,1] \\ \gamma &\sim \text{Unif}[-30,30] \end{aligned}$$

where η is the probability of a location being a member of the binomial group, π_1 is the probability of LOH in the binomial group, π_2 is the probability of LOH in the beta-binomial group, and γ controls the variability of the beta-binomial group (on the logit scale) [1].

We have parameterized the Beta-Binomial so that γ_2 is a variance parameter defined on the range $-\infty \leq \gamma_2 \leq \infty$. As $\gamma_2 \to -\infty$ the beta-binomial becomes a binomial and as $\gamma_2 \to +\infty$ the beta-binomial becomes a uniform distribution on $[0,1]$. This re-

[1]We have also performed the analysis using a Logit(1) prior on γ. This reduced the posterior probability of the smaller mode from 0.047 to to 0.0086 but gave substantively similar results. The results for this model are given in Appendix C.

sults in the unnormalized posterior density

$$p(\eta, \pi_1, \pi_2, \gamma) = \prod_{i=1}^{N} f(x_i, n_i | \eta, \pi_1, \pi_2, \omega_2) \tag{2.8}$$

on the prior range, where

$$\begin{aligned} f(x, n | \eta, \pi_1, \pi_2, \omega_2) &= \eta \binom{n}{x} \pi_1^x (1-\pi_1)^{n-x} \\ &\quad + (1-\eta) \binom{n}{x} \frac{\Gamma(\frac{1}{\omega_2})}{\Gamma(\frac{\pi_2}{\omega_2})\Gamma(\frac{1-\pi_2}{\omega_2})} \frac{\Gamma(x+\frac{\pi_2}{\omega_2})}{\Gamma(n-x+\frac{1-\pi_2}{\omega_2})\Gamma(n+\frac{1}{\omega_2})} \end{aligned} \tag{2.9}$$

and $\omega_2 = \frac{\exp(\gamma)}{2(1+\exp(\gamma))}$.

Unlike most mixture models, where all of the components come from the same parametric family, the proposed model mixes two different distributions; a binomial (with one parameter) and a beta-binomial (with two parameters). We have intentionally omitted fixing which mixture component corresponds to the background group and which corresponds to the TSG group so that we can discover which of the two possible arrangements is better supported by the data. As a consequence, the model has two well separated *non-symmetric* modes, both of which may contribute considerable probability mass. Thus, accurate estimation of this posterior density requires effective sampling from both modes.

2.5.2 Fitting

To locate posterior modes, we employed the Nelder-Mead function maximizer provided with the software package R (Ihaka & Gentleman, 1996). We started the maximizer

from a large number of initial states sampled from the prior. This yielded two well separated, peaks one at (0.903, 0.228, 0.708, 3.54) and another at (0.078, 0.832, 0.230, -18.51). The first mode has log-likelihood of -88.09, while the second is somewhat lower at -90.01. In the absence of other information, we would expect the modes to have posterior probability of approximately $0.87 = 1/(1 + \exp(-90.01 - (-88.09)))$ and $0.13 = 1 - 0.87$, respectively.

Table 2.8: Parameters and likelihood maxima for the Binomial-BetaBinomial model.

		Likelihood Maxima	
Parameter	Description	Mode 1	Mode 2
η	Proportion in Group 1	0.903	0.078
π_1	Group 1 probability of LOH	0.228	0.832
π_2	Group 2 probability of LOH	0.708	0.230
γ_2	Variability of LOH in Group 2	3.54	-18.51
	Log-Likelihood	-88.09	-90.01

We constructed the NKC using 120 component states and initialized half of the states to each of the two local maxima. Starting with the prior variance, we used two preliminary runs of 6,480 iterations (54 complete scans) to estimate the variance of the two modes. We then ran the NKC for 23,640 iterations (197 complete scans) to generate samples for estimation. (See Chapter 3 for a complete description of the fitting process.) Using the output from the NKC, we estimated posterior means and credible regions for the entire posterior distribution as well as for the individual modes.

2.5.3 Results

Using the output from the NKC, we estimated posterior means and credible regions for the entire posterior distribution as well as for the individual modes. Table 2.9 gives

Table 2.9: Means and 95% credible intervals for the Binomial-BetaBinomial model

Overall Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.832	0.82	0.0748	0.965
π_1	0.246	0.257	0.193	0.829
π_2	0.617	0.612	0.23	0.912
γ	12.82	12.3	-21.2	29.3
Prob(Mode 1)	0.970	0.954		
Prob(Mode 2)	0.030	0.047		

Mode 1 Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.854	0.856	0.656	0.966
π_1	0.229	0.229	0.192	0.266
π_2	0.629	0.631	0.318	0.913
γ	13.73	13.7	-4.97	-29.3

Mode 2 Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.091	0.0839	0.0174	0.219
π_1	0.825	0.832	0.741	0.914
π_2	0.232	0.23	0.199	0.261
γ	-16.28	-17.5	-29.5	-4.11

these estimates, as well as gives estimates obtained by direct numerical integration using adaptive quadrature (Berntsen et al., 1991).

Figure 2.7 gives a 2 dimensional histogram constructed from the output of simulation 3 clearly showing the asymmetry of the two modes. We estimate that smaller mode contains only 4.7% of the posterior mass. For this mode, the variance parameter γ is very small (-17.5), forcing the BetaBinomial component to act like a Binomial. The fact that the preferred mode uses the Binomial mixture to explain the background loss rate and that the secondary mode forces the BetaBinomial to act like a Binomial

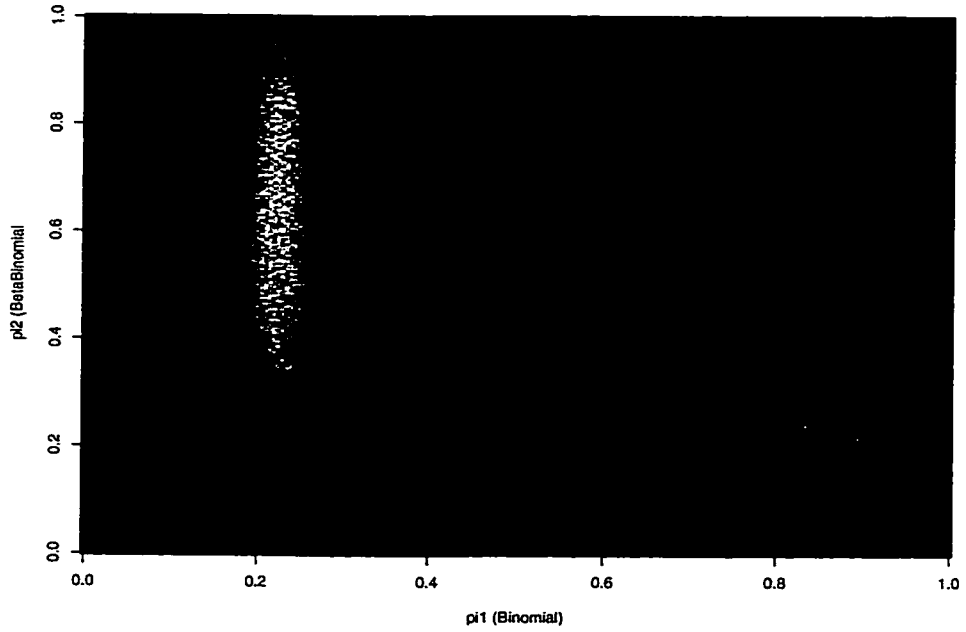
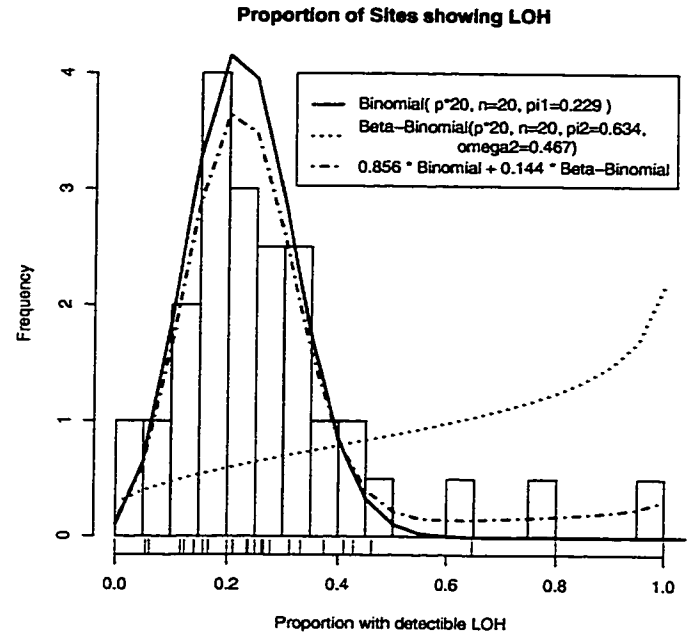


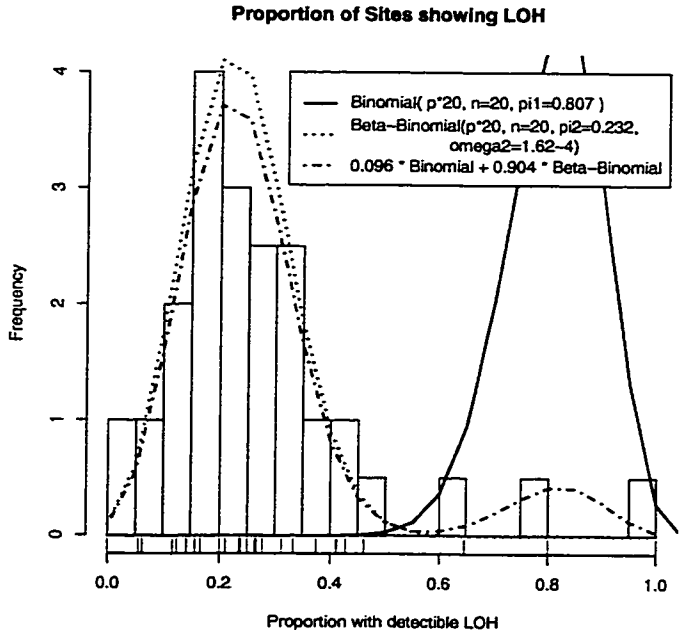
Figure 2.7: 2-d histogram of the joint marginal density of π_1 and π_2 generated from the MCMC simulation. Bins are boxes of side length 0.01 and intensity is proportional to log-frequency.

with the same mean suggests that a binomial model is sufficient for the "background" probability of LOH.

While both modes estimate the LOH probability in the "background" group to be approximately 0.23, the two modes give quite different distributions for the TSG group. Figures 2.8(a) and 2.8(b) plot the fitted distributions for each mode against a histogram of the original data. Figure 2.8(b) shows that the "Binomial-Binomial" mode assigns most of the mass for the TSG group to a binomial which has its density concentrated near 0.83. In contrast, Figure 2.8(a) shows that the larger "Binomial-BetaBinomial" mode, which contains roughly 96% of the posterior probability, spreads the TSG group



(a) Binomial-BetaBinomial Mode



(b) Binomial-Binomial Mode

Figure 2.8: Histogram and fitted distributions (curves) for the Barrett's LOH data

much wider, with considerable probability density over the range of the "background" group.

Figure 2.9 plots the estimated posterior probability of belonging to the TSG group as a function of the LOH rate. Regions with LOH frequencies above 50% almost certainly belong to the TSG group, while regions with loss rates below 50% are more likely to belong to the background group, although there is still a possibility that they belong instead to TSG group.

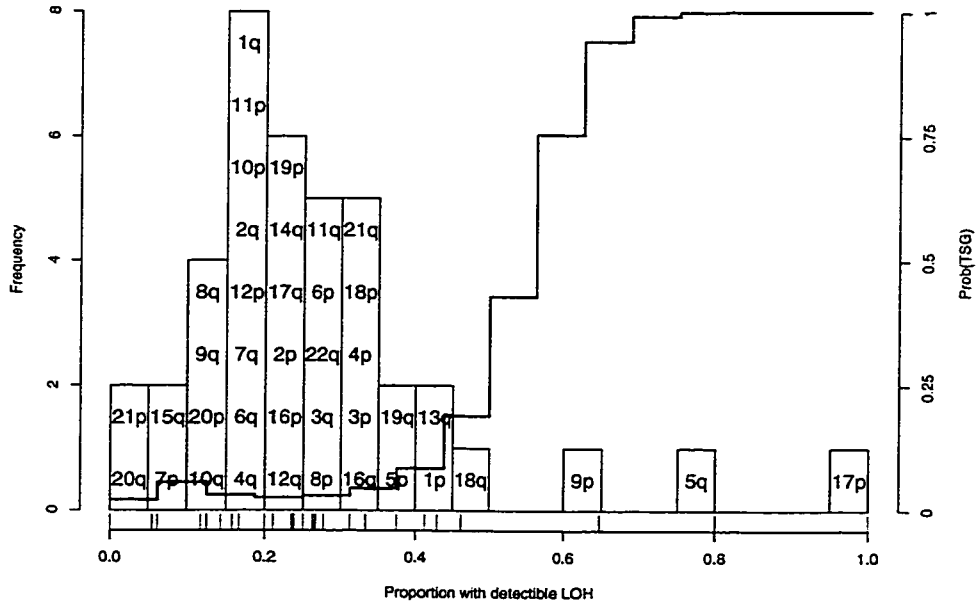


Figure 2.9: Posterior probability (curve, right axis) of membership in the TSG group. The histogram (left axis) shows the observed LOH rate with text labels giving the location of each chromosome arm. Chromosome arms with 50% or higher posterior probability of membership in the TSG group are labeled in red.

Appendix D lists the posterior probability of membership in the TSG group for each of the 40 chromosome arms. Only five have a membership probability above

Table 2.10: Chromosome Arms with more than 10% posterior probability of belonging to the TSG group.

Chromosome Arm	Observed Prob(LOH)	Binomial Quantile	Posterior Prob(TSG)
1p	0.41	0.98	0.14
13q	0.43	0.98	0.15
18q	0.46	0.99	0.21
9p	0.65	> 0.99	0.98
5q	0.80	> 0.99	> 0.99
17p	1.00	> 0.99	> 0.99

10%: 1p, 13q, 9p, 5q and 17p. Of these, only 9p, 5q, and 17p, with memberships probabilities 0.975, >0.999, and >0.999, are more likely to belong to the TSG group than the background group. This aligns well with research on the biology of TSGs and the role of LOH in the inactivation of these genes.

In particular, 17p is the location of the p53 (TP53) gene and 9p is the location of the p16 (CDKN2A) gene. Barrett et al. (1999) showed that LOH of 17p and 9p, along with mutation or hyper-methylation of the remaining p53 or p16 allele is necessary for the development of esophageal cancer from Barretts epithelium. Barrett et al. (1999) also evaluated the role of LOH at 5q, 13q, and 18q in the development of cancer. For these sites they found no evidence that LOH was required for the development of cancer.

2.6 Discussion

In conjunction with the multi-stage tuning strategy, the NKC is a promising tool for efficiently sampling from multi-modal distributions in low and moderate dimensions. A number of practical issues need to be addressed before the NKC can be widely applied.

It is necessary to develop methods for selecting an appropriate number of component states, initializing the current set, and determining of the proper kernel bandwidth (h^2 and V). Further, an appropriate convergence diagnostic is needed. We address these issues in Chapter 3.

For the NKC to be more broadly useful, it will be necessary to explore better methods for tuning the variance of the NKC. The strategy of manually dividing the parameter space into regions which are used to estimate the variance of each modes quickly becomes unmanageable as the number of modes increases.

We are currently exploring methods for adapting the kernel variance V using the current states. This is justified by observing that the essential feature of the proof of Theorem 1 is the infinite support of the normal kernel. While we assumed that the variance matrix $\mathbf{V}$ is fixed, this is not required for the proof to hold. Instead, $\mathbf{V}$ may be replaced with a function $\mathbf{V}_t = \mathbf{V}(X_t^{(\cdot)})$ of the current set provide that $\mathbf{V}_t$ is guaranteed to be positive-definite and bounded in the proper matrix sense.

We have experimented with several alternatives for $\mathbf{V}(X_t^{(\cdot)})$. Modifying the NKC to use a bounded estimate of the variance of the current set performs well for unimodal problems, but very poorly for multi-modal problems. Another approach is to use a function which maximizes the probability of regenerating each element of the current set given the remaining elements. This can be extended by using a different value for each each component state. Both approaches seem promising, but further work is required to characterize their performance and to determine which is preferable.

The current implementation of the NKC is inefficient in high dimensions. In part,

this is a consequence of updating all of the parameters of a given component state as a block. While it would be possible to update only a small subset of the parameters, this is not a reasonable approach in the context of multiple modes because it prevents moves between unconnected modes that do not lie along the coordinate axes. Instead, we are exploring an approach similar to the Adaptive Direction Sampler (Gilks et al., 1994a), where updates are generated on a subspace selected adaptively using the set of current states.

Another method of improving the performance of the NKC in high dimensions is to "retry" failed proposals. The idea, introduced by Tierney & Mira (1999), is to generate a second candidate point from a different proposal distribution if the initial proposal is rejected. This second candidate point is then accepted or rejected using an adjusted acceptance function. By this means, when a NKC proposal is rejected, a standard random-walk Metropolis step can be performed instead. This would increase the overall acceptance rate, and would allow for more local moves than otherwise possible.

We feel that there is great potential for adaptive methods based on the idea of maintaining a set of current states. ADS and NKC are two examples, and a number of other possibilities are evident to us. We hope that additional methods will be proposed by others.

Chapter 3

USING THE NORMAL KERNEL COUPLER

In this chapter we address issues that arise when using the Normal Kernel Coupler in practice. We give guidance on choosing the kernel bandwidth, the number of component states, and starting values. We then provide a run length diagnostic and give an iterative method for tuning the proposal variance and determining the appropriate run length. We conclude with a detailed example of applying the Normal Kernel Coupler to the example introduced in Chapter 2.

3.1 Choosing the Number of Component States

The first issue that arises in applying the NKC to a specific problem is choosing the number of component states. There are several conflicting factors, including the efficiency of the kernel estimator, the MCMC convergence rate, the computational cost, and the accuracy of estimation for small modes.

It is well known that the accuracy of estimates provided by kernel density estimators improves with the number of components (Silverman, 1986; Scott, 1992). Since the NKC uses a kernel density estimate to make proposals, this suggests that the sampler will be more efficient with a larger number of component states.

Balancing this is the observation that the time required for the system as a whole

to converge to the target distribution is a function of the time required to update all of the proposals. This suggests that the total number of components be kept small. Fortunately, this problem can be moderated considerably by starting the NKC at the posterior modes, which can be located efficiently using a numerical maximization technique.

Another factor is the cost of evaluating the proposal probabilities. The cost of evaluating the kernel density estimate is linear in the number of component states. Since this is done twice for each iteration, once for $q(X|Y)$ and once for $q(Y|X)$, the cost of each MCMC iteration increases as a factor of $2C$. In most statistical applications the cost of evaluating the unnormalized density is the dominating cost of each MCMC iteration, so the cost associated with evaluating the kernel density estimate will be negligable unless an extremely large number of states is used.

Finally, the number of component states should be large enough to ensure proper estimation of all modes. In our experience, each mode should be expected to contain at least three components at any given time. Most analyses include estimation of posterior 2.5% and 97.5% quantiles of the parameters. This dictates that we obtain reliable estimates of any mode containing 2.5% or more posterior probability. For a moderate number of modes, 120 component states should be sufficient to ensure that, on average, at least three components occupy any mode with 2.5% or more probability. Setting the number of component states to 200 ensures that the expected number of components occupying such modes will be at least five.

The $120 - 200$ component states suggested by considering the estimation of small

modes appears to be a reasonable value in light of the other issues. It is neither so small so that the overall kernel density estimate is poor (unless the dimensions are large), nor too costly to compute, nor so large as to significantly impair the convergence rate.

3.2 Selecting Which Component State to Update at Each Iteration

There are three possible methods for selecting which component state to update at each iteration, fixed scan, random scan, and random (Gilks & Roberts, 1996). Fixed scan updates the states in a predetermined order. Random scan uses a randomly selected permutation that remains fixed until all components have been updated. Random simply selects one component with uniform probability at each iteration.

We prefer the random-scan method for selecting the update order. It ensures that each component state is updated during each scan, but avoids the potential for undesirable behavior due to a particular order. The random scan method can be implemented by generating a permutation of the integers $\{1 \ldots C\}$ at the start of each scan and following this order until all components have been updated.

