

Metropolis-Hastings Markov Chain Monte Carlo

Matthew Strand
Chapman University
Orange, CA 92866
Email: stran104@chapman.edu
May 2009

Abstract—Markov Chain Monte Carlo is a class of random walk algorithms commonly used for carrying out complicated numerical integration. This is useful for computing expectations of complicated high-dimension probability distributions. In this paper we provide an overview of Metropolis-Hastings MCMC starting with a brief introduction of its usefulness in Bayesian statistics. Next we describe the basics of Monte Carlo integration and introduce the Metropolis-Hastings algorithm for building Markov chains. Descriptions of many of the important decisions for a successful simulation are provided; including error reporting via Monte Carlo standard error. To reinforce these concepts we introduce a software implementation that allows for graphical study of the Metropolis-Hastings sampler. Finally we discuss situations in which the algorithm can be applied and when other methods should be considered instead.

I. INTRODUCTION

Markov Chain Monte Carlo provides a method of computing high-dimension integrals using Markov chains. MCMC relies on the premise that when one is unable to draw independent samples from a random process, drawing dependent samples is almost as good. [1]. This is achieved using a Markov chain to draw a sample based on the last sample drawn and the prior knowledge of the data. This is particularly useful for the computation of statistical analysis and modeling of real world data that is generally non-uniform and difficult to sample from directly.

All of the algorithms in this class consist of two basic steps:

- building a Markov chain
- performing Monte Carlo integration on the Markov chain.

This concept was first formalized by Nicholas Metropolis et al. in 1952 for the specific case of the Boltzmann distribution. The Metropolis method works on symmetric distributions and was later generalized for arbitrary distributions regardless of symmetry by W.K. Hastings in 1970. The resulting algorithm is referred to as Metropolis-Hastings Markov Chain Monte Carlo.

There are a variety of MCMC methods, each with its own merits. The Gibbs sampler is a particularly popular method. It is easier to implement than Metropolis-Hastings, may run faster, and requires no *tuning* but requires that the conditional distribution of each variable be known. Since this is not always the case, the Metropolis-Hastings algorithm is often used instead. For this reason we will focus our attention to this variety of Markov Chain Monte Carlo.

MCMC methods are frequently used in computational physics, computational biology, econometrics, and a variety of

other fields. It is most frequently used in Bayesian statistics to approximate the expectation of a complicated probability distribution.

II. A BAYESIAN MOTIVATION

In Bayesian inference we need to construct a probability model based on observed data. In this framework the model parameters themselves are also considered to be random quantities. To perform formal inference we must set up a joint probability model to take all random quantities into account. If we let D denote the data and θ denote the model parameters and missing data then we need to construct $P(D, \theta)$.

This joint distribution is formed by taking the product of two functions. $f(\theta)$: The prior distribution representing the underlying probability distribution from which the data is believed to arise. The second quantity is the likelihood function for the data, $L(\theta|D)$ which incorporates the effect of the data on the model. Since

$$L(\theta|D) = P(D|\theta)$$

We write the likelihood function as $P(D|\theta)$.

Using the prior and the likelihood we can form a *full probability model* as

$$P(D, \theta) = P(D|\theta)P(\theta)$$

It is possible to obtain the posterior distribution by applying Bayes Rule to $P(D|\theta)$

$$P(\theta|D) = \frac{P(\theta)P(D|\theta)}{P(D)}$$

Where $P(D) = \int P(D, \theta)d\theta = \int P(D|\theta)P(\theta)d\theta$. So we obtain

$$P(\theta|D) = \frac{P(\theta)P(D|\theta)}{\int P(\theta)P(D|\theta)d\theta}$$

This describes the probability that the parameters of interest assume particular values given the observed data. Now the goal becomes finding the expectation of a function of θ , $E(f(\theta))$.

Using the posterior distribution, we can write this as