During the remainder of the text, we will use the term "iteration" to mean a single pass of the NKC algorithm (see Section 2.2.1), which updates only a single component state. We shall use the term "scan" to mean a set of iterations with length equal to the number of current states. Thus a "scan" is sufficient to update all of the component states, provided that the fixed scan or random scan update method was selected.

3.3 Parameters of the Kernel Density Estimator

The key parameter of the kernel density estimator is the covariance matrix used by the normal kernel. We have parameterized this covariance matrix with two components, a bandwidth h^2 and a covariance matrix $\mathbf{V}$. Together, these determine the shape and scale of the kernel.

3.3.1 Choosing the Covariance Matrix $\mathbf{V}$

For a single multivariate normal target distribution, the optimal normal kernel density estimator (Scott, 1992; Silverman, 1986) is constructed using $\mathbf{V} = \mathrm{Var}(\Pi)$. For target distributions with well separated modes, (eg $X \sim \frac{1}{2}\mathrm{N}(\mu_1,\ 1) + \frac{1}{2}\mathrm{N}(\mu_2,\ 1)$ with $\mu_1 - \mu_2 \geq 6$) the overall covariance is inflated by the distance between the modes. When this covariance is used for $\mathbf{V}$, the kernel density estimate is oversmoothed. As shown in figure 3.1, this leads to a density estimate which poorly reflects the shape of the distribution.

Instead, a covariance matrix should be used which better reflects the structure of the individual modes. When all of the modes share a common covariance matrix, this common covariance is a reasonable value for $\mathbf{V}$. When the individual modes do not have a common variance, the choice of $\mathbf{V}$ is not so clear. In this situation, we have found that setting $\mathbf{V}$ to the average of the individual covariance matrices performs well.

In practice, the true covariance of the modes is unknown and must be estimated. Several approaches are available. When possible, the inverse-Hessian can be com-

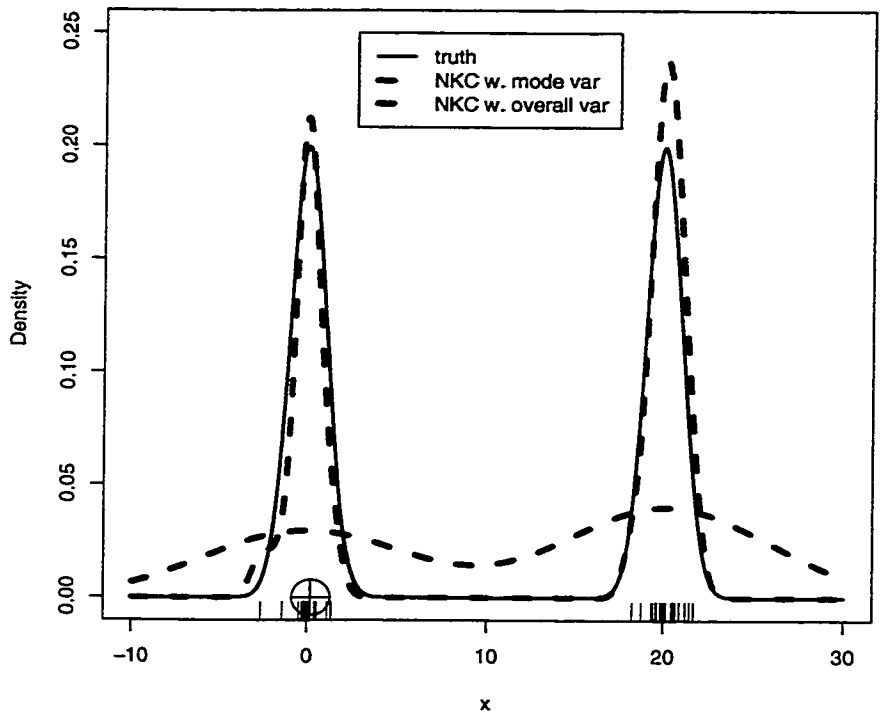


Figure 3.1: Comparison of NKC proposals constructed using mode and overall variance. The proposal curves are based on forty points (hash marks) drawn from the distribution $X \sim \frac{1}{2}\mathrm{N}(0,\ 1) + \frac{1}{2}\mathrm{N}(20,\ 1)$ (solid curve). The mode variance is 1 and the overall variance is 11.

puted at the peak of each mode to provide a reasonable variance estimate. When the Hessian is not available, as with the example introduced in Chapter 2, covariance estimates can be obtained from a sequence of short "tuning" runs. The first run of the NKC uses a preliminary estimate of the mode variance, such as the prior covariance. After the run is completed, estimates of the variance of the modes can be computed using a three step process. First, split the parameter space into regions corresponding to each the modes. Second, divide the simulated points in groups by region. Third, compute the variance within these groups to give estimates of the mode covariances. Averaging

these estimates provides an overall "working" variance for the NKC. We recommend repeating this tuning process a several times to refine the covariance estimates. Section 3.6 illustrates this approach.

3.3.2 Choosing the Bandwidth Parameter h^2

The choice of the kernel bandwidth h^2 is critical to the effectiveness of the kernel density estimate upon which the Normal Kernel Coupler depends. It is well known (Scott, 1992; Silverman, 1986) that the bandwidth of kernel estimators should decrease slowly, on the order of $\left(\frac{1}{C}\right)^{\frac{2}{d+4}}$, as a function of the sample size, in this case, the number of component states, C.

We evaluated several methods for choosing the bandwidth as a function of the number of dimensions and number of component states. We considered four scaling methods:

- the optimal bandwidth for a normal kernel density estimator (Scott, 1992) operating on a multivariate normal target:

$$h_o^2 = \left(\frac{4}{(1+2d)\,C}\right)^{\frac{2}{d+4}}$$

- Terrell's (1990) maximal smoother:

$$h_T^2 = \left[\frac{(d+8)^{(d+6)/2}\,\pi^{d/2}}{16\Gamma[(d+8)/2](d+2)}\frac{1}{C}\right]^{\frac{2}{d+4}}$$

- Constant bandwidth:

$$h_k^2 = k \text{ where } k \text{ is any positive constant}$$

- Constant scaled bandwidth:

$$h_s^2 = k \left(\frac{1}{C}\right)^{\frac{2}{d+4}} \text{ where } k \text{ is any positive constant}$$

Using numerical integration, we evaluated the performance of each of these alternatives for two target distributions, a single multivariate normal and a mixture of two well separated multivariate normals. For each combination of 2 to 20 dimensions and 2 to 500 components states we computed expected step size $\mathbf{E}[||X_t - X_{t+1}||_2]$ and expected acceptance rate, where the expectation is with respect to the true distribution.

For the constant bandwidth and the scaled bandwidth methods, we performed a (crude) numerical search to locate the optimal scale factor. This was accomplished by computing the expectations for the set of scale factors $\{0.5, 0.6, \ldots, 1.9, 2.0\}$. For the constant bandwidth, the best performing value was 1.0. The value 1.4 gave the best performance for the scaled bandwidth. In both cases, the expected distance was relatively insensitive over ± 0.2, and decreased for larger or smaller values.

Overall, the scaled bandwidth generally yielded the largest expected step distance, with the optimal scaling proposed by Scott performing somewhat better for 20 or more dimensions. The unscaled unit bandwidth provided surprisingly good results, while Terrell's maximal smoothing bandwidth performed very poorly in more than 4 dimen-

Table 3.1: Expected step distance for a normal target distribution using 100 current states

Method	Dimension				
	2	4	10	20	50
Terrell's maximal smoother	0.992	0.675	0.053	2.16e-04	< 1.00e-04
Scott's optimal for d-variate normal	1.013	0.826	0.301	0.040	< 1.00e-04
Constant Bandwidth (k=1.0)	0.936	0.818	0.443	0.081	< 1.00e-04
Scaled Bandwidth (k=1.4)	1.036	0.906	0.445	0.094	8.85e-04

Table 3.2: Expected acceptance rate for a normal target distribution using 100 current states

Method	Dimension				
	2	4	10	20	50
Terrell's maximal smoother	0.759	0.454	0.031	1.06e-04	< 1.00e-04
Scott's optimal for d-variate normal	0.816	0.624	0.217	0.029	< 1.00e-04
Constant Bandwidth (k=1.0)	0.707	0.567	0.295	0.055	< 1.00e-04
Scaled Bandwidth (k=1.4)	0.819	0.656	0.300	0.062	5.92e-04

Table 3.3: Expected step distance for a bimodal target distribution using 100 current states

Method	Dimension				
	2	4	10	20	50
Terrell's maximal smoother	3.730	2.271	0.162	5.84e-04	< 1.00e-04
Scott's optimal for d-variate normal	2.364	1.697	0.796	0.255	0.030
Constant Bandwidth (k=1.0)	3.527	2.775	1.345	0.206	1.91e-03
Scaled Bandwidth (k=1.4)	3.829	2.987	1.315	0.252	2.01e-03

Table 3.4: Expected acceptance rate for a bimodal target distribution using 100 current states

Method	Dimension				
	2	4	10	20	50
Terrell's maximal smoother	0.739	0.441	0.031	1.41e-04	< 1.00e-04
Scott's optimal for d-variate normal	0.448	0.325	0.149	0.049	5.47e-03
Constant Bandwidth (k=1.0)	0.695	0.542	0.258	0.040	3.10e-04
Scaled Bandwidth (k=1.4)	0.767	0.590	0.254	0.049	2.76e-04

sions. Given its strong overall performance and its simplicity, we recommend using the scaled bandwidth $h^2 = 1.4\left(\frac{1}{n}\right)^{\frac{2}{d+4}}$, and use this bandwidth for the remainder of this text.

A summary of the results for 100 component states are shown in tables 3.1, 3.3, 3.2, and 3.4. More detailed results are provided in Appendix E. All distances are scaled by $1/\sqrt{d}$ for ease of comparison.

3.4 Determining Run Length

Raftery & Lewis (1992) introduced the `gibbsit` method for determining the required run length of a single MCMC sampler. Our `mcgibbsit` method extends their work by allowing exchangeable MCMC sequences, such as those produced by the NKC, and explicitly adjusting for between-sequence correlation. Our presentation of `mcgibbsit` follows Raftery & Lewis (1992).

Suppose that we want to estimate the posterior quantiles of a function U of the parameters. Given a set of exchangeable MCMC sequences, `mcgibbsit` computes the number of burn-in and sample iterations required to estimate $P(U \leq u|Data)$ to within $\pm r$ with probability s. For example, running `mcgibbsit` with $q = 0.025$, $r = 0.0125$, and $s = 0.95$ generates an estimate of the total number of iterations required to ensure that the cumulative distribution function of the 0.025 quantile of U will be estimated to within ± 0.0125 with probability 0.95.

The `mcgibbsit` algorithm consists of seven steps:

1. Compute U for each of the sampled points:

$$U_t^{(c)} = U(X_t^{(c)})$$

2. Estimate u, the qth quantile of U from the empirical distribution of the combined sequences:

$$u = \min\left\{u^* : \left(\frac{1}{CT}\sum_{c=1}^{C}\sum_{t=1}^{T}\delta(U_t^{(c)} \leq u^*)\right) \geq q\right\}$$

where $\delta(\cdot)$ is the indicator function which takes value 1 when the argument is true, and is 0 otherwise.

3. Discretize the $\{U_t^{(c)}\}$ sequences to form C binary 0-1 processes:

$$Z_t^{(c)} = \begin{cases} 1 & \text{if } U_t^{(c)} \leq q, \\ 0 & \text{otherwise} \end{cases}$$

4. Estimate the thinning constant k necessary to render the discretized sequences approximately Markov.

This is accomplished by finding the smallest value k for which a first order Markov model is preferred to a second order model using the Bayes Information Criterion:

$$BIC = L^2 - 2\log n,$$

where L^2 is the likelihood ratio test statistic and n is the sample size (Schwarz,

1978; Raftery, 1986). Since the component $Z^{(c)}$ sequences are exchangeable, we assume that they are each generated by the same first or second order Markov model. Thus the table of transitions for each model is the component-wise sum of tables computed for each $Z^{(c)}$ sequence.

5. Estimate the transition matrix Q of a single first order Markov chain using the thinned sequences.

   Parameterize the transition matrix Q as

$$Q = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$$

   where α is the probability of changing from 0 to 1, and β is the probability of changing from 1 to 0. The parameters of the transition matrix are again computed by pooling across component chains.

6. Compute the number of burn-in iterations, M, required to guarantee that the m-step transition probabilities for the thinned sequences will be within ϵ of the equilibrium values.

   Note that the equilibrium distribution of the Markov chain with transition matrix Q is

$$\pi = (\pi_0, \pi_1) = (\text{Prob}[U \leq u],\ \text{Prob}[U > u])$$

and the l-step transition matrix is

$$Q^l = \begin{pmatrix} \pi_0 & \pi_1 \\ \pi_0 & \pi_1 \end{pmatrix} + \frac{\lambda^l}{\alpha+\beta} \begin{pmatrix} \alpha & -\alpha \\ -\beta & \beta \end{pmatrix},$$

where $\lambda = (1 - \alpha - \beta)$. Requiring the m-step transition probabilities for the thinned sequences to be within ϵ of the equilibrium values is equivalent to requiring

$$\lambda^m \leq \frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)},$$

which holds when

$$m = \frac{\log(\epsilon(\alpha+\beta))}{\log(\lambda)}.$$

Since it is necessary for the thinned sequence for each component chain to converge, the appropriate burn-in length is $M = C \times m \times k$.

7. Compute the total run length N:

Note that the estimate of Prob$[U \leq u]$ is

$$\overline{\overline{Z}} = \frac{1}{CT} \sum_{c=1}^{C} \sum_{t=1}^{T} \delta(U_t^{(c)} \leq u).$$

Since this mean is taken over the set of C exchangeable sequences, it will be approximately normally distributed with variance

$$\frac{(1+\rho(C-1))}{CT} \frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}$$

where $(1 + \rho(C - 1))$ is the inflation of the variance due to common pairwise correlation ρ between the $Z^{(c)}$ sequences (see Appendix G for the derivation).

To ensure that $\text{Prob}[q - r \leq \overline{\overline{Z}} \leq q + r] = s$ is satisfied, set $N = C \times n$ with

$$n = \frac{(1 + \rho(C-1))}{C} \; \frac{\alpha\beta(2 - \alpha - \beta)}{(\alpha + \beta)^3} \left\{ \frac{r}{\Phi\left(\frac{1}{2}(1+s)\right)} \right\}^{-2}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function. A reasonable estimate of ρ is

$$\begin{aligned} \hat{\rho} &= \frac{\frac{1}{C(C-1)T} \sum_{i=1}^{C} \sum_{j \neq i}^{C} \sum_{t=1}^{T} (Z_t^{(i)} - \overline{\overline{Z}})(Z_t^{(j)} - \overline{\overline{Z}})}{\frac{1}{CT} \sum_{i=1}^{C} \sum_{t=1}^{T} (Z_t^{(i)} - \overline{\overline{Z}})^2} \\ &= \frac{1}{(C-1)} \frac{\sum_{i=1}^{C} \sum_{j \neq i}^{C} \text{Cov}(Z^{(i)}, Z^{(j)})}{\sum_{i=1}^{C} \text{Var}(Z^{(i)})}. \end{aligned}$$

Often, the analyst desires to ensure the (individual) accuracy of quantiles for several posterior quantities. The overall run-length can be obtained by applying `mcgibbsit` to each quantile of interest and taking taking largest combined value of $M + N$.

We have implemented the `mcgibbsit` diagnostic using the `S` statistical language (Becker et al., 1988; Chambers & Hastie, 1993). It may be used with either the `R` (Ihaka & Gentleman, 1996) or `S-PLUS` (Mathsoft, 1998) software packages. Documentation and Source code are provided in appendix F and are also available from the author's web site (Warnes, 2000).

3.5 A Generic Algorithm

The two most difficult aspects of using the NKC are selecting an appropriate proposal variance **V** and determining the required number of iterations. As a practical solution, we recommend an extension of the "3-Simulation" strategy originally proposed by Raftery & Lewis (1996) for the variable-at-a-time Metropolis sampler. Our "3-Simulation" method uses two preliminary MCMC simulations to provide information on the proper run length and proposal variance. The run length is computed using the `mcgibbsit` procedure described in the previous section, while the variance can be computed directly from the MCMC samples. Once the variance and run length have been determined, a third simulation is run. The `mcgibbsit` procedure is then used to determine whether the third stage needs to be repeated with additional iterations. Once `mcgibbsit` reports that the required run length is less than the length of the most recent simulation, inference proceeds using the samples from the final simulation run.

Our algorithm differs that of Raftery & Lewis (1996) in 4 ways. First, `mcgibbsit` is substituted for `gibbsit`. Second, we recommend that the component states for the first simulation be initialized using mode centers obtained by numerical maximization. Third, we suggest that the final state of each simulation phase be used as the initial state for the following phase. Finally, we are using a different sampling method, so apply a different method for estimating the proposal variance.

In detail, the recommended algorithm consists of four stages:

Pre-Simulation

1. Locate the posterior modes

   The mode centers can be obtained efficiently by repeated application of a numerical optimization technique starting from overdispersed starting points. This dramatically reduces the required burn-in since the sampler is initialized using regions that have high probability under the true target distribution.

2. Use `mcgibbsit` to compute the minimum sample size, $N_{\min}$.

Simulation Stage 1

1. Initialize the component states by placing each component at the center of one of the posterior modes.

2. Construct the NKC proposal using initial approximation of the posterior variance.

   Possible initial values for the variance include the mean of the inverse information matrices for each mode (when these can be computed), the variance from a distribution that approximates the posterior, and the marginal variances of the prior distribution. The latter is especially attractive as it is easy to obtain from the prior.

3. Run the NKC for $\sqrt{C} \times N_{\min}$ iterations.

The inflation of $N_{\min}$ by a factor of $\sqrt{C}$ is a heuristic adjustment for initial dependence among the component samplers.

Simulation Stage 2

1. Initialize the component states to the final values from the first simulation.
2. Construct the NKC proposal using a variance estimate, $\hat{V}_2$, computed from the output of the first simulation.

   In computing the variance from the previous simulation runs, it is important to estimate the variance of individual modes rather than the overall variance.
3. Run the NKC for $\sqrt{C} \times N_{\min}$ iterations.
4. Compute M and N for the next simulation by applying `mcgibbsit`.

Simulation Stage 3

1. Initialize the component states to the final values from the previous simulation.
2. Construct the NKC proposal using a variance estimate, $\hat{V}_2$, computed from the output of the previous simulation.
3. Run the NKC for $M + N$ iterations.
4. Compute new estimates of M and N by applying `mcgibbsit`.
5. If the length of the current simulation was *less* than the new estimate of $M + N$, return to the beginning of Simulation Stage 3.

6. If the length of the current simulation was *greater* than the new estimate of $M + N$, use the output of the current stage to perform inference.

This strategy is straightforward and is amenable to automation. Example code using the statistical package R (Ihaka & Gentleman, 1996) and the MCMC simulation library Hydra (see Chapter 4) is available from the authors.

3.6 Applied Example

To illustrate the use of the Normal Kernel Coupler, we will use our "3-Simulation" strategy to estimate the parameters of the Binomial-BetaBinomial model we introduced in Section 2.5.

3.6.1 Pre-Simulation Stage

Locate Posterior Modes

We locate posterior modes by using the Nelder-Mead maximization function included with the R statistical package. We initialize the algorithm using random states drawn from the prior. This locates two well separated modes, one at (0.903, 0.228, 0.708, 3.54) and another at (0.078, 0.832, 0.230 , -18.51). The first mode has log-likelihood of -88.09, while the second is somewhat lower at -90.01.

Compute N_{min}

We wish to make use of posterior 95% credible intervals for the parameters, so we compute $N_{\min}$ by running `mcgibbsit` with $q = 0.025$, $r = 0.0125$, $p = 0.95$ and with

$q = 0.975$, $r = 0.0125$, $p = 0.95$. Both give $N_{\min} = 600$.

3.6.2 Simulation 1

For simulation 1, we use construct a NKC with 120 component states with initial values divided equally between the two mode maximums. For the initial variance we use a diagonal matrix formed from the marginal prior variances:

$$\begin{pmatrix} \frac{1}{12} & 0 & 0 & 0 \\ 0 & \frac{1}{12} & 0 & 0 \\ 0 & 0 & \frac{1}{12} & 0 \\ 0 & 0 & 0 & \frac{60}{12} \end{pmatrix}$$

The NKC is then run for 6,480 iterations, which is the largest factor of 120 smaller than $N_{\min} \times \sqrt{(C)} = 600 \times \sqrt{120} = 6573$.

Estimated means, 0.025 and 0.975 quantiles, as well as the acceptance rate and the results of running `mcgibbsit` are given in Table 3.5. Time series trace and density plots are given in Figure 3.2. The extremely large values produced by `mcgibbsit`, $\max M + N = 6.2e6$, $\max I = 1250$, and $\max R = 28.88$ indicate that the sampler is performing very poorly. This is confirmed by the low acceptance rate of 0.04.

3.6.3 Simulation 2

For simulation 2, we estimate the variance of the two modes by splitting the data from simulation 1 into two subsets using a cutoff of 0.5 for π_1. The two estimated mode

Table 3.5: Means and 95% credible intervals for each simulation stage.

	Adaptive Quadrature	Simulation Run		
		1	2	3
Number of Iterations	640,000	6,480	6,480	23,640
Acceptance Rate		0.04	0.62	0.65
Overall Estimates				
η	0.832	0.714	0.824	0.820
π_1	0.246	0.342	0.253	0.257
π_2	0.617	0.572	0.606	0.612
γ	12.82	-0.338	10.1	12.3
Mode 1 Estimates				
Prob(Mode 1)	0.970	0.811	0.961	0.954
η	0.854	0.861	0.854	0.856
π_1	0.229	0.228	0.229	0.229
π_2	0.629	0.652	0.621	0.631
γ	13.73	3.87	11.3	13.7
Mode 2 Estimates				
Prob(Mode 2)	0.030	0.189	0.039	0.047
η	0.091	0.0805	0.077	0.0839
π_1	0.825	0.829	0.841	0.832
π_2	0.232	0.23	0.229	0.230
γ	-16.28	-18.4	-19.3	-17.5
`mcgibbsit`				
$\max(M+N)$		6.2e6	23,623	11,352
$\max I$		1250	22.7	10.6
$\max R$		28.80	5.60	1.02

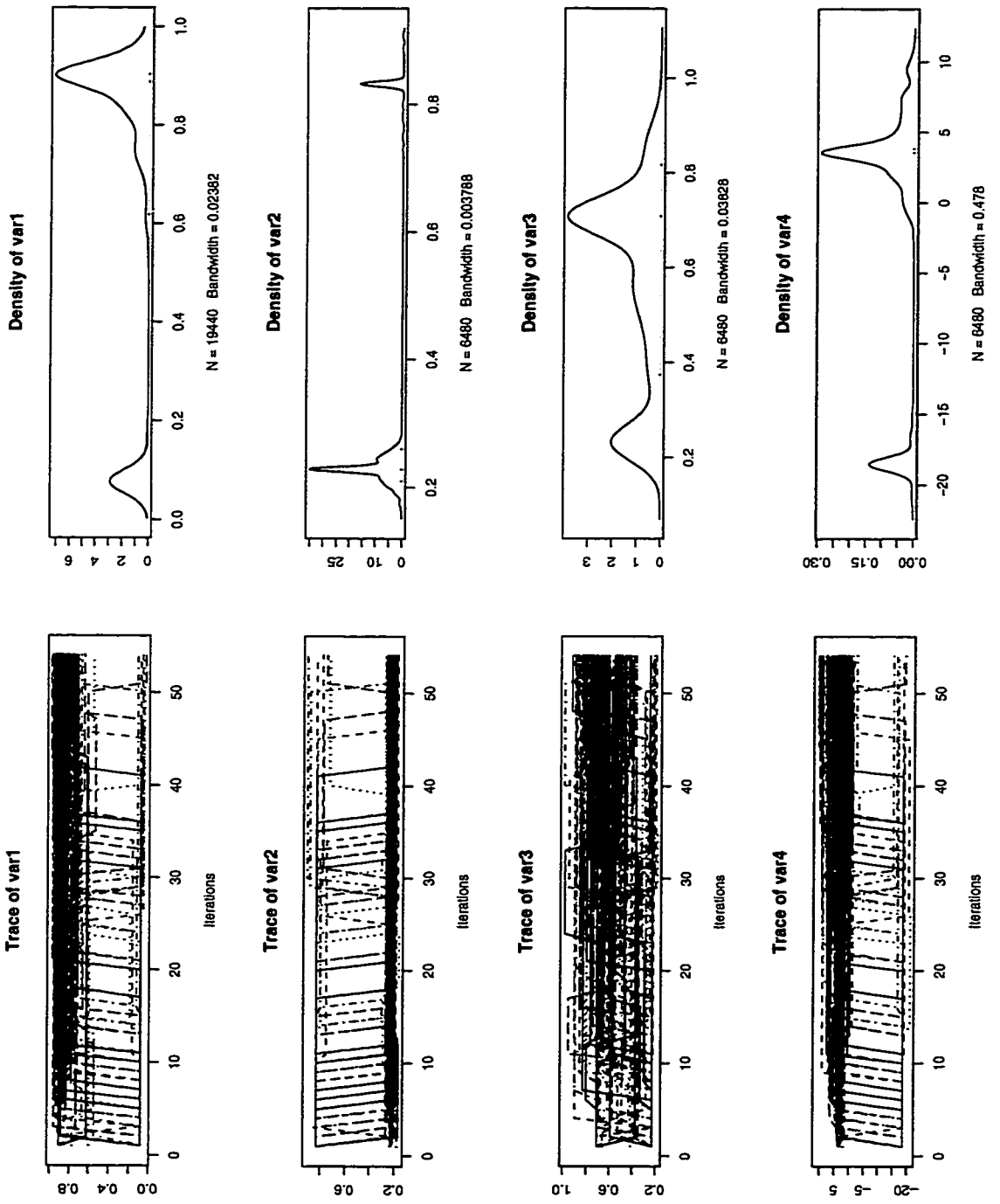


Figure 3.2: Trace and density plots for simulation stage 1

variances are

$$\widehat{\mathrm{Var}}_{m1} = \begin{pmatrix} 3.74 \times 10^{-4} & -9.26 \times 10^{-5} & 3.58 \times 10^{-5} & -1.85 \times 10^{-3} \\ -9.26 \times 10^{-5} & 5.69 \times 10^{-4} & -6.68 \times 10^{-6} & 4.60 \times 10^{-3} \\ 3.58 \times 10^{-5} & -6.68 \times 10^{-6} & 7.90 \times 10^{-5} & -4.69 \times 10^{-3} \\ -1.85 \times 10^{-3} & 4.60 \times 10^{-3} & -4.69 \times 10^{-3} & 0.585 \end{pmatrix}$$

$$\widehat{\mathrm{Var}}_{m2} = \begin{pmatrix} 6.59 \times 10^{-3} & -4.95 \times 10^{-6} & 6.03 \times 10^{-3} & -1.50 \times 10^{-2} \\ -4.95 \times 10^{-6} & 3.33 \times 10^{-4} & 1.94 \times 10^{-5} & -5.84 \times 10^{-4} \\ 6.03 \times 10^{-3} & 1.94 \times 10^{-5} & 2.02 \times 10^{-2} & -4.27 \times 10^{-2} \\ -1.50 \times 10^{-2} & -5.84 \times 10^{-4} & -4.27 \times 10^{-2} & 5.35 \end{pmatrix}$$

both considerably smaller than the prior variance. Averaging these together yields,

$$\mathrm{Var}_2 = \begin{pmatrix} 3.48 \times 10^{-3} & -4.89 \times 10^{-5} & 3.03 \times 10^{-3} & -8.44 \times 10^{-3} \\ -4.89 \times 10^{-5} & 4.51 \times 10^{-4} & 6.37 \times 10^{-6} & 2.01 \times 10^{-3} \\ 3.03 \times 10^{-3} & 6.37 \times 10^{-6} & 1.01 \times 10^{-2} & -2.37 \times 10^{-2} \\ -8.44 \times 10^{-3} & 2.01 \times 10^{-3} & -2.37 \times 10^{-2} & 2.96 \end{pmatrix}$$

which we use to construct the NKC. We then run the NKC for another 6480 iterations starting from the final state of simulation 1. The results of this simulation run are shown in Table 3.5 and Figure 3.3.

Using the new variance matrix dramatically improves the quality of the simulation. Table 3.5 shows that the acceptance rate is now 0.62 and `mcgibbsit` reports a required sample size of 23,623, which seems reasonable considering the need to estimate two

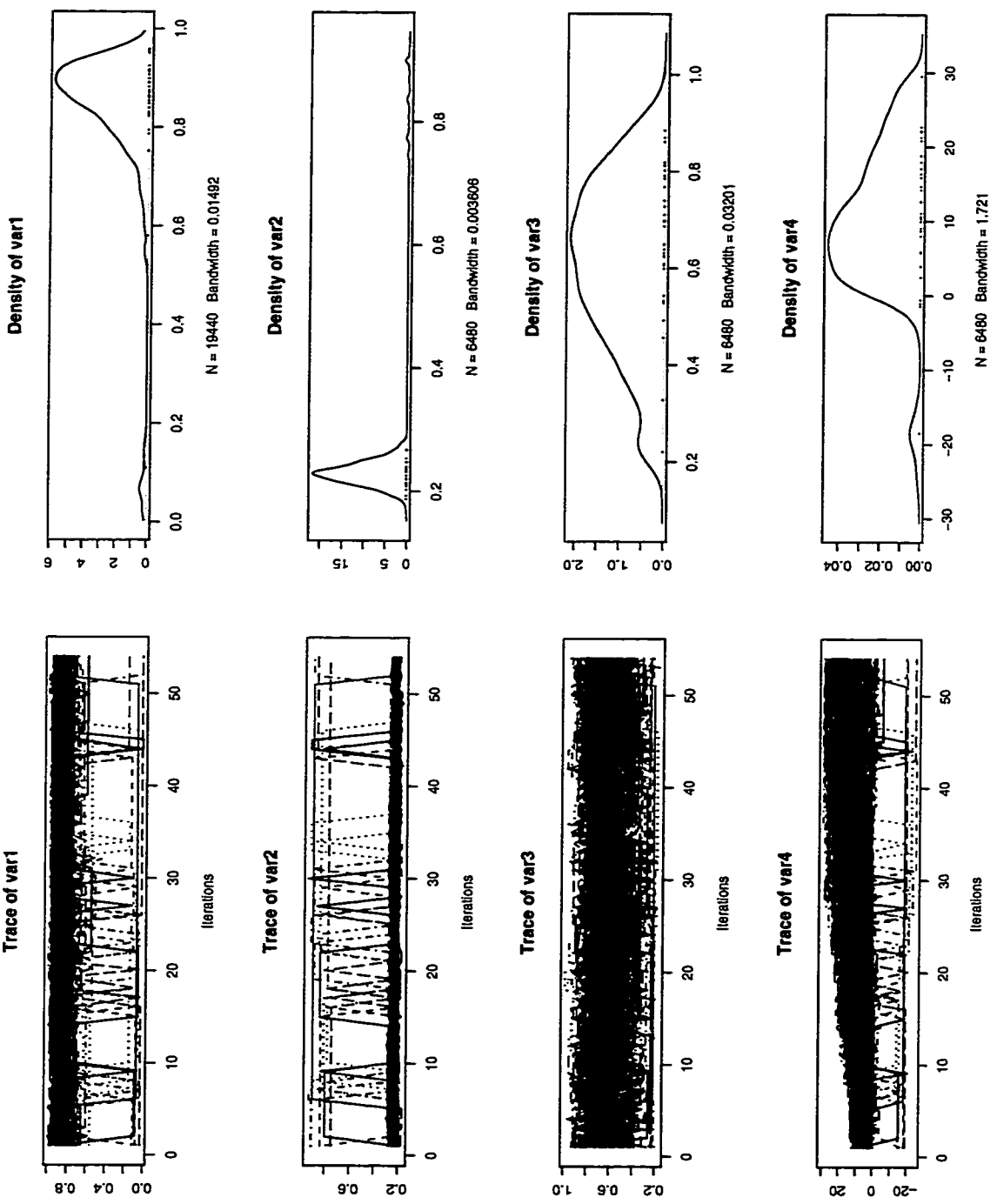


Figure 3.3: Trace and density plots for simulation stage 2

modes, one of which is quite small.

3.6.4 Simulation 3

We again separately estimate the variance of the two modes and then average, obtaining a new proposal variance of

$$\text{Var}_3 = \begin{pmatrix} 4.34 \times 10^{-3} & 6.35 \times 10^{-4} & 3.06 \times 10^{-3} & 1.27 \times 10^{-2} \\ 6.35 \times 10^{-4} & 1.73 \times 10^{-3} & -2.31 \times 10^{-4} & -2.20 \times 10^{-2} \\ 3.06 \times 10^{-3} & -2.31 \times 10^{-4} & 1.31 \times 10^{-2} & -3.80 \times 10^{-2} \\ 1.27 \times 10^{-2} & -2.20 \times 10^{-2} & -3.80 \times 10^{-2} & 36.4 \end{pmatrix}.$$

Using this variance and the final state of simulation 2, we run the NKC for 23,640 iterations. Results from this run are given in Table 3.5 and Figure 3.4. The performance of this simulation is also good, yielding an average acceptance rate of 0.65. `mcgibbsit` now reports that only 11,352 iterations are required. Since this is less than the number of iterations run for the current simulation, we consider the simulation process completed.

3.6.5 Results

We described the fitted values and discussed their scientific relevance in Section 2.5.3. This section focuses on how well the 3-Simulation strategy performed.

First, consider whether `mcgibbsit` achieves the goal of ensuring that the 2.5% and 97.5% quantile estimates are within ± 0.0125 of the true values. Table 3.6 and Figure

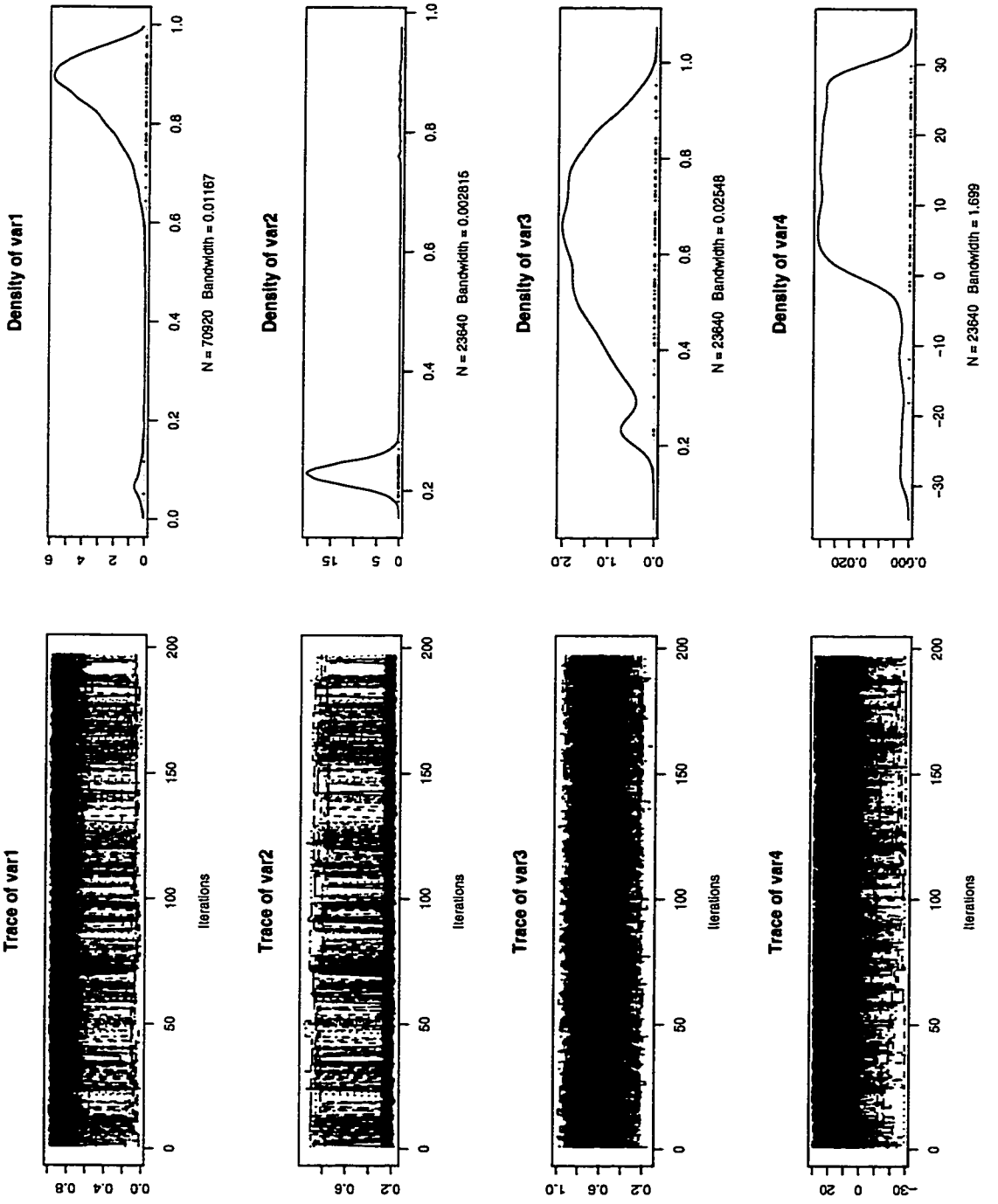


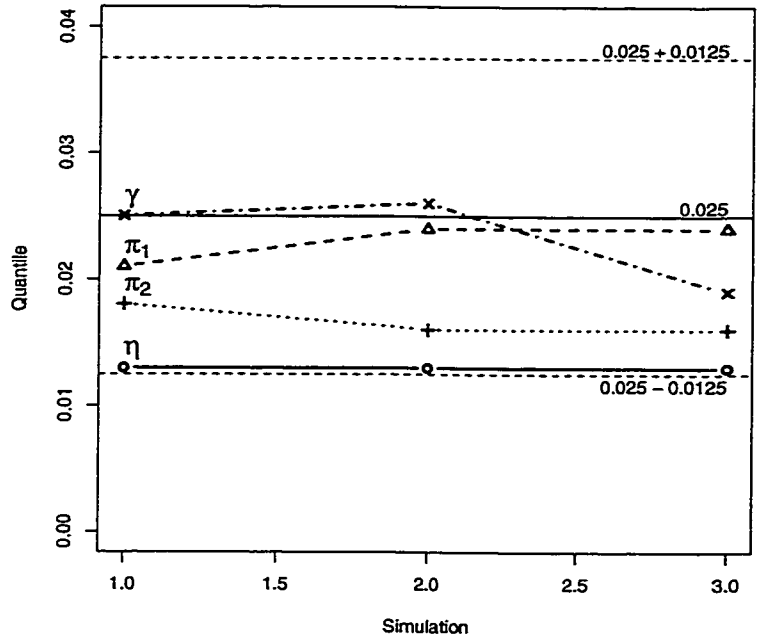
Figure 3.4: Trace and density plots for simulation stage 3

Table 3.6: Means, quantile estimates, and error of quantile estimates from each simulation stage. The error in the quantile estimates was computed using Adaptive Quadrature.

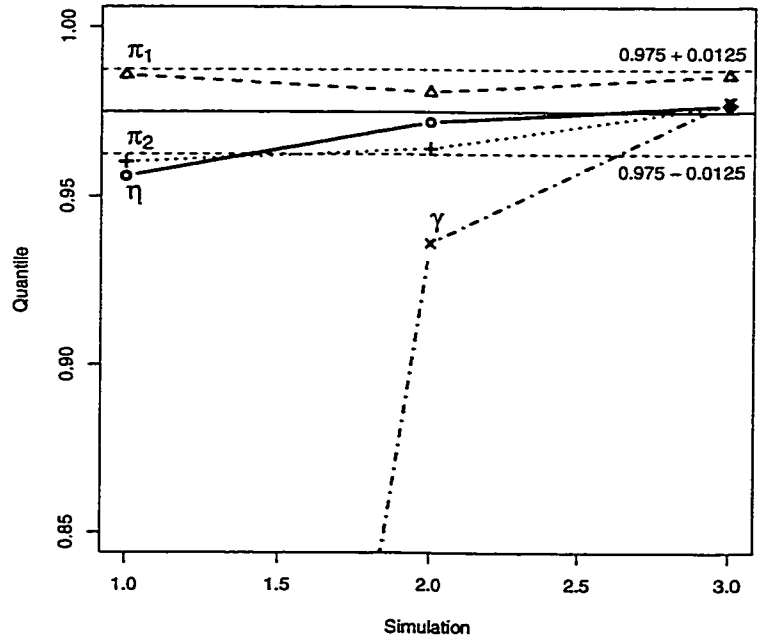
	Est.	Quantile Estimates		Error	
Parameter	Mean	2.5%	97.5%	2.5%	97.5%
Simulation 1					
η	0.714	0.078	0.957	-0.012	-0.019
π_1	0.342	0.192	0.832	-0.004	0.011
π_2	0.572	0.23	0.891	-0.007	-0.015
γ	-0.338	-18.5	9.46	0.001	-0.608
Simulation 2					
η	0.824	0.078	0.963	-0.012	-0.003
π_1	0.253	0.193	0.809	0.001	0.006
π_2	0.606	0.230	0.896	-0.009	-0.011
γ	10.1	-18.1	27.9	0.001	0.039
Simulation 3					
η	0.82	0.0748	0.965	-0.012	0.002
π_1	0.257	0.193	0.829	-0.001	0.011
π_2	0.612	0.23	0.912	-0.009	0.002
γ	12.3	-21.2	29.3	-0.006	0.003

3.5 show that the 2.5% quantile estimates for all four parameters are within the specified ± 0.0125 range after the first simulation stage. While the 97.5% quantile estimate for π_1 is also within the specified range from simulation 1 on, the 97.5% quantile estimate for π_2 and η don't meet the accuracy goal until the completion of simulation stage 2. The last 97.5% estimate to meet the accuracy goal is for γ. It does not do so until the end of simulation 3. These results correspond well to `mcgibbsit`'s recommendations at the end of each stage. It recommended continuing the simulation process after both simulations 1 and 2, and stopping after stage 3.

Second, consider how well the 3-simulation strategy tuned the proposal variance. The acceptance rate, run-length inflation due to serial correlation ("max I"), and run length inflation due to between-chain correlation ("max R") are shown in Table 3.5.



(a) 2.5% Quantiles



(b) 97.5% Quantiles

Figure 3.5: Relative error of the 2.5% and 97.5% quantile estimates on the quantile scale.

The prior variance used for simulation stage 1 gave very poor performance, yielding an acceptance rate of 4% and high run length inflation due to serial correlation (1250) and between chain correlation (28.8). Using the output of this stage to tune the variance improved the performance dramatically. The acceptance rate rose to 62% and the inflation due serial correlation fell to 22.7, a 55-fold decrease. The inflation due to between chain dropped to 5.6, a 5-fold decrease. Updating the variance for simulation 3 provide additional improvement. While the acceptance rate rose only slightly (from 62% to 64%), the inflation due to serial correlation dropped 2 fold to 10.2 and inflation due to the between-chain correlation dropped 5 fold to 1.02. The latter value, which corresponds to a between chain correlation of about 1×10^{-4}, is quite remarkable. When taken together, these results show that the variance tuning was very effective.

Comparing the performance of the NKC to the adaptive quadrature software gives a surprise. The adaptive quadrature software required 640,000 likelihood evaluations before it properly estimated the mass and the parameters of the lower mode, and consequently the overall means. At fewer iterations, the lower mode was incorrectly estimated to have negligible mass. In contrast, the NKC generated estimates that were substantially correct (ie, quantile estimates within ± 0.0125 of the true quantile) after a total of only 36,000 iterations. This is surprising since we expect adaptive numerical integration to work better than MCMC on low-dimensional problems.

3.7 Discussion

In this chapter, we have addressed various issues that arise when using the Normal Kernel Coupler in practice. We described how to implement the Normal Kernel Coupler; provided guidance on choosing the number of component states; described an appropriate run length diagnostic; and defined a general method for selecting starting values, choosing the proposal variance, and selecting the number of simulation iterations. We have illustrated these principles by applying the NKC to a real model for loss of genetic material by esophageal cancer cells.

A practical difficulty remains in estimating the proper proposal variance. For unimodal and obviously bimodal problems, such as our LOH example, it is relatively straightforward to estimate the proper proposal variance by splitting the data along a small subset of parameters. In higher dimensions or when there are more modes, this technique becomes impractical, and is, in any case, difficult to automate. In Section 2.6 we considered several techniques for adaptively selecting the proposal variance. We expect that this work will lead to a variant of the NKC where no variance tuning is required.

Finally, the exchangeable but non-independent parallel sequences produced by the the NKC provide both additional opportunities and lead to additional difficulties for implementing MCMC diagnostics. Two directions seem to hold potential. First, we would like to see the development of a convergence diagnostic along the lines of the Gelman-Rubin (1992) Potential Scale Reduction Factor which adjusts appropriately for the between-chain correlation. Second, the between-chain correlation itself may

provide a convergence diagnostic. The form of the NKC guarantees that the component sequences are jointly converging to independent copies of the original target distribution. As a consequence, the correlation between the sequences necessarily becomes small as the NKC converges.

Chapter 4

THE HYDRA SOFTWARE LIBRARY

HYDRA is an open-source, platform-neutral library for performing Markov Chain Monte Carlo. It implements the logic of standard MCMC samplers within a framework designed to be easy to use and to extend while allowing integration with other software tools. In this chapter, we describe the problem that motivated our work, outline our goals for the HYDRA project, and describe the current features of the HYDRA library. We then provide a step-by-step example of using HYDRA to simulate from a mixture model drawn from cancer genetics, first using a variable-at-a-time Metropolis sampler and then a Normal Kernel Coupler. We conclude with a discussion of future directions for HYDRA.

4.1 Previous Work

As of this writing, two software packages are available that implement Markov Chain Monte Carlo for statistical applications, `WinBUGS` and `FBM`.

`WinBUGS` (Gilks et al., 1992; Gilks et al., 1994b) is a software package for performing Bayesian inference using Gibbs sampling. It provides tools for specifying the model, running the Gibbs sampler, and monitoring convergence using a "point-and-click" graphical interface. A noteworthy feature is that it allows models to be specified

using either a text-based notation or via a graphical model created with the DoodleBUGS interface (Spiegelhalter et al., 1999).

While WinBUGS is mature and is available free of charge from the MRC Biostatistics Unit web site (Stevens, 2000), it has several drawbacks. First, WinBUGS is designed to perform only Gibbs and componentwise Metropolis sampling and does not allow specification of alternative sampling methods. As a consequence, WinBUGS cannot be used when Gibbs sampling or Metropolis-within-Gibbs are inappropriate.

Second, the source code to WinBUGS is not available to the user. This makes it impossible for users to add features to WinBUGS. Thus, users who require features (such as statistical distributions or sampling methods) not provided by WinBUGS are forced to abandon the package entirely. In addition the inability to access the source code prohibits the use of WinBUGS for experimentation with or customization of sampling algorithms. This prevents WinBUGS from being used as a tool for research on MCMC methods.

Radford Neal's FBM ("Flexible Bayesian Modeling") software (Neal, 2000), first released in 1995, is a less well known package that implements a variety of MCMC methods and includes the C source code. While FBM is more flexible than WinBUGS, the FBM documentation and interface are considerably more difficult to understand.

Both WinBUGS and FBM are restricted to specific operating systems. While older versions of BUGS were available for Unix systems, the current version is available only for systems running versions of Microsoft Windows. FBM, on the other hand, is available only for Unix systems. In addition, neither package is integrated with standard

statistical tools. This requires the user to learn the interface of an additional software package in order to use MCMC.

These drawbacks appear to have discouraged or prevented many users from taking advantage of the considerable effort and expertise represented by `WinBUGS` and `FBM`. As evidence of the general dissatisfaction with the available tools, all of the statisticians we have observed using or researching MCMC write their own custom software. This results in considerable duplicated effort. Worse, since properly debugging and verifying software algorithms is a difficult and time-consuming task, it is likely that many of the hand-written software programs contain undetected errors. This can lead to the presentation of faulty analyses. Finally, lack of integrated software support for MCMC has led many applied researchers to avoid Bayesian statistical methods entirely.

4.2 Our Approach

Clearly, there is a need for better MCMC software. Our goal is is to produce a software tool that

1. implements standard MCMC techniques,

2. is easy to use,

3. is reliable,

4. is applicable to a wide variety of problems,

5. allows access to the underlying algorithms,

6. can be easily customized and extended,

7. is integrated with traditional statistical packages, and

8. is available on all common computer platforms.

The HYDRA MCMC library is a first step toward providing software that achieves these goals. HYDRA is a object-oriented library that implements the logic of standard MCMC methods. Although HYDRA can be used directly in custom MCMC programs, is designed to be used as a basis for MCMC software which provides a more user-friendly interface and which is integrated with standard statistical packages.

HYDRA is implemented using Java, a platform independent object-oriented language designed for general programming tasks. We selected Java (Joy et al., 2000) because it enabled the library to meet a number of our stated goals. First, Java's support of formal *interfaces* facilitated the construction of a library that is easy to use while being flexible and easy to extended. In particular, the use of interfaces permits users to provide components of the MCMC algorithm that are tuned to their specific problem. This allows the user to extend the HYDRA package to support new problems or MCMC techniques without changing the existing code. Second, Java provides features that reduce common programming errors and is supported by a wealth of standard libraries and programming tools. Not only do Java's features make it easier to write code that is error-free, but they also make it easier to locate and correct bugs that do exist. This supports construction of a reliable library and frees time otherwise spent

debugging for the development of additional features. Third, Java provides a clear interface for interacting with other languages. This gives a well defined method for HYDRA to be used with existing programming languages and software applications. Although early versions of Java suffered from performance problems, recent versions of the Java virtual machine (runtime) can provide speed comparable to C and C++ for numerical computations (Rijk, 2000; Lewis, 2000; Zachmann, 2000; Schulman, 1997).

The remainder of this text assumes a basic familiarity with the Java language on the order of Flanagan (1997).

4.3 Constructing Metropolis-Hastings Samplers Using HYDRA

HYDRA supports the full generality of Markov Chain Monte Carlo by providing a hierarchy of classes implementing the most common MCMC techniques. Classes exist that implement the general Metropolis-Hastings algorithm, the Metropolis sampler, and the Gibbs sampler. HYDRA also implements the multi-state Adaptive Metropolis Sampling (Gilks & Roberts, 1996) algorithm that forms the basis of Adaptive Direction Sampling (Gilks et al., 1994a) and Normal Kernel Coupling (see Chapters 2 and 3). We will focus on the implementation of the Metropolis-Hastings method (see section 1.1) since it includes the others as special cases.

The `CustomMetropolisHastingsSampler` class implements the logic of Metropolis-Hastings samplers (see Section 1.1) using a target distribution (model), initial state, and proposal distribution specified by the user. This is made possible by requiring the objects representing the target and proposal distributions to provide

certain methods. These methods are defined by the `UnnormalizedDensity` and `GeneralProposal` interfaces, respectively. No restriction is placed on the initial state, save that it be compatible with the user-specified target and proposal distributions.

To allow flexible reporting of the progress of the MCMC sampler, the `CustomMetropolisHastingsSampler` maintains a list of user defined objects that are notified at the completion of the acceptance step of each iteration. When detailed reporting is selected, these "listeners" receive an object containing a great deal of information about each MCMC iteration.

4.3.1 The `UnnormalizedDensity` Interface for Target Distributions

Target distributions implement the `UnnormalizedDensity` interface, which defines two methods:

```
public double unnormalizedPDF    ( Object state );
public double logUnnormalizedPDF( Object state );
```

These methods compute the (log) unnormalized density of the model for the state passed as a parameter.

4.3.2 The `GeneralProposal` Interface for Proposal Distributions

Proposal distributions implement the `GeneralProposal` interface, which has 4 methods:

```
public double conditionalPDF    ( Object next, Object current);
public double logConditionalPDF( Object next, Object current );

public double transitionProbability( Object from, Object to );
public double logTransitionProbability( Object from, Object to );
```

The methods `conditionalPDF` and `logConditionalPDF` compute the probability of generating the object `next` when the current state is `current`. The second two methods perform the same computation, but reverse the arguments[1].

4.3.3 *The `MCMCListener` Interface for Listener Objects*

Objects that are notified at the completion of each MCMC iteration implement the `MCMCListener` interface, which defines one method:

```
public void notify( MCMCEvent event );
```

The parameter of the notify method is an object containing information about the MCMC iteration. This information can be used by the object in various ways. Possibilities include storing the current state to a file, displaying it on a plot, and computing cumulative statistics.

When detailed reporting is disabled, the object passed to `notify` is a `GenericChainStepEvent`. This object has a single field:

```
public Object current;
```

which contains the current state (X_t) of the sampler.

When detailed reporting is enabled, the object passed to `notify` is a `DetailChainStepEvent` which has the additional fields:

```
public Object    proposed;
public Object    last;
public double    lastProb;
public double    proposedProb;
public double    forwardProb;
public double    reverseProb;
public double    probAccept;
public double    acceptRand;
```

[1]The `transitionProbability` and `logTransitionProbability` are depreciated and will not be required in a future release of the software.

Table 4.1: Intepretation of the fields of the `DetailChainStepEvent`

Field	Intepretation
`public Object current;`	X_t
`public Object proposed;`	Y
`public double proposedProb;`	$p(Y)$
`public Object last;`	X_{t-1}
`public double lastProb;`	$p(X_{t-1})$
`public double forwardProb;`	$q(Y\|X_{t-1})$
`public double reverseProb;`	$q(X_{t-1}\|Y)$
`public double probAccept;`	$\alpha(X_{t-1}, Y)$
`public double acceptRand;`	uniform value used to accept/reject
`public boolean accepted;`	was Y accepted?
`public double acceptRate;`	average value of $\alpha(X_{t-1}, Y)$

```
public boolean   accepted;
public double    acceptRate;
```

These fields provide a great deal of information about the MCMC iteration and are useful for debugging and for evaluating the performance of different proposal distributions. The interpretation of each is given in Table 4.1.

4.4 Example

The classes provided by HYDRA can be used directly in compiled Java programs or interactively with various Java-based tools, such as `JPython` and the `Omegahat` statistical language. For ease of presentation, we will focus on the pure Java interface.

We will give an example by using HYDRA construct two different samplers for the Binomial-BetaBinomial mixture introduced in Chapters 2 and 3. We first show how to implement the unnormalized density corresponding to Binomial-BetaBinomial mixture model in Java so that it can be used with HYDRA. Using this model we create a

Table 4.2: Class implementing the hierarchical Binomial Beta-Binomial model for the LOH data.

```
1  package org.omegahat.Simulation.MCMC.Examples;
2
3  import java.lang.Math;
4  import org.omegahat.GUtilities.ArrayTools;
5  import org.omegahat.Probability.Distributions.UnnormalizedDensity;
6
7  public class Binomial_BetaBinomial_SimpleLikelihood implements UnnormalizedDensity
8  {
9    int loh[]={ 7, 3, 4, 3, 5, 4, 5, 3, 6, 12, 5, 3, 1, 3, 5, 3, 11, 2, 2, 2, 3, 5, 3,
10             4, 6, 3, 1, 4, 5, 19, 5, 5, 6, 5, 6, 2, 0, 0, 6, 4 };
11   int   n[]={17, 15, 17, 18, 15, 15, 15, 19, 16, 15, 18, 19, 18, 19, 19, 21, 17, 16,
12            12, 17, 18, 18, 19, 19, 15, 12, 16, 19, 16, 19, 21, 15, 13, 20, 16, 17,
13             8,  7, 18, 15};
14
15   // unnormalized binomial density
16   double udb(int x, int n, double pi) {return Math.pow(pi,x)*Math.pow(1.0-pi, n-x);}
17
18   // unnormalized beta-binomial density
19   double udbb(int x, int n, double pi, double omega) {
20     int r; double tmp0 = 1.0; double tmp1 = 1.0; double tmp2 = 1.0;
21
22     for(r=0; r <= (x     - 1 ); r++) tmp0 *= (      pi + ((double) r) * omega);
23     for(r=0; r <= (n - x - 1 ); r++) tmp1 *= (1.0 - pi + ((double) r) * omega);
24     for(r=0; r <= (n     - 1 ); r++) tmp2 *= (1.0 +      ((double) r) * omega);
25     return ( tmp0 * tmp1 / tmp2 );
26   }
27
28   // unnormalized binomial-betabinomial mixture density
29   double ud_b_bb( int x[], int n[], double eta,
30                   double pi0, double pi1, double omega1) {
31     double retval = 1.0;
32
33     for(int i=0; i < x.length; i++)
34       retval *= (      eta) * udb (x[i], n[i], pi0) +
35                 (1.0 - eta) * udbb(x[i], n[i], pi1, omega1) ;
36     return retval;
37   }
38
39   // Constructor //
40   public Binomial_BetaBinomial_SimpleLikelihood() {}
41
42   // Log Unnormalized Density //
43   public double logUnnormalizedPDF( Object parms)  {
44     return Math.log(unnormalizedPDF( parms ));}
45
46   // Unnormalized Density //
47   public double unnormalizedPDF( Object paramObj ) {
48     double[] parms = ArrayTools.Otod( paramObj );
49     double eta=parms[0], pi0=parms[1], pi1=parms[2], omega1=parms[3];
50
51     // check range
52     if( (eta<0.0) || (pi0<0.0) || (pi1<0.0) || (omega1<0.0) ||
53         (eta>1.0) || (pi0>1.0) || (pi1>1.0) || (omega1>0.5) )
54       return 0.0;
55     else
56       return ud_b_bb(loh, n, eta, pi0, pi1, omega1 );
57   }
58 }
```

variable-at-a-time Metropolis sampler, and then a Normal Kernel Coupler.

4.4.1 Overview

There are four user-specified components of a Metropolis-Hastings sampler: target distribution (model), initial state, proposal distribution, and uniform random number generator. The HYDRA library provides a reliable random number generator and a selection of standard proposal distributions, leaving the user to construct the target distribution and initial state.

4.4.2 Creating a Target Distribution

To create an object representing the target distribution (model), the user needs to write a Java class that implements the `UnnormalizedDensity` interface. For our example problem, we wish to implement a class for the Bayesian hierarchical model

$$\begin{aligned} X_i &\sim \eta\ \text{Binomial}(N_i, \pi_1) \\ &\quad +(1-\eta)\ \text{Beta-Binomial}(N_i, \pi_2, \omega_2) \\ \eta &\sim \text{Unif}(0,1) \\ \pi_1 &\sim \text{Unif}(0,1) \\ \pi_2 &\sim \text{Unif}(0,1) \\ \omega_2 &\sim \text{Unif}(0,1/2) \end{aligned}$$

where the density of the Beta-Binomial distribution is defined as

$$f(X_i|N_i,\pi_2,\omega_2) = \binom{n}{x} \frac{\Gamma(\frac{1}{\omega_2})}{\Gamma(\frac{\pi_2}{\omega_2})\Gamma(\frac{1-\pi_2}{\omega_2})} \frac{\Gamma(x+\frac{\pi_2}{\omega_2})}{\Gamma(n-x+\frac{1-\pi_2}{\omega_2})\Gamma(n+\frac{1}{\omega_2})}$$

A Java class implementing the density for this model is provided in table 4.2. We shall we highlight the programming details that allow use of this class as a target distribution.

First, we need to indicate to the Java compiler where to find the `UnnormalizedDensity` interface that this class will implement. This is accomplished by the line

```
5  import org.omegahat.Probability.Distributions.UnnormalizedDensity;
```

Now we can declare the class and indicate that it implements the `UnnormalizedDensity` interface.

```
7  public class Binomial_BetaBinomial_SimpleLikelihood implements UnnormalizedDensity
```

Next, the class needs a constructor that will accomplish any required initialization, such as observed data. In this case, no initialization is required since the data is hard-coded into the class, so that the line

```
40      public Binomial_BetaBinomial_SimpleLikelihood() {}
```

is sufficient.

Now, our class must provide the `unnormalizedPDF` and `logUnnormalizedPDF` methods. These methods are used by the Metropolis-Hastings sampler to compute the acceptance probability for a given proposed state.

```
42    // Log Unnormalized Density //
43    public double logUnnormalizedPDF( Object parms)  {
44      return Math.log(unnormalizedPDF( parms ));}
45
46    // Unnormalized Density //
47    public double unnormalizedPDF( Object paramObj ) {
48      double[] parms = ArrayTools.Otod( paramObj );
49      double eta=parms[0], pi0=parms[1], pi1=parms[2], omega1=parms[3];
50
51      // check range
52      if( (eta<0.0) || (pi0<0.0) || (pi1<0.0) || (omega1<0.0) ||
53          (eta>1.0) || (pi0>1.0) || (pi1>1.0) || (omega1>0.5) )
54        return 0.0;
55      else
56        return ud_b_bb(loh, n, eta, pi0, pi1, omega1 );
57    }
```

In this example, we have written separate functions that compute the unnormalized density, so these methods simply convert the arguments to the appropriate type (`doubles`), check their range, call the appropriate function.

Note that the the interface defines the argument passed to the `unnormalized-`

PDF and logUnnormalizedPDF as an Object. The user must decide what type of object will represent the model parameters. For most purposes, an array of doubles (double[]) is an appropriate choice. For this reason, the predefined proposal methods (see section I) all operate on arrays of doubles. Any other type of object can be used, however, this requires the user to implement an appropriate proposal distribution.

4.4.3 Creating a Variable-at-a-time Metropolis Sampler

Table 4.3: Class implementing a variable-at-a-time Metropolis sampler for the LOH model.

```
1  package org.omegahat.Simulation.MCMC.Examples;
2
3  import org.omegahat.Simulation.MCMC.*;
4  import org.omegahat.Simulation.MCMC.Proposals.*;
5  import org.omegahat.Simulation.MCMC.Listeners.*;
6  import org.omegahat.Simulation.RandomGenerators.*;
7  import org.omegahat.Probability.Distributions.*;
8
9  public class Binomial_BetaBinomial_SimpleExample {
10   static public void main( String[] argv ) throws Throwable {
11
12     CollingsPRNGAdministrator a = new CollingsPRNGAdministrator();
13     PRNG prng = new CollingsPRNG( a.registerPRNGState() );
14
15     UnnormalizedDensity target = new Binomial_BetaBinomial_SimpleLikelihood();
16
17     double[] diagVar = new double[]{ 0.083, 0.083, 0.083, 0.042};
18
19     SymmetricProposal proposal =
20       new NormalMetropolisComponentProposal(diagVar, prng );
21
22     double[] state = new double[]{0.90, 0.23, 0.71, 0.49 };
23
24     CustomMetropolisHastingsSampler mcmc =
25        new CustomMetropolisHastingsSampler(state, target, proposal,
26                                            prng, true);
27
28     MCMCListener l = new ListenerPrinter();
29     MCMCListenerHandle lh = mcmc.registerListener(l);
30
31     mcmc.iterate( 10 );
32
33   }
34 }
```

Now that we have a class that implements the unnormalized density for the Binomial-BetaBinomial model, we can construct a MCMC sampler. We first implement a variable-at-a-time Metropolis sampler. The complete class file for this sampler is shown in table 4.3.

Again we provide the Java compiler with the locations of the classes we will be using. This time there are five `import` statements:

```
import org.omegahat.Simulation.MCMC.*;
import org.omegahat.Simulation.MCMC.Proposals.*;
import org.omegahat.Simulation.MCMC.Listeners.*;
import org.omegahat.Simulation.RandomGenerators.*;
import org.omegahat.Probability.Distributions.*;
```

After declaring the object, we create a `main` function that will get called to actually do the work of creating and running the MCMC sampler. Within `main`, the first object we need to create is a pseudo-random number generator. The HYDRA library provides an implementation of the Collings random number generator (Collings, 1987), which can be created using the 2 lines:

```
12      CollingsPRNGAdministrator a = new CollingsPRNGAdministrator();
13      PRNG prng = new CollingsPRNG( a.registerPRNGState() );
```

Next, we need to instantiate (create) a copy of our class that implements the unnormalized density of the model. This is accomplished by

```
15      UnnormalizedDensity target = new Binomial_BetaBinomial_Likelihood();
```

Now we instantiate the proposal distribution. For a Metropolis-Hastings sampler, there are several choices (see Appendix I), including a variable-at-a-time random-walk proposal using a normal distribution. This is implemented by the `NormalMetropolisComponentProposal` class. Its constructor allows the specification of a proposal variance for each parameter. We'll use the variance of the parameters under

the prior:

```
22      double[] diagVar = new double[]{ 0.083, 0.083, 0.083, 0.042};
23
24      SymmetricProposal proposal =
25        new NormalMetropolisComponentProposal(diagVar, prng );
```

Now we need to define an initial state for the sampler. We'll use the MLE, which is $\eta = 0.90, \pi_1 = 0.23, \pi_2 = 0.71, \omega_2 = 0.49$:

```
22      double[] state = new double[]{0.90, 0.23, 0.71, 0.49 };
```

With the random number generator, initial state, target distribution, and the proposal distribution defined, we can create the actual sampler:

```
24      CustomMetropolisHastingsSampler mcmc =
25        new CustomMetropolisHastingsSampler(state, target, proposal,
26                                            prng, true);
```

This gives us a working MCMC sampler. The final parameter is an optional flag indicating whether the sampler should report all of the details about the MCMC iteration when it calls the listeners, or whether to just report the new state. We wish to see all of the details, so we provide the value `true`.

We need to attach a listener to the MCMC sampler so that we can see the results of each iteration. There is a variety of predefined listeners (see Appendix J), but we'll start with the simplest listener. Its notify method simply displays the object it receives.

```
28      MCMCListener l = new ListenerPrinter();
29      MCMCListenerHandle lh = mcmc.registerListener(l);
```

Finally, with the MCMC sampler defined and a listener attached, we are ready to run the MCMC sampler. This is accomplished by calling the MCMC sampler's `iterate` method with the number of iterations to perform:

```
31      mcmc.iterate( 10 );
```

4.4.4 Running the Variable-at-a-time Metropolis Sampler

When combined with the HYDRA library, the two classes we've created form a complete Java program. On Unix-like systems with the standard Sun Java tools installed, the classes can be compiled using the `javac` command:

```
> javac Binomial_BetaBinomial_SimpleLikelihood
> javac Binomial_BetaBinomial_SimpleExample
```

Once the classes are compiled, the MCMC sampler can be run using the Java interpreter by

```
> java org.omegahat.Simulation.MCMC.Examples.Binomial_BetaBinomial_SimpleExample
```

This will cause the MCMC sampler to print detailed information about each of the ten iterations to the screen. The output for the first iteration is:

```
Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.71 0.49 ]
Last      Prob   = -359.046964566765
Proposed State   = ContainerState: [ 0.9 0.4751068415506061 0.71 0.49 ]
Proposed Prob    = -423.26454869568283
Current   State  = ContainerState: [ 0.9 0.23 0.71 0.49 ]
Forward Prob     = -0.0369493853787235
Reverse Prob     = -0.0369493853787235
Acceptance Prob = -64.21758412891785
Acceptance Val   = 0.658405257229882
Accepted?        = false
Acceptance Rate = 0.0
```

This gives the current and proposed states, the value of the unnormalized density, the forward and reverse proposal probabilities, the acceptance probability, a flag indicating whether or not the proposed state was accepted, the new state, and the cumulative acceptance rate. Note that the unnormalized density and probabilities are reported on the log scale. The output for all 10 iterations is given in Appendix K

4.4.5 Enhancing the Variable-at-a-time Metropolis Sampler

This example can be enhanced in a number of ways. First, the class can be modified to use a different proposal method. To use a (complete-state) random-walk Metropolis sampler, simply replace the `NormalMetropolisComponentProposal` with a `NormalMetropolisProposal`. Alternatively, the user could define a custom proposal distribution and use it instead.

Second, it is impractical to store and interpret all of the detailed information produced by using the `StepListenerPrinter` listener for more than a few iterations. Instead, we would like to store just the current state to a disk file. This is accomplished by replacing the `StepListenerPrinter` object with a `StrippedListenerWriter`. Change lines 28 and 29 to

```
28      ListenerWriter l = new StrippedListenerWriter("MCMC.output");
29      MCMCListenerHandle lh = mcmc.registerListener(l);
```

and replace line 32 with

```
32      l.close();
```

The `l.close();` command makes sure that the file that is used to store the MCMC output is properly closed once the MCMC iterations are complete.

Now that the output is being stored to a disk file, it is reasonable to increase the number of iterations. Naturally, this is done by changing the value in the `mcmc.iterate` call to the desired value, say 10,000.

Compiling and running the modified class now generates a data file containing 10,000 MCMC iterations. This output can be read into a standard statistical package for computation of diagnostics or to perform inference. For this, we have found the

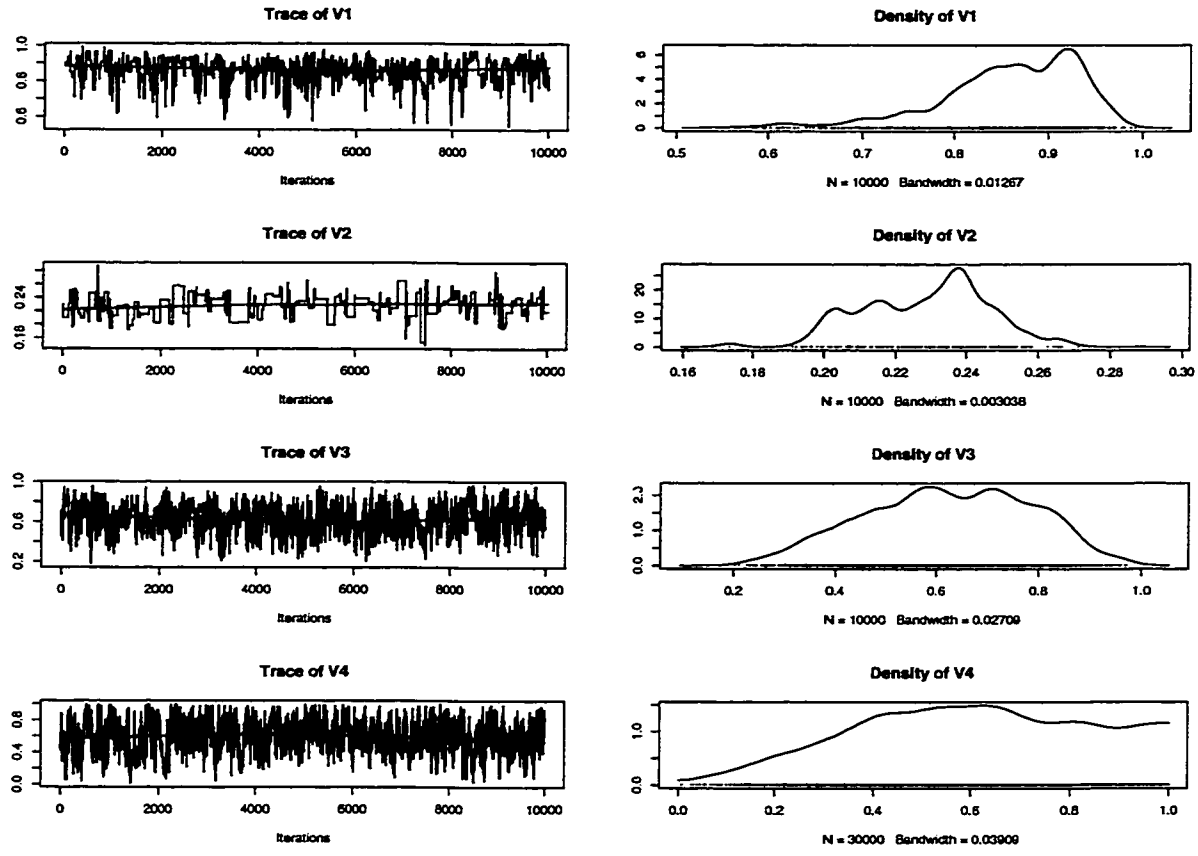


Figure 4.1: CODA plots for 10,000 MCMC iterations.

CODA package of MCMC diagnostics, which exists in versions for both R and S-PLUS, particularly helpful (see the appendix for information on obtaining CODA) . For either version, the commands

```
library(data)
mcmc.data <- mcmc(as.matrix(read.table("MCMC.output")));
```

will load the CODA library (provided it is installed) and properly import the MCMC data. A selection of diagnostics, plots, and summaries is then available. For instance,

the default CODA plots and summary statistics for our 10,000 iterations are shown in Figure 4.1.

4.4.6 Implementing the Normal Kernel Coupler

To implement the Normal Kernel Coupler (NKC) introduced in Chapter 2, only a three changes need to be made to our example class. First, we use a different proposal distribution. Second, we initialize a set of initial values rather than a single value. Third, we use the `CustomHastingsCoupledSampler` class instead of the `CustomMetropolisHastingsSampler` class. The complete source code for the modified class is given in table 4.4.

The proposal distribution for the NKC is implemented by the class `NormalKernelProposal`. Its constructor requires two arguments, a random number generator, and a matrix that specifies the variance for the normal kernel. For our example, the proposal is instantiated by the lines:

```
17      double[][] Var =  {{ 0.003, 0.0   , 0.0   , 0.0   },
18                         { 0.0   , 0.001, 0.0   , 0.0   },
19                         { 0.0   , 0.0   , 0.012, 0.0   },
20                         { 0.0   , 0.0   , 0.0   , 0.007}};
21
22      HastingsCoupledProposal proposal = new NormalKernelProposal(Var, prng );
```

The NKC maintains a set of current states that must be initialized. We use a `MultiDoubleState`, which holds a list of `double` values, to hold the initial values.

```
24      int numComponents = 200;
25
26      MultiDoubleState state0 = new MultiDoubleState ( numComponents );
27      for(int i=0; i < numComponents/2; i++)
28          state0.add( new double[]{0.903, 0.228, 0.708, 0.486 } );
29
30      for(int i=numComponents/2; i < numComponents; i++)
31          state0.add( new double[]{0.078, 0.831, 0.230, 4.5e-9 } );
```

In this case, we've initialized half of the values to each of the two local maxima.

Table 4.4: Class implementing a Normal Kernel Coupler for the LOH model.

```
1  package org.omegahat.Simulation.MCMC.Examples;
2
3  import org.omegahat.Simulation.MCMC.*;
4  import org.omegahat.Simulation.MCMC.Proposals.*;
5  import org.omegahat.Simulation.MCMC.Listeners.*;
6  import org.omegahat.Simulation.RandomGenerators.*;
7  import org.omegahat.Probability.Distributions.*;
8
9  public class Binomial_BetaBinomial_SimpleExample_NKC {
10   static public void main( String[] argv ) throws Throwable {
11
12     CollingsPRNGAdministrator a = new CollingsPRNGAdministrator();
13     PRNG prng = new CollingsPRNG( a.registerPRNGState() );
14
15     UnnormalizedDensity target = new Binomial_BetaBinomial_SimpleLikelihood();
16
17     double[][] Var =  {{ 0.003, 0.0  , 0.0  , 0.0  },
18                        { 0.0  , 0.001, 0.0  , 0.0  },
19                        { 0.0  , 0.0  , 0.012, 0.0  },
20                        { 0.0  , 0.0  , 0.0  , 0.007}};
21
22     HastingsCoupledProposal proposal = new NormalKernelProposal(Var, prng );
23
24     int numComponents = 200;
25
26     MultiDoubleState state0 = new MultiDoubleState ( numComponents );
27     for(int i=0; i < numComponents/2; i++)
28         state0.add( new double[]{0.903, 0.228, 0.708, 0.486 } );
29
30     for(int i=numComponents/2; i < numComponents; i++)
31         state0.add( new double[]{0.078, 0.831, 0.230, 4.5e-9 } );
32
33     CustomHastingsCoupledSampler mcmc =
34         mcmc = new CustomHastingsCoupledSampler( state0, numComponents,
35                                                  target, proposal, prng,
36                                                  false);
37
38     ThinningProxyListener pL  = new ThinningProxyListener(numComponents);
39     MCMCListenerHandle    pLh = mcmc.registerListener(pL);
40
41     MCMCListenerWriter l = new StrippedListenerWriter("NKC.output");
42     MCMCListenerHandle lh = pL.registerListener(l1);
43
44     mcmc.iterate( 10000 );
45
46     l.close();
47   }
48 }
```

The logic of multi-state MCMC samplers is implemented by the `CustomHastings-CoupledSampler` class. This class is instantiated using 6 parameters, the set of initial states, the number of current states to maintain, the target (model) distribution, the proposal distribution, a random number generator, and a flag indicating whether to report the details of the iteration:

```
33      CustomHastingsCoupledSampler mcmc =
34          mcmc = new CustomHastingsCoupledSampler( state0, numComponents,
35                                                   target, proposal, prng,
36                                                   true);
```

Although we could have simply used a `StrippedListenerWriter` this would generate very large output file by writing out the entire set of 200 current states at each iteration. Instead, we use using `ThinningProxyListener` class, which "thins" the events it receives by a specified factor before passing them on:

```
38      ThinningProxyListener pL  = new ThinningProxyListener(numComponents);
39      MCMCListenerHandle    pLh = mcmc.registerListener(pL);
40
41      MCMCListenerWriter l = new StrippedListenerWriter("NKC.output");
42      MCMCListenerHandle lh = pL.registerListener(l1);
```

Running the sampler now will output the complete state once every 200 iterations.

4.5 Conclusions and Future Directions

Our example has shown that the HYDRA MCMC library makes it easy to create different Metropolis-Hastings samplers without extensive programming. This should encourage additional statisticians to experiment with and use the Metropolis-Hastings method.

We hope that the HYDRA library will form the basis of a set of MCMC tools that are easy to use, robust, and complete. In particular we intend to integrate HYDRA with the

statistical tools `R`, `Splus`, and `SAS`, as well as the new `Omegahat` statistical computing language (Temple Lang, 2000; Chambers, 2000; Bates et al., 2000). These interfaces promise to provide flexible and powerful interactive environments for MCMC.

Other goals for the HYDRA library include

- visual tools for specifying and monitoring MCMC simulations
- support for distributed/parallel computing
- a library of target distributions corresponding to common statistical models, such as GLM's and mixture models.

Chapter 5

CONCLUSION

5.1 Overview

This text introduced the Normal Kernel Coupler, a conceptually simple method for sampling from posterior distributions that can be applied whether the target distribution has one or several modes. We have proven that the NKC is a ergodic MCMC sampler for any continuous distribution on d-dimensional Euclidean space. We have also shown that the NKC outperforms standard random-walk Metropolis samplers and a custom independence sampler when the true variance is unknown. In fact, the NKC outperforms the random-walk Metropolis samplers constructed using the true posterior variance.

Further, have shown how the NKC can be applied in practice. We have shown how to select the bandwidth (h^2) and variance ($\mathbf{V}$) parameters of the normal kernel and have introduced `mcgibbsit`, a run-length diagnostic that is appropriate for the NKC. We also provided a systematic method for selecting the kernel variance and the run length. We have also given direction for selecting other parameters of the NKC. We have demonstrated these methods on a real example using a model with two distinct and dissimilar modes. The results from fitting this model using the NKC compare favorably with with those obtained by adaptive quadrature at much lower computa-

tional cost. Finally, we presented the HYDRA software library and showed that HYDRA makes it simple to apply MCMC samplers, including the NKC.

5.2 Future Work

In its current form, the NKC is well suited for continuous distributions in low and moderate dimensions. To become more broadly useful, it is necessary both to extend the NKC to problems in higher dimensions and to provide a better automatic method for selecting the kernel variance. In Section 2.6 we considered several techniques for adaptively selecting the proposal variance. We expect that this work will lead to a variant of the NKC which is both more efficient and simpler to use.

We have introduced the `mcgibbsit` run-length diagnostic and applied it to the NKC. We have shown that it works well on our applied example, but further exploration of its properties for other problems is merited. This diagnostic is applicable to any multi-sequence MCMC sampler with exchangeable sequences. We would like to explore its performance for other multi-state and multi-chain samplers, including multiple independent MCMC runs.

Finally, we hope to extend our work on the HYDRA MCMC library to provide a tool for Markov Chain Monte Carlo that is both efficient and easy to use. Because HYDRA is implemented in Java it should be straightforward to provide interfaces to HYDRA within common statistical packages. By removing the need to perform custom programming or to learn a new statistical tool, this offers the potential to make it as easy to use MCMC for statistical problems as it is to use generalized linear models.

BIBLIOGRAPHY

BARRETT, M. T., GALIPEAU, P. C., SANCHEZ, C. A., EMOND, M. J., & REID, B. J. (1996). Determination of the frequency of loss of heterozygosity in esophageal adenocarcinoma by cell sorting, whole genome amplification and microsatellite polymorphisms. *Oncogene* **12**, 1873–1878.

BARRETT, M. T., SANCHEZ, C. A., PREVO, L. J., WONG, D. J., PAULSON, T. G., RABINOVICH, P. S., & REID, B. J. (1999). Evolution of neoplastic cell lineages in Barrett oesophagus. *Nature Genetics* **22**, 106–109.

BATES, D., CHAMBERS, J., COOK, D., DALGAARD, P., GENTLEMAN, R., HORNIK, K., IHAKA, R., LEISCH, F., LUMLEY, T., MACHLER, M., MASAROTTO, G., MURRELL, P., NARASIMHAN, B., RIPLEY, B., SAWITZKI, G., TEMPLE LANG, D., TIERNEY, L., & VENABLES, B. (2000). The Omega project for statistical computing. web site. `http://www.omegahat.org/`.

BECKER, R. A., CHAMBERS, J. M., & WILKS, A. R. (1988). *The New S Language*. Wadsworth, Pacific Grove, California.

BERNTSEN, J., ESPELID, T. O., & GENZ, A. (1991). An adaptive multidimensional integration routine for a vector of integrals. *Transactions on Mathematical Software* **17**, 452–456.

CHAMBERS, J. M. (2000). Users, programmers, and statistical software. *Journal of Computational and Graphical Statistics* .

Chambers, J. M. & Hastie, T. J., editors (1993). *Statistical Models in S*. Chapman & Hall, New York.

CHAN, K. S. & GEYER, C. J. (1994). Comment on "Markov chains for exploring posterior distributions". *The Annals of Statistics* **22**, 1747–1758.

COLLINGS, B. J. (1987). Compound random number generators. *Journal of the American Statistical Association* **82**, 525–527.

DESAI, M. (2000). *Mixture Models for Genetic Changes in Cancer Cells*. PhD thesis, University of Washington.

FEARON, E. R. (1998). Tumor suppressor genes. *The Genetic Basis of Human Cancer* **7**, 145.

FLANAGAN, D. (1997). *Java in a Nutshell*. O'Reilly & Associates, second edition.

GELFAND, A. E. & SAHU, S. K. (1994). On Markov chain Monte Carlo acceleration. *Journal of Computational and Graphical Statistics* **3**, 261–276.

GELMAN, A., ROBERTS, G., & GILKS, W. (1995). Efficient Metropolis jumping rules. In Berger, J. O., Bernardo, J. M., Dawid, A. P., & Smith, A. F. M., editors, *Bayesian Statistics V*. Oxford University Press.

GELMAN, A. & RUBIN, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science* **7**, 473–483.

GEMAN, S. & GEMAN, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721–741.

GEYER, C. J. (1991). Markov chain Monte Carlo maximum likelihood. In Kerami-das, E. M., editor, *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, pages 156–163, Seattle, WA. Interface Foundation of North America.

GEYER, C. J. & THOMPSON, E. A. (1994). Annealing Markov chain Monte Carlo with applications to ancestral inference. Technical Report 589, School of Statistics, University of Minnesota.

GILKS, W. R., RICHARDSON, S., & SPIEGELHALTER, D. J. E. (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall.

GILKS, W. R. & ROBERTS, G. O. (1996). Strategies for improving MCMC. In *Markov Chain Monte Carlo in Practice*, pages 89–114. Chapman & Hall.

GILKS, W. R., ROBERTS, G. O., & GEORGE, E. I. (1994a). Adaptive direction sampling. *The Statistician* **43**, 179–189.

GILKS, W. R., ROBERTS, G. O., & SAHU, S. K. (1998). Adaptive Markov chain

Monte Carlo through regeneration. *Journal of the American Statistical Association* **93**, 1045–1054.

GILKS, W. R., THOMAS, A., & SPIEGELHALTER, D. J. (1992). Software for the Gibbs sampler. In *Computing Science and Statistics. Proceedings of the 24rd Symposium on the Interface*, pages 439–448. Interface Foundation of North America (Fairfax Station, VA).

GILKS, W. R., THOMAS, A., & SPIEGELHALTER, D. J. (1994b). A language and program for complex Bayesian modeling. *The Statistician* **43**, 169–177.

GIVENS, G. H. & RAFTERY, A. E. (1996). Local adaptive importance sampling for multivariate densities with strong nonlinear relationships. *Journal of the American Statistical Association* **91**, 132–141.

HAARIO, H., SAKSMAN, E., & TAMMINEN, J. (1999). Adaptive proposal distributions for random walk Metropolis algorithm. *Computational Statistics* **14**, 375–395.

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

IHAKA, R. & GENTLEMAN, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**, 299–314.

JOY, B., STEELE, G., GOSLING, J., & BRACHA, G. (2000). *The*

Java Language Specification. Addison-Wesley, second edition. also at `http://java.sun.com/docs/books/jls`.

KLEIN, G. (1987). The approaching era of the tumor suppressor genes. *Science* **238**, 1539–1544.

LEWIS, J. P. (2000). Java versus C/C++ benchmarks. web site. `http://www.idiom.com/~zilla/Computer/javaCbenchmark.html`.

MARIANI, E. & PARISI, G. (1992). Simulated tempering: A new Monte Carlo scheme. *Europhysics Letters* **19**, 451–458.

MARSHALL, C. J. (1991). Tumor suppressor genes. *Cell* **64**, 313–326.

Mathsoft (1998). *S-PLUS 5 for UNIX User's Guide*. Data Analysis Products Divison, Mathsoft, Seattle, Washington.

METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., & TELLER, A. H. (1953). Equations of state calculations by fast computing machine. *Journal of Chemical Physics* **21**, 1087–1091.

NEAL, R. (2000). Software for flexible Bayesian modeling and Markov chain sampling. Web Site. `http://www.cs.toronto.edu/~radford/fbm.software.html`.

NEAL, R. M. (1996). Sampling from multimodal distributions using tempered transitions. *Statistics and Computing* **6**, 353–366.

NEWTON, M. A., GOULD, M. N., REZNIKOFF, C. A., & HAAG, J. D. (1998). On the statistical analysis of allelic–loss data. *Statistics in Medicine* **17**, 1425–1445.

OEHLERT, G. W. (1998). Faster adaptive importance sampling in low dimensions. *Journal of Computational and Graphical Statistics* **7**, 158–174.

RAFTERY, A. E. (1986). A note on Bayes factors for log–linear contingency table models with vague prior information. *Journal of the Royal Statistical Society, Series B* **48**, 249–250.

RAFTERY, A. E. & LEWIS, S. M. (1992). How many iterations in the Gibbs sampler? In Bernardo, J. M., Berger, J. O., Dawid, A. P., & Smith, A. F. M., editors, *Bayesian Statistics 4. Proceedings of the Fourth Valencia International Meeting*, pages 763–773. Clarendon Press (Oxford).

RAFTERY, A. E. & LEWIS, S. M. (1996). Implementing MCMC. In *Markov Chain Monte Carlo in Practice*, pages 115–130. Chapman & Hall.

RIJK, C. (2000). Binaries vs byte–codes. *Ace's Hardware (web site)* `http://www.aceshardware.com/Spades/read.php?article_id=153`.

SCHEAFFER, R. L., MENDENHALL, W., & OTT, L. (1990). *Elementary Survey Sampling*. Duxbury Press, fourth edition.

SCHULMAN, A. (1997). Java on the fly. *Webreview.com (web site)* `http://webreview.com/wr/pub/97/07/25/grok/`.

SCHWARZ, G. (1978). Estimating the dimension of a model. *Annals of Statistics* **6**, 461–464.

SCOTT, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons.

SILVERMAN, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall.

SPIEGELHALTER, D. J., THOMAS, A., & BEST, N. G. (1999). *WinBUGS Version 1.2 User Manual*. MRC Biostatistics Unit.

STEVENS, A. (2000). The BUGS project. Web Site. `http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml`.

TEMPLE LANG, D. (2000). The Omega project: New possibilities for statistical software. *Journal of Computational and Graphical Statistics* .

TERRELL, G. (1990). The maximal smoothing principle in density estimation. *Journal of the American Statistical Association* **85**, 470–477.

TIERNEY, L. (1994). Markov chains for exploring posterior distributions. *Annals of Statistics* **22**, 1701–1762.

TIERNEY, L. (1996). Introduction to general state–space markov chain theory. In *Markov Chain Monte Carlo in Practice*, pages 59–74. Chapman & Hall.

TIERNEY, L. & MIRA, A. (1999). Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in Medicine* **18**, 2507–2515.

TJIELMELAND, H. & HEGSTAD, B. K. (2000). Mode jumping proposals in MCMC. *Scandinavian Journal of Statistics* (to appear).

WARNES, G. R. (2000). Hydra mcmc web site. Web Site. `http://www.warnes.net/MCMC`.

WEST, M. (1993). Approximating posterior distributions by mixtures. *Journal of the Royal Statistical Society, Series B* **55**, 409–422.

ZACHMANN, G. (2000). Java/C++ benchmark. web site. `http://www.igd.fhg.de/~zach/benchmarks/`.

ZHANG, P. (1996). Nonparametric importance sampling. *Journal of the American Statistical Association* **91**, 1245–1253.

Appendix A

DETAILED SIMULATION RESULTS

Table A.1: MSE at 10,000 iterations by target distribution, 4 dimensions

MSE and efficiency for means

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000122	1.66e-05	8.60e-05	2.91e-05	0.000170	5.17e-05
CNM Ideal	0.00196	0.000288	0.192	0.0744	0.248	0.0690
NM Ideal	0.00146	0.000261	0.00146	0.000514	0.0223	0.00380
NKC Ideal	0.000116	1.79e-05	0.000225	5.94e-05	0.00106	0.000309
Independent 3Stage	0.000192	2.62e-05	0.000450	0.000258	0.0143	0.00664
CNM 3Stage	0.00264	0.000319	0.321	0.0847	0.264	0.0478
NM 3Stage	0.00219	0.000376	0.00114	0.000447	0.145	0.0406
NKC 3-Stage	0.000196	3.65e-05	0.000174	5.67e-05	0.00190	0.000323

MSE and efficiency for 2.5% Quantiles

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000821	9.39e-05	0.000650	0.000113	0.0779	0.00252
CNM Ideal	0.0133	0.00170	0.436	0.134	0.324	0.0683
NM Ideal	0.00643	0.000602	0.00623	0.00150	0.161	0.0245
NKC Ideal	0.00128	0.000271	0.000989	0.000285	0.0374	0.00421
Independent 3Stage	0.00119	0.000151	0.00861	0.00570	0.0898	0.0304
CNM 3Stage	0.0174	0.00283	0.470	0.136	0.392	0.0961
NM 3Stage	0.00940	0.00106	0.00537	0.00144	0.208	0.0600
NKC 3-Stage	0.00171	0.000255	0.00146	0.000260	0.0454	0.00588

MSE and efficiency for 97.5% Quantiles

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000732	0.000116	0.000779	0.000147	0.0721	0.00193
CNM Ideal	0.0138	0.00205	0.304	0.0717	0.246	0.0997
NM Ideal	0.00489	0.000822	0.00328	0.000583	0.147	0.0529
NKC Ideal	0.00121	0.000187	0.00159	0.000333	0.0438	0.00451
Independent 3Stage	0.00153	0.000239	0.0162	0.0130	0.150	0.0747
CNM 3Stage	0.0147	0.00188	0.602	0.192	0.352	0.0622
NM 3Stage	0.00846	0.00135	0.00782	0.00173	0.259	0.0689
NKC 3-Stage	0.00148	0.000291	0.00199	0.000368	0.0351	0.00390

Table A.2: MSE at 10,000 iterations by target distribution, 4 dimensions

MSE and efficiency for means

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.00173	0.000511	0.00167	0.000515	0.00239	0.000767	0.00169	0.000524
CNM Ideal	20.2	0.0749	20.2	0.0529	25.5	6.97	21.2	0.984
NM Ideal	0.134	0.0354	5.26	1.39	1.30	0.341	0.772	0.226
NKC Ideal	0.00509	0.00167	0.258	0.0321	0.0164	0.00470	0.0140	0.00421
Independent 3Stage	0.00327	0.00119	0.00218	0.000702	0.00425	0.00134	1.34	1.06
CNM 3Stage	20.3	0.0530	20.2	0.0302	25.5	6.99	20.7	0.836
NM 3Stage	19.2	1.04	20.3	0.0371	25.6	7.00	20.3	0.0320
NKC 3-Stage	0.00767	0.00180	0.0276	0.0102	0.0636	0.0194	0.0221	0.00909

MSE and efficiency for 2.5% Quantiles

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.00100	0.000117	0.00100	0.000117	0.00272	0.000521	0.000811	9.72e-05
CNM Ideal	46.0	8.81	62.3	12.0	53.3	10.0	57.2	11.0
NM Ideal	0.0216	0.00265	5.30	5.19	17.9	8.16	0.246	0.0448
NKC Ideal	0.00164	0.000148	0.0169	0.00197	0.0370	0.00746	0.0132	0.00280
Independent 3Stage	0.00204	0.000309	0.00185	0.000304	0.00365	0.000695	4.59	4.70
CNM 3Stage	45.3	8.68	62.0	11.9	53.0	9.94	48.7	9.45
NM 3Stage	52.6	11.5	67.9	13.0	58.3	11.0	68.8	12.7
NKC 3-Stage	0.00252	0.000498	0.00896	0.00107	0.0894	0.0229	0.0222	0.00359

MSE and efficiency for 97.5% Quantiles

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000964	0.000162	0.0946	0.000479	0.126	0.00106	0.000847	0.000142
CNM Ideal	30.5	8.74	20.7	5.92	20.8	5.95	37.3	10.6
NM Ideal	0.0358	0.00443	0.0775	0.00986	0.122	0.00954	0.0397	0.00754
NKC Ideal	0.00208	0.000239	0.0914	0.000945	0.125	0.00174	0.0201	0.00383
Independent 3Stage	0.00161	0.000262	0.0941	0.000616	0.126	0.00123	0.0437	0.0356
CNM 3Stage	30.4	8.73	20.7	5.91	20.8	5.97	32.5	9.24
NM 3Stage	33.8	9.51	22.8	6.70	22.9	6.73	46.9	12.8
NKC 3-Stage	0.00245	0.000370	0.0916	0.00127	0.128	0.00136	0.0117	0.00254

Table A.3: MSE at 100,000 iterations by target distribution, 20 dimensions

MSE and efficiency for means

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	9.28e-06	8.29e-07	8.06e-06	1.88e-06	1.19e-05	2.57e-06
CNM Ideal	0.00112	7.37e-05	0.169	0.0356	0.379	0.0692
NM Ideal	0.000579	3.82e-05	0.000595	0.000165	0.108	0.0115
NKC Ideal	0.000291	2.73e-05	0.000493	0.000159	0.000964	0.000141
Independent 3Stage	0.163	0.0733	0.000143	2.35e-05	0.0913	0.0192
CNM 3Stage	0.00146	0.000101	0.225	0.0560	0.281	0.0423
NM 3Stage	0.00197	0.000262	0.0404	0.0165	0.129	0.00533
NKC 3-Stage	0.000584	3.73e-05	0.000700	0.000159	0.0551	0.00888

MSE and efficiency for 2.5% Quantiles

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	7.20e-05	5.44e-06	6.36e-05	8.67e-06	0.0951	0.000598
CNM Ideal	0.00591	0.000336	0.639	0.108	0.681	0.114
NM Ideal	0.00196	0.000111	0.00171	0.000215	0.301	0.0594
NKC Ideal	0.00160	8.94e-05	0.00252	0.000511	0.348	0.0260
Independent 3Stage	4.07	0.118	3.84	0.00264	1.57	0.233
CNM 3Stage	0.00728	0.000526	0.577	0.133	0.620	0.0895
NM 3Stage	0.00419	0.000298	1.08	0.121	0.332	0.0470
NKC 3-Stage	0.00349	0.000208	0.00351	0.000390	0.525	0.0552

MSE and efficiency for 97.5% Quantiles

	Target					
	OneMode		Narrow		Banana	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	6.70e-05	5.37e-06	8.24e-05	1.18e-05	0.0942	0.000660
CNM Ideal	0.00573	0.000429	0.378	0.105	0.475	0.0872
NM Ideal	0.00167	9.89e-05	0.00116	0.000147	0.352	0.0483
NKC Ideal	0.00155	0.000125	0.00167	0.000250	0.344	0.0297
Independent 3Stage	3.94	0.0563	3.84	0.00264	1.47	0.189
CNM 3Stage	0.00831	0.000858	0.461	0.120	0.431	0.0937
NM 3Stage	0.00467	0.000479	0.918	0.0824	0.246	0.0431
NKC 3-Stage	0.00346	0.000265	0.00403	0.000894	0.423	0.0409

Table A.4: MSE at 100,000 iterations by target distribution, 20 dimensions

MSE and efficiency for means

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000142	6.13e-05	0.000135	5.81e-05	0.000211	9.65e-05	0.000157	7.52e-05
CNM Ideal	20.3	0.0202	20.2	0.0130	25.6	7.00	20.5	0.925
NM Ideal	3.51	0.927	20.3	0.00627	25.6	7.01	20.3	0.112
NKC Ideal	0.00470	0.00110	0.126	0.0268	9.45	0.458	0.0671	0.0230
Independent 3Stage	14.7	1.88	18.1	1.50	44.2	5.91	18.2	1.40
CNM 3Stage	20.3	0.0169	20.2	0.00945	25.6	7.01	19.7	0.564
NM 3Stage	20.2	0.0410	20.2	0.0252	25.5	7.00	20.3	0.0613
NKC 3-Stage	0.0217	0.00461	1.60	0.236	1.89	0.320	9.74	1.58

MSE and efficiency for 2.5% Quantiles

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	8.24e-05	4.80e-06	8.24e-05	4.80e-06	0.000278	5.30e-05	8.75e-05	8.65e-06
CNM Ideal	45.5	8.72	62.0	11.9	53.0	9.95	56.7	10.9
NM Ideal	0.173	0.0877	68.0	13.1	53.4	10.0	46.1	8.77
NKC Ideal	0.00296	0.000224	0.0299	0.00237	0.506	0.0191	0.872	0.0508
Independent 3Stage	12.2	7.48	26.8	11.2	14.9	7.73	19.0	9.31
CNM 3Stage	45.4	8.70	61.9	11.9	53.0	9.95	48.3	9.34
NM 3Stage	45.4	8.70	61.9	11.9	52.9	9.97	45.7	8.69
NKC 3-Stage	0.00701	0.000635	0.161	0.0156	0.149	0.0179	18.8	8.37

MSE and efficiency for 97.5% Quantiles

	Target							
	TwoMode		BigAndSmall		HeavyAndLight		TwoNarrow	
Sampler	MSE	Std Err	MSE	Std Err	MSE	Std Err	MSE	Std Err
Independent Ideal	0.000103	1.02e-05	0.0949	0.000106	0.127	0.000161	0.000101	1.43e-05
CNM Ideal	30.3	8.72	20.6	5.89	20.7	5.92	38.1	10.7
NM Ideal	0.0628	0.0163	20.5	5.91	20.7	5.94	34.8	10.0
NKC Ideal	0.00328	0.000243	0.00636	1.65e-05	0.00883	0.000148	0.946	0.0914
Independent 3Stage	56.9	11.9	51.0	10.2	52.1	9.14	83.6	11.1
CNM 3Stage	30.3	8.71	20.6	5.88	20.6	5.91	33.9	9.70
NM 3Stage	31.3	8.99	21.5	6.15	21.5	6.17	36.5	10.5
NKC 3-Stage	0.00778	0.000749	0.0333	0.00252	0.0829	0.00430	0.843	0.162

Appendix B

BARRETT'S LOH DATA

Chromosome Arm	# with LOH	# Informative
1p	7	17
1q	3	15
2p	4	17
2q	3	18
3p	5	15
3q	4	15
4p	5	15
4q	3	19
5p	6	16
5q	12	15
6p	5	18
6q	3	19
7p	1	18
7q	3	19
8p	5	19
8q	3	21
9p	11	17
9q	2	16
10p	2	12
10q	2	17
11p	3	18
11q	5	18
12p	3	19
12q	4	19
13q	6	14
14q	3	12
15q	1	16
16p	4	19
16q	5	16
17p	19	19
17q	5	21
18p	5	15
18q	6	13
19p	5	20
19q	6	16
20p	2	17
20q	0	8
21p	0	7
21q	6	18
22q	4	15

Appendix C

RESULTS USING A LOGIT PRIOR FOR γ

Table C.1: Means and 95% posterior credible intervals from fitting the Binomial-BetaBinomial model with a logit(1) prior on γ.

Overall Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.858	0.855	0.618	0.967
π_1	0.231	0.233	0.192	0.271
π_2	0.665	0.666	0.323	0.924
γ	0.835	0.567	-2.111	4.255
Prob(Mode)	0.995	0.991		
Prob(Mode)	0.005	0.0086		

Mode 1 Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.862	0.862	0.653	0.967
π_1	0.228	0.228	0.192	0.267
π_2	0.667	0.670	0.347	0.924
γ	0.856	0.593	-1.866	4.255

Mode 2 Estimates

	Adaptive Quadrature	Estimates		
		Mean	2.5% Quantile	97.5% Quantile
η	0.079	0.112	0.046	0.148
π_1	0.861	0.852	0.845	0.861
π_2	0.240	0.239	0.214	0.266
γ	-3.22	-3.547	-4.820	-2.442

Appendix D

POSTERIOR PROBABILITY OF BELONGING TO THE TSG GROUP

Chromosome Arm	Observed Prob(LOH)	Binomial Quantile	Posterior Prob(TSG)
20q	0.000	0.125	0.007
21p	0.000	0.162	0.006
7p	0.056	0.059	0.072
15q	0.062	0.090	0.057
10q	0.118	0.217	0.033
20p	0.118	0.217	0.033
9q	0.125	0.255	0.031
8q	0.143	0.257	0.028
4q	0.158	0.337	0.026
6q	0.158	0.337	0.026
7q	0.158	0.337	0.026
12p	0.158	0.337	0.026
2q	0.167	0.382	0.026
11p	0.167	0.382	0.026
10p	0.167	0.458	0.029
1q	0.200	0.539	0.027
12q	0.211	0.552	0.024
16p	0.211	0.552	0.024
2p	0.235	0.654	0.027
17q	0.238	0.655	0.025
19p	0.250	0.700	0.027
14q	0.250	0.713	0.034
8p	0.263	0.744	0.029
3q	0.267	0.754	0.033
22q	0.267	0.754	0.033
6p	0.278	0.786	0.033
11q	0.278	0.786	0.033
16q	0.312	0.862	0.045
3p	0.333	0.894	0.054
4p	0.333	0.894	0.054
18p	0.333	0.894	0.054
21q	0.333	0.904	0.054
5p	0.375	0.947	0.085
19q	0.375	0.947	0.085
1p	0.412	0.976	0.141
13q	0.429	0.976	0.151
18q	0.462	0.985	0.210
9p	0.647	> 0.999	0.977
5q	0.800	> 0.999	> 0.999
17p	1.000	> 0.999	> 0.999

Appendix E

EXPECTED STEP DISTANCE AND ACCEPTANCE RATE FOR EACH CANDIDATE SCALING METHOD

Table E.1: Expected step distance (scaled by $1/\sqrt{d}$) for several methods of setting the NKC kernel bandwidth. Unimodal target distribution.

Number of Components	Dimension				
	2	4	10	20	50
2	0.452	0.313	0.105	0.025	1.08e-03
4	0.620	0.427	0.119	0.017	4.52e-04
10	0.804	0.607	0.194	0.023	1.66e-04
20	0.873	0.703	0.269	0.032	1.71e-04
50	0.918	0.781	0.380	0.063	< 1.00e-04
100	0.936	0.818	0.443	0.081	< 1.00e-04
500	0.951	0.833	0.522	0.168	< 1.00e-04

(a) Unit Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.482	0.339	0.099	0.014	1.28e-04
4	0.630	0.448	0.139	0.019	1.05e-04
10	0.797	0.607	0.218	0.034	1.00e-04
20	0.883	0.705	0.280	0.051	< 1.00e-04
50	0.980	0.823	0.380	0.080	2.19e-04
100	1.036	0.906	0.445	0.094	8.85e-04
500	1.126	1.028	0.590	0.169	1.16e-03

(b) Scaled Bandwidth

Table E.1: (continued)

Number of Components	Dimension				
	2	4	10	20	50
2	0.451	0.307	0.105	0.025	7.40e-04
4	0.584	0.383	0.107	0.017	3.32e-04
10	0.754	0.521	0.143	0.019	1.49e-04
20	0.844	0.621	0.182	0.020	1.60e-04
50	0.952	0.743	0.254	0.034	< 1.00e-04
100	1.013	0.826	0.301	0.040	< 1.00e-04
500	1.111	0.969	0.423	0.079	< 1.00e-04

(c) Scott's Optimal Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.473	0.217	6.35e-03	< 1.00e-04	< 1.00e-04
4	0.605	0.301	0.011	< 1.00e-04	< 1.00e-04
10	0.756	0.411	0.018	< 1.00e-04	< 1.00e-04
20	0.839	0.494	0.025	< 1.00e-04	< 1.00e-04
50	0.931	0.590	0.040	< 1.00e-04	< 1.00e-04
100	0.992	0.675	0.053	2.16e-04	< 1.00e-04
500	1.096	0.829	0.089	2.05e-04	< 1.00e-04

(d) Terrell's Maximal Smoothing Bandwidth

Table E.2: Expected acceptance rate for everal methods of setting the NKC kernel bandwidth. Unimodal target distribution.

Number of Components	Dimension				
	2	4	10	20	50
2	0.539	0.348	0.120	0.031	1.35e-03
4	0.579	0.377	0.109	0.018	5.67e-04
10	0.654	0.458	0.145	0.020	1.94e-04
20	0.678	0.512	0.190	0.024	2.14e-04
50	0.696	0.551	0.256	0.043	< 1.00e-04
100	0.707	0.567	0.295	0.055	< 1.00e-04
500	0.713	0.580	0.344	0.110	< 1.00e-04

(a) Unit Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.523	0.320	0.089	0.013	1.19e-04
4	0.575	0.372	0.109	0.015	< 1.00e-04
10	0.666	0.459	0.154	0.024	< 1.00e-04
20	0.715	0.525	0.193	0.034	< 1.00e-04
50	0.776	0.603	0.256	0.052	1.46e-04
100	0.819	0.656	0.300	0.062	5.92e-04
500	0.888	0.757	0.401	0.110	7.34e-04

(b) Scaled Bandwidth

Table E.2: (continued)

Number of Components	Dimension				
	2	4	10	20	50
2	0.540	0.350	0.122	0.030	8.83e-04
4	0.578	0.366	0.106	0.019	3.96e-04
10	0.661	0.430	0.118	0.018	1.67e-04
20	0.708	0.492	0.140	0.017	1.99e-04
50	0.772	0.570	0.184	0.025	< 1.00e-04
100	0.816	0.624	0.217	0.029	< 1.00e-04
500	0.887	0.734	0.306	0.055	< 1.00e-04

(c) Scott's Optimal Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.421	0.160	3.83e-03	< 1.00e-04	< 1.00e-04
4	0.493	0.209	6.69e-03	< 1.00e-04	< 1.00e-04
10	0.587	0.274	0.011	< 1.00e-04	< 1.00e-04
20	0.641	0.334	0.015	< 1.00e-04	< 1.00e-04
50	0.709	0.399	0.023	< 1.00e-04	< 1.00e-04
100	0.759	0.454	0.031	1.06e-04	< 1.00e-04
500	0.846	0.578	0.054	1.51e-04	< 1.00e-04

(d) Terrell's Maximal Smoothing Bandwidth

Table E.3: Expected step distance (scaled by $1/\sqrt{d}$) for several methods of setting kernel bandwidths. Bimodal target distribution.

Number of Components	Dimension				
	2	4	10	20	50
2	0.334	0.232	0.090	0.025	8.93e-04
4	1.269	0.697	0.159	0.020	5.76e-04
10	2.437	1.595	0.398	0.029	2.22e-04
20	2.970	2.111	0.667	0.070	1.85e-04
50	3.385	2.672	1.043	0.130	< 1.00e-04
100	3.527	2.775	1.345	0.206	1.91e-03
500	3.640	3.027	1.771	0.482	1.31e-04

(a) Unit Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.357	0.239	0.080	0.013	< 1.00e-04
4	1.302	0.768	0.193	0.024	< 1.00e-04
10	2.399	1.589	0.475	0.059	< 1.00e-04
20	2.965	2.069	0.718	0.120	1.52e-04
50	3.530	2.729	1.044	0.181	1.11e-04
100	3.829	2.987	1.315	0.252	2.01e-03
500	4.340	3.700	1.902	0.492	1.22e-03

(b) Scaled Bandwidth

Table E.3: (continued)

Number of Components	Dimension				
	2	4	10	20	50
2	0.611	0.417	0.147	0.057	2.73e-03
4	0.990	0.703	0.313	0.104	n/a
10	1.425	1.014	0.505	0.165	0.016
20	1.769	1.254	0.570	0.238	0.022
50	2.160	1.554	0.699	0.246	0.025
100	2.364	1.697	0.796	0.255	0.030
500	2.606	2.265	0.932	0.352	0.026

(c) Scott's Optimal Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.344	0.152	5.51e-03	< 1.00e-04	< 1.00e-04
4	1.318	0.612	0.019	< 1.00e-04	< 1.00e-04
10	2.346	1.210	0.047	5.13e-04	< 1.00e-04
20	2.883	1.547	0.075	< 1.00e-04	< 1.00e-04
50	3.429	2.049	0.110	< 1.00e-04	< 1.00e-04
100	3.730	2.271	0.162	5.84e-04	< 1.00e-04
500	4.252	3.014	0.286	< 1.00e-04	< 1.00e-04

(d) Terrell's Maximal Smoothing Bandwidth

Table E.4: Expected acceptance rate for everal methods of setting the NKC kernel bandwidth. Bimodal target distribution.

Number of Components	Dimension				
	2	4	10	20	50
2	0.420	0.280	0.109	0.031	1.14e-03
4	0.455	0.280	0.083	0.018	6.93e-04
10	0.561	0.372	0.101	0.011	2.93e-04
20	0.623	0.444	0.136	0.017	2.35e-04
50	0.676	0.523	0.205	0.026	< 1.00e-04
100	0.695	0.542	0.258	0.040	3.10e-04
500	0.712	0.579	0.334	0.087	< 1.00e-04

(a) Unit Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.395	0.246	0.076	0.012	< 1.00e-04
4	0.455	0.281	0.078	0.011	< 1.00e-04
10	0.561	0.371	0.111	0.015	< 1.00e-04
20	0.633	0.440	0.144	0.024	< 1.00e-04
50	0.716	0.541	0.205	0.035	< 1.00e-04
100	0.767	0.590	0.254	0.049	2.76e-04
500	0.859	0.714	0.359	0.089	2.72e-04

(b) Scaled Bandwidth

Table E.4: (continued)

Number of Components	Dimension				
	2	4	10	20	50
2	0.187	0.114	0.042	0.013	5.44e-04
4	0.230	0.153	0.068	0.021	n/a
10	0.281	0.201	0.098	0.032	3.14e-03
20	0.339	0.239	0.108	0.044	3.63e-03
50	0.404	0.293	0.128	0.048	3.99e-03
100	0.448	0.325	0.149	0.049	5.47e-03
500	0.503	0.433	0.173	0.062	5.38e-03

(c) Scott's Optimal Bandwidth

Number of Components	Dimension				
	2	4	10	20	50
2	0.314	0.117	3.48e-03	< 1.00e-04	< 1.00e-04
4	0.408	0.172	5.45e-03	< 1.00e-04	< 1.00e-04
10	0.524	0.256	0.010	< 1.00e-04	< 1.00e-04
20	0.599	0.315	0.014	< 1.00e-04	< 1.00e-04
50	0.686	0.394	0.022	< 1.00e-04	< 1.00e-04
100	0.739	0.441	0.031	1.41e-04	< 1.00e-04
500	0.837	0.576	0.053	< 1.00e-04	< 1.00e-04

(d) Terrell's Maximal Smoothing Bandwidth

Appendix F

R SOURCE CODE FOR mcgibbsit

The mcgibbsit code listed below was designed to be used with the R statistical package and the CODA library for R, both of which can be obtained free of charge from the web site http://www.r-project.org. The most recent version of the mcgibbsit, including any corrections or changes, can be obtained from the web site http://www.warnes.net/MCMC.

```
1  ### File version $Id: mcgibbsit.R,v 1.5 2000/09/30 02:10:49 warnes Exp $
2
3  ###
4  ###
5  ### Warnes's multi-chain extension to Raftery and Lewis's diagnostic
6  ###
7  ### Description:
8  ###
9  ###      'mcgibbsit' is a multi-chain extension of the Rafery and
10 ###      Lewis 'gibbsit' run length control diagnostic.  'mcgibbsit'
11 ###      is based on a criterion of accuracy of estimation of the
12 ###      quantile 'q'.  When applied to a pilot run it provides an
13 ###      estimate of the required total run length.  When applied to
14 ###      a full simulation it shows whether the MCMC sequence has
15 ###      been run for a sufficient number of iterations.
16 ###
17 ### Usage:
18 ###
19 ###      mcgibbsit(data, q=0.025, r=0.0125, s=0.95, converge.eps=0.001, correct.cor=T)
20 ###
21 ### Arguments:
22 ###
23 ###         data: an 'mcmc' object
24 ###
25 ###            q: quantile(s) to be estimated.
26 ###
27 ###            r: the desired margin of error of the estimate.
28 ###
29 ###            s: the probability of obtaining an estimate in the interval
30 ###
31 ### converge.eps: Precision required for estimate of time to convergence.
32 ###
33 ###  correct.cor: should the between-chain correlation correction (R) be
34 ###               computed and applied.  Set to false for independent
35 ###               mcmc chains.
```

```
36  ###
37  ### Details:
38  ###
39  ###       'mcgibbsit' computes the minimum run length 'Nmin', required
40  ###       burn in 'M', total run length 'N', run length inflation due
41  ###       to *auto*correllation 'I', and the run length inflation due
42  ###       to *between*chain* correlation 'R' for a set of exchangable
43  ###       MCMC simulations which need not be independent.
44  ###
45  ###       If the number of iterations in 'data' is too small, an error
46  ###       message is printed indicating the minimum length of pilot
47  ###       run.
48  ###
49  ###       Note that q, r, s, converge.eps, and correct.cor can be
50  ###       supplied as vectors.  This will cause 'mcgibbsit' to produce
51  ###       a list of results, with one element produced for each set of
52  ###       values.  IE, setting q=(0.025,0.975), r=(0.0125,0.005) will
53  ###       yield a list containg two mcgibbsit objects, one computed
54  ###       with parameters q=0.025, r=0.0125, and the other with
55  ###       q=0.975, r=0.005.
56  ###
57  ### Value:
58  ###
59  ###      An 'mcgibbsit' object with components 'call', 'params',
60  ###      'resmatrix', 'nchains' and 'len'.
61  ###
62  ###      call:  parameters used to call 'mcgibbsit'
63  ###
64  ###    params:  values of r, s, and q used
65  ###
66  ### resmatrix:  a matrix with 6 columns:
67  ###
68  ###           Nmin:  The minimium required sample size for a chain with no
69  ###                  correlation between consecutive samples.  Positive
70  ###                  autocorrelation will increase the required sample
71  ###                  size above this minimum value.
72  ###
73  ###              M:  The number of 'burn in' iterations to be discarded
74  ###                  (total over all chains).
75  ###
76  ###              N:  The number of iterations after burn in required to
77  ###                  estimate the quantile q to within an accuracy of
78  ###                  +/- r with probability p (total over all chains).
79  ###
80  ###          Total:  Overall number of iterations required (M + N).
81  ###
82  ###              I:  An estimate (the 'dependence factor') of the extent to
83  ###                  which auto-correlation inflates the required sample
84  ###                  size.  Values of 'I' larger than 5 indicate strong
85  ###                  autocorrelation which may be due to a poor choice of
86  ###                  starting value, high posterior correlations or
87  ###                  'stickiness' of the MCMC algorithm.
88  ###
89  ###              R:  An estimate of the extent to which between-chain
90  ###                  correlation inflates the required sample size.  Large
91  ###                  values of 'R' indicate that there is significant
92  ###                  correlation between the chains and may be indicative
93  ###                  of a lack of convergenge or a poor multi-chain
94  ###                   algorithm.
95  ###
96  ###
97  ###    nchains:  the number of MCMC chains in the data
98  ###
```

```
99  ###       len:  the length of each chain
100 ###
101 ###
102 ### Note:
103 ###
104 ###       'mcgibbsit' is based on the the R function 'raftery.diag'
105 ###       which is part of the 'CODA' library.  'raftery.diag', in
106 ###       turn, is based on the FORTRAN program 'gibbsit' written by
107 ###       Steven Lewis which is available from the Statlib archive.
108 ###
109 ### References:
110 ###
111 ###       Warnes, G.W. (2000).  Multi-Chain and Parallel Algorithms for
112 ###       Markov Chain Monte Carlo. Dissertation, Department of
113 ###       Biostatistics, University of Washington,
114 ###
115 ###       Raftery, A.E. and Lewis, S.M. (1992).  One long run with
116 ###       diagnostics: Implementation strategies for Markov chain Monte
117 ###       Carlo. Statistical Science, 7, 493-497.
118 ###
119 ###       Raftery, A.E. and Lewis, S.M. (1995).  The number of iterations,
120 ###       convergence diagnostics and generic Metropolis algorithms.  In
121 ###       Practical Markov Chain Monte Carlo (W.R. Gilks, D.J. Spiegelhalter
122 ###       and S. Richardson, eds.). London, U.K.: Chapman and Hall.
123 ###
124 ###
125
126 # Revision History
127 # $Log: mcgibbsit.R,v $
128 # Revision 1.5  2000/09/30 02:10:49  warnes
129 # Better late than never.
130 #
131 # Revision 1.4  2000/07/20 17:30:04  warnes
132 #
133 # Cleaned up the code.  Added some additional documentation and comments.
134 #
135 # Revision 1.3  2000/07/20 07:29:21  warnes
136 #
137 # 1) Changed computation of the thining factor (kthin) to use common 1st
138 #    order and 2nd order models across chains.
139 #
140 # 2) Corrected correlation inflation factor
141 #
142 # 3) Previous versions gave N = (burn in + estimation length) as N.  In
143 #    fact, N = estimation length.  Added a new column to output table for
144 #    total (N+M) and corrected the output of the $N$ column.
145 #
146 # 4) Added a note to output of print.mcgibbsit that the number of
147 #    iterations reported are overall rather than per-chain.
148 #
149 # Revision 1.2  2000/07/20 05:38:44  warnes
150 #
151 # Made change to use MLE for rho (  mean of cov / mean of var )
152 #
153 # Revision 1.1.1.1  2000/07/18 13:53:27  warnes
154 #
155 # Initial Checkin.  Moved old dissertation files to CVS directory "dissertation-orig".
156 #
157 #
158 # Revision 1.2  2000/07/15 21:25:37  warnes
159 #
160 # 1) Corrected burn in ("nburn") to be inflated by the number of chains.
161 #
```

```
# 2) Instead of turning any "NA" obtained from trying to take the
#    correlation of two discretized sequences where one never changes, use
#    the mean of the correlations which we can successfully estimate.
#    The first method was over-conservative.
#
# 3) Corrected the computation of the run-length inflation due to serial
#    correlation ("I"). It was including the inflation due to
#    between-chain correlation. (The actual computed run-length was
#    correct.)
#
# 4) Corrected the computation of the inflation due to between-chain correlation.
#    (The actual run length was correct.)
#


"mcgibbsit" <- function (data, q = 0.025, r = 0.0125, s = 0.95,
                         converge.eps = 0.001, correct.cor=T )
{
  ## if asked for more than one q,r,s,..., combination, recurse and return
  ## a list of results

  parms - cbind( q, r, s, converge.eps, correct.cor ) # so one can use
                                                      # different r/s values
                                                      # for each q

  if(dim(parms)[1] > 1 )
    {

      retval - list();
      for(i in 1:length(q) )
        {
        retval[[i]] - mcgibbsit( data, q=parms[i,"q"], r=parms[i,"r"],
                                 s=parms[1,"s"],
                                 converge.eps=parms[i,"converge.eps"],
                                 correct.cor=parms[i,"correct.cor"] )
      }
      return(retval)
    }

  if (is.mcmc.list(data))
    {
      nchains <- length(data)

      ## check that all of the chains are conformant #
      for( ch in 1:nchains)
        if( (start(data[[1]]) != start(data[[ch]] )) ||
           (end  (data[[1]]) != end  (data[[ch]] )) ||
           (thin (data[[1]]) != thin (data[[ch]] )) ||
           (nvar (data[[1]]) != nvar (data[[ch]] )) )
          stop(paste("All chains in mcmc.list must have same 'start',",
                     "'end', 'thin', and number of variables"));

      if(is.matrix(data[[1]]))
        combined <- mcmc(do.call("rbind",data))
      else
        combined <- mcmc(as.matrix(unlist(data)))
    }
  else
    {
      data <- mcmc.list(mcmc(as.matrix(data)))
      nchains <- 1
      combined <- mcmc(as.matrix(data))
    }
```

```
225
226
227     resmatrix <- matrix(nrow = nvar(combined),
228                         ncol = 6,
229                         dimnames = list( varnames(data, allow.null = TRUE),
230                                          c("M", "N", "Total", "Nmin",
231                                            "I", "R" )) )
232
233     # minimum number of iterations
234     phi <- qnorm(0.5 * (1 + s))
235     nmin <- as.integer(ceiling((q * (1 - q) * phi^2)/r^2))
236
237     if (nmin > niter(combined))
238       resmatrix <- c("Error", nmin)
239     else
240       for (i in 1:nvar(combined))
241         {
242           dichot <- list()
243
244           if (is.matrix(data[[i]]))
245             {
246               quant <- quantile(combined[, i, drop = TRUE], probs = q)
247
248               for(ch in 1:nchains)
249                 {
250                   dichot[[ch]] <- mcmc(data[[ch]][, i, drop = TRUE] <= quant,
251                                        start = start(data[[ch]]),
252                                        end   = end(data[[ch]]),
253                                        thin  = thin(data[[ch]]))
254                 }
255             }
256           else
257             {
258               quant <- quantile(combined, probs = q)
259
260               for(ch in 1:nchains)
261                 {
262                   dichot[[ch]] <- mcmc(data[[ch]] <= quant,
263                                        start = start(data[[ch]]),
264                                        end = end(data[[ch]]),
265                                        thin = thin(data[[ch]]))
266                 }
267             }
268           kthin <- 0
269           bic <- 1
270
271           while (bic >= 0)
272             {
273               kthin <- kthin + thin(data[[1]])
274
275               to.table _ function(dichot, kthin)
276                 {
277                   testres <- as.vector(window.mcmc(dichot, thin = kthin))
278                   newdim <- length(testres)
279                   testtran <- table(testres[1:(newdim - 2)],
280                                     testres[2:(newdim - 1)],
281                                     testres[3:newdim])
282
283                   ## handle the case where one or more of the transition never
284                   ## happens, so that the testtran array has two few dimensions
285                   if( any(dim(testtran!=2)) )
286                     {
287                       tmp _ array( 0, dim=c(2,2,2),
```

```
                                dimnames=list( c("FALSE","TRUE"),
                                               c("FALSE","TRUE"),
                                               c("FALSE","TRUE") ) )
                    for(t1 in dimnames(testtran)[[1]] )
                      for(t2 in dimnames(testtran)[[2]])
                        for(t3 in dimnames(testtran)[[3]] )
                          tmp[t1,t2,t3] <- testtran[t1,t2,t3]

                    testtran <- tmp
                  }

                testtran <- array(as.double(testtran), dim = dim(testtran))
                return(testtran)
              }

            tmp <- sapply( dichot, to.table, kthin=kthin, simplify=F )

            ## add all of the transition matrixes together
            testtran <- tmp[[1]]
            if(nchains>1)
              for(ch in 2:nchains)
                testtran <- testtran + tmp[[ch]]

            ## compute the likelihoood
            g2 <- 0
            for (i1 in 1:2)
              {
                for (i2 in 1:2)
                  {
                    for (i3 in 1:2)
                      {
                        if (testtran[i1, i2, i3] != 0) {
                          fitted <- (sum(testtran[i1, i2, 1:2]) *
                                     sum(testtran[1:2, i2, i3])) /
                                       (sum(testtran[1:2, i2, 1:2]))
                          g2 <- g2 + testtran[i1, i2, i3] *
                                     log(testtran[i1, i2, i3]/fitted) * 2
                        }
                      }
                  }
              }

            ## compute bic
            bic <- g2 - log( sum(testtran) - 2 ) * 2

          }

        ## estimate the parameters of the first-order markov chain
        alpha <- sum(testtran[1, 2, 1:2]) / (sum(testtran[1, 1, 1:2]) +
                                             sum(testtran[1, 2, 1:2]))
        beta  <-sum(testtran[2, 1, 1:2]) / (sum(testtran[2, 1, 1:2]) +
                                             sum(testtran[2, 2, 1:2]))

        ## compute burn in
        tempburn <- log((converge.eps * (alpha + beta)) /
                        max(alpha, beta))/(log(abs(1 - alpha - beta)))

        nburn <- as.integer(ceiling(tempburn) * kthin)

        ## compute iterations after burn in
        tempprec <- ((2 - alpha - beta) * alpha * beta * phi^2)/
                    (((alpha + beta)^3) * r^2)
        nkeep  <- ceiling(tempprec * kthin)
```

```
351
352         ## compute the correlation
353         if(nchains>1 && correct.cor)
354           {
355             dat <- do.call("cbind", dichot)
356             varmat <- var(dat)
357             denom <- mean(diag(varmat))  # overall variance
358             diag(varmat) <- NA
359             numer <- mean(c(varmat), na.rm=T) # overall covariance
360             rho <- numer / denom
361           }
362         else
363           rho <- 1.0
364
365         ## inflation factors
366         iratio <- (nburn + nkeep)/nmin
367         R      <- ( 1 + rho * (nchains - 1) )
368
369         resmatrix[i, 1] <- M<- ceiling( nburn * nchains )  # M
370         resmatrix[i, 2] <- N<- ceiling( nkeep * R       )  # N
371         resmatrix[i, 3] <- M + N                           # Total
372         resmatrix[i, 4] <- nmin                            # nmin
373         resmatrix[i, 5] <- signif(iratio,  digits = 3)     # I
374         if(nchains > 1 && correct.cor )
375           resmatrix[i, 6] <- signif(R     ,  digits = 3)   # R
376         else
377           resmatrix[i, 6] <- NA                            # R
378
379       }
380   y <- list(params = c(r = r, s = s, q = q),
381             resmatrix = resmatrix, call=match.call(),
382             nchains = nchains, len=(end(data[[1]]) - start(data[[1]]) + 1) )
383   class(y) <- "mcgibbsit"
384   return(y)
385 }
386
387 "print.mcgibbsit" <-
388   function (x, digits = 3, ...)
389 {
390   cat("                  Multi-Chain Gibbsit \n")
391   cat("                  ------------------\n")
392   cat("\n");
393
394   cat("Call              = "); print(x$call)
395   cat("\n");
396
397   cat("Number of Chains =", x$nchains, "\n" )
398   cat("Per-Chain Length =", x$len, "\n" )
399   cat("Total Length     =", x$nchains * x$len, "\n")
400   cat("\n");
401
402   cat("Quantile (q)     =", x$params["q"] , "\n")
403   cat("Accuracy (r)     = +/-", x$params["r"], "\n")
404   cat("Probability (s)  =", x$params["s"], "\n")
405   cat("\n")
406
407   if (x$resmatrix[1] == "Error")
408     cat("\nYou need a sample size of at least", x$resmatrix[2],
409         "with these values of q, r and s\n")
410   else {
411     out <- x$resmatrix
412     for (i in ncol(out)) out[, i] <- format(out[, i], digits = digits)
413     out <- rbind(matrix(c("Burn-in ", "Estimation", "Total", "Lower bound ",
```

```
414                              "Auto-Corr.", "Between-Chain",
415                              "(M)",        "(N)",        "(M+N)", "(Nmin)",
416                              "factor (I)", "Corr. factor (R)"),
417                            byrow = TRUE, nrow = 2), out)
418      if (!is.null(rownames(x$resmatrix)))
419        out <- cbind(c("", "", rownames(x$resmatrix)), out)
420      dimnames(out) <- list(rep("", nrow(out)), rep("", ncol(out)))
421
422      print.default(out, quote = FALSE, ...)
423      cat("\n")
424
425      cat("NOTE: The values for M, N, and Total are combined numbers",
426          " of iterations \n")
427      cat("      based on using", x$nchains, "chains.\n");
428      cat("\n")
429
430    }
431    invisible(x)
432  }
```

Appendix G

DERIVATION OF THE PAIRWISE CORRELATION CORRECTION

Let $X_1, .., X_n$ be an exchangeable sample of n observations which share a common variance σ^2 and a common pairwise correlation ρ. The variance of the mean of the n observations is then

$$\begin{aligned}
\mathrm{Var}(\overline{X}_n) &= \mathrm{Var}(\frac{1}{n}\sum_{i=1}^{n} X_i) \\
&= \frac{1}{n^2}\mathrm{Var}(\sum_{i=1}^{n} X_i) \\
&= \frac{1}{n^2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{Cov}(X_i, X_j)\right) \\
&= \frac{1}{n^2}\left(\sum_{i=1}^{n}\mathrm{Var}(X_i) + \sum_{i=1}^{n}\sum_{j\neq i}^{n}\mathrm{Cov}(X_i, X_j)\right) \\
&= \frac{1}{n^2}\left(n\sigma^2 + n(n-1)\mathrm{Cov}(X_i, X_j)\right) \\
&= \frac{1}{n}\left(\sigma^2 + (n-1)\,\sigma^2\rho\right) \\
&= \frac{\sigma^2}{n}\left(1 + \rho\,(n-1)\right)
\end{aligned}$$

This result is a well known in survey sampling, eg Scheaffer et al. (1990).

Appendix H

INSTALLING HYDRA

The HYDRA Java package is available in two forms, as a Jar file (`Hydra.jar`) containing only the compiled classes and as a gzipped tar file (`Hydra.current.tar.gz`) containing the full source code as well as the compiled classes. Both files are available from the HYDRA web page located at `http://www.warnes.net/Hydra`.

H.1 Installing the jar File

Download `Hydra.jar` and append the full path to the jar file to the CLASSPATH. For example, if `Hydra.jar` has been placed in the directory `/home/user/jars/`, the proper command for setting the CLASSPATH using `sh` compatible shells is

```
> CLASSPATH=$CLASSPATH:/home/user/jars/Hydra.jar
> export CLASSPATH
```

and using `csh` compatible shells

```
> setenv CLASSPATH $CLASSPATH:/home/user/jars/Hydra.jar
```

H.2 Installing the Full Source

Download `Hydra.current.tar.gz`. It can then be unpacked using GNU tar via

```
> tar -xvzf Hydra.current.tar.gz
```

which will unpack a directory tree with root "Hydra". The Java files and source code are contained in directories under `Hydra/org/omegahat`

The location of this directory then needs to be added to the Java class path. If the directory tree was unpacked in `/home/user/jsrc/` this can be accomplished using `sh` compatible shells by

```
> CLASSPATH=$CLASSPATH:/home/user/jsrc/Hydra
> export CLASSPATH
```

and using `csh` compatible shells

```
> setenv CLASSPATH $CLASSPATH:/home/user/jsrc/Hydra
```

H.3 Other Packages

Two additional Java packages may be required to use particular features of the HYDRA library, Visual Numerics' `JNL` and `Omegahat`. In addition, the `CODA` package, in conjunction with either `R` or `SPLUS` statistical packages, provides a useful suite of tools for evaluating and making inference using MCMC output.

- Visual Numerics' JNL library is required for several of the HYDRA classes, in particular those used in the Binomial-BetaBinomial example given below. `JNL` is available free of charge from the Visual Numerics web site:
  `http://www.vni.com/products/wpd/jnl/`.

- The HYDRA MCMC classes were designed to be compatible with the `Omegahat` statistical programming system. `Omegahat` provides an interactive environment for statistical programming and analysis and is under active development by the `Omegahat` project, `http://www.omegahat.org`.

- The statistical package R is a free re-implementation of the S language and may be obtained free of charge from http://www.r-project.org.

- The CODA package of MCMC diagnostics and other tools for Splus can be obtained from

  http://www.mrc-bsu.cam.ac.uk/bugs/classic/coda04/readme.shtml.

  A version for R can be obtained from http://www-fis.iarc.fr/coda.

Appendix I

PREDEFINED PROPOSAL DISTRIBUTIONS

HYDRA provides a selection of predefined proposal methods for the Metropolis, Metropolis-Hastings, and Hastings-Coupled techniques.

Metropolis Samplers

Class Name	Description
`NormalMetropolisProposal`	normal random-walk proposal
`NormalMetropolisComponentProposal`	variable-at-a-time random-walk proposal

Metropolis-Hastings Samplers

Class Name	Description
`NormalProposal`	(fixed) normal proposal
`NormalMetropolisProposal`	normal random-walk proposal
`NormalMetropolisComponentProposal`	variable-at-a-time random-walk proposal
`MixtureProposal`	finite mixture proposal using specified components

Hastings-Coupled (Multi-Chain) Samplers

Class Name	Description
`IndependentHastingsCoupledProposal`	Wrapper for independent Metropolis-Hastings Samplers
`AdaptiveNormalMetropolisProposal`	(Variance) Adaptive Normal Metropolis Proposal
`AdaptiveNormalProposal`	(Mean, Variance) Adaptive Normal Proposal
`NormalKernelProposal`	Normal Kernel Coupler
`AdaptiveNormalKernelProposal`	(Variance) Adaptive Normal Kernel Coupler
`LocallyAdaptiveNormalKernelProposal`	Locally-Adaptive Normal Kernel Coupler
`KernelDirectionSampler`	Kernel Direction Sampler

Appendix J

PREDEFINED LISTENERS

A variety of predefined listeners are available. These allow monitoring various features of the MCMC simulation and give several storage methods.

Class Name	Description
AcceptanceWriter	Stores the cumulative acceptance rate to a file
CovarianceWriter	Stores the cumulative covariance matrix to a file
DistanceListener	Computes the observed and expected acceptance rate, step distance, step distance conditional on acceptance
DistanceWriter	Stores the observed and expected acceptance rate, step distance, step distance conditional on acceptance to a file
HistogramWriter	Stores a cumulative histogram of the current states a file
ListenerGzipWriter	Stores the current state to GZIP compressed file
ListenerPrinter	Prints the event passed to notify()
ListenerWriter	Stores the event passed to notify() to a file
MeanWriter	Stores the cumulative mean vector to a file
PosteriorProbWriter	Stores the (unnormalized) posterior probability of the current state to a file
QuantileWriter	Stores the cumulative quantiles to a file
StepListenerPrinter	Prints `MCMCStepEvents`
StrippedListenerGzipWriter	Stores the current state to a GZIP compressed file
StrippedListenerWriter	Stores the current state to a file
ThinningProxyListener	A proxy for other listeners that thins the reported events by a specified factor, eg 1 out of every 100

Appendix K

OUTPUT FROM BINOMIAL_BETABINOMIAL_EXAMPLE.JAVA

```
> java org.omegahat.Simulation.MCMC.Examples.Binomial_BetaBinomial_Example

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.71 0.49 ]
Last      Prob   = -359.046964566765
Proposed State   = ContainerState: [ 0.9 0.4751068415506061 0.71 0.49 ]
Proposed Prob    = -423.26454869568283
Current  State   = ContainerState: [ 0.9 0.23 0.71 0.49 ]
Forward Prob     = -0.0369493853787235
Reverse Prob     = -0.0369493853787235
Acceptance Prob = -64.21758412891785
Acceptance Val   = 0.658405257229882
Accepted?        = false
Acceptance Rate = 0.0

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.71 0.49 ]
Last      Prob   = -359.046964566765
Proposed State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.49 ]
Proposed Prob    = -360.0165066537818
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.49 ]
Forward Prob     = -0.06327351354448152
Reverse Prob     = -0.06327351354448152
Acceptance Prob = -0.969542087016805
Acceptance Val   = 0.31056162077494043
Accepted?        = true
Acceptance Rate = 0.5

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.49 ]
Last      Prob   = -360.0165066537818
Proposed State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Proposed Prob    = -360.1951841070053
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Forward Prob     = 0.6154440530408976
Reverse Prob     = 0.6154440530408976
Acceptance Prob = -0.17867745322348583
Acceptance Val   = 0.13272539253007873
Accepted?        = true
Acceptance Rate = 0.6666666666666666

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Last      Prob   = -360.1951841070053
Proposed State   = ContainerState: [ 1.0245145624463277 0.23 0.45610096830883196
0.4225189574152196 ]
Proposed Prob    = -Infinity
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Forward Prob     = 0.23049155040119496
```

```
Reverse Prob     = 0.23049155040119496
Acceptance Prob = -Infinity
Acceptance Val   = 0.335025050367706
Accepted?        = false
Acceptance Rate = 0.5

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Last      Prob   = -360.1951841070053
Proposed State   = ContainerState: [ 0.9 0.1856011046441249 0.45610096830883196
0.4225189574152196 ]
Proposed Prob    = -363.1065248979661
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Forward Prob     = 0.3116872397632934
Reverse Prob     = 0.3116872397632934
Acceptance Prob = -2.9113407909608213
Acceptance Val   = 0.14726731467399157
Accepted?        = false
Acceptance Rate = 0.4

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Last      Prob   = -360.1951841070053
Proposed State   = ContainerState: [ 0.9 0.23 0.1825256980628881 0.4225189574152196 ]
Proposed Prob    = -364.9924350935826
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Forward Prob     = -0.1255457772139429
Reverse Prob     = -0.1255457772139429
Acceptance Prob = -4.797250986577353
Acceptance Val   = 0.4310649477090058
Accepted?        = false
Acceptance Rate = 0.3333333333333333

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.4225189574152196 ]
Last      Prob   = -360.1951841070053
Proposed State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.5879703533000055 ]
Proposed Prob    = -359.85442835072524
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.5879703533000055 ]
Forward Prob     = 0.3415983954458079
Reverse Prob     = 0.3415983954458079
Acceptance Prob = 0.0
Acceptance Val   = 0.36058319097411967
Accepted?        = true
Acceptance Rate = 0.42857142857142855

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.5879703533000055 ]
Last      Prob   = -359.85442835072524
Proposed State   = ContainerState: [ 0.6696587069504394 0.23 0.45610096830883196
0.5879703533000055 ]
Proposed Prob    = -361.2666064185844
Current  State   = ContainerState: [ 0.9 0.23 0.45610096830883196 0.5879703533000055 ]
Forward Prob     = 0.00517213125315924
Reverse Prob     = 0.00517213125315924
Acceptance Prob = -1.412178067859145
Acceptance Val   = 0.9268299028867995
Accepted?        = false
Acceptance Rate = 0.375

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.23 0.45610096830883196 0.5879703533000055 ]
Last      Prob   = -359.85442835072524
```

```
Proposed State   = ContainerState: [ 0.9 0.2099774552506214 0.45610096830883196
0.5879703533000055 ]
Proposed Prob    = −360.42346255905113
Current  State   = ContainerState: [ 0.9 0.2099774552506214 0.45610096830883196
0.5879703533000055 ]
Forward Prob     = 0.32110939780366615
Reverse Prob     = 0.32110939780366615
Acceptance Prob = −0.5690342083258884
Acceptance Val   = 0.3311132575995816
Accepted?        = true
Acceptance Rate = 0.4444444444444444

Chain Step Event (with details)
Last             = ContainerState: [ 0.9 0.2099774552506214 0.45610096830883196
0.5879703533000055 ]
Last      Prob   = −360.42346255905113
Proposed State   = ContainerState: [ 0.9 0.2099774552506214 0.15833697164158184
0.5879703533000055 ]
Proposed Prob    = −365.44521171685466
Current  State   = ContainerState: [ 0.9 0.2099774552506214 0.45610096830883196
0.5879703533000055 ]
Forward Prob     = −0.2084655958574132
Reverse Prob     = −0.2084655958574132
Acceptance Prob = −5.0217491578035265
Acceptance Val   = 0.7418864833851747
Accepted?        = false
Acceptance Rate = 0.4
```

VITA

Gregory R. Warnes is an Eagle Scout and served for two years as a full-time missionary for the Church of Jesus Christ of Latter Day Saints. He earned an Associate of Science degree in Computer Science from Weber State University in 1992, a Bachelor of Science degree in Statistics with an emphasis in Computer Science from Brigham Young University in 1995, and Master of Science degree from the department of Biostatistics at the University of Washington in 1998.

Mr. Warnes' research interests lie at the intersection of statistics, bio-medical research, and computer science. His research activities include the development of software tools for parallel computing using clusters of commodity workstations, the development of Monte Carlo techniques for simulating from multi-modal and high dimensional statistical distributions, and the analysis of data from DNA micro-arrays.

Previous works by Mr. Warnes:

YANEZ, N. D., WARNES, G. R., & KRONMAL, R.A. (in press). A univariate measurement error model for longitudinal change. *Communications in Statistics*.

LYON, J. L., FELLINGHAM, G., TOLLEY, H. D., HARRIS, T., HILTON, G., & WARNES, G. (1995). Mortality differences among adult men in Utah 1975-79 associated with differences in tobacco and alcolhol usage. Unpublished.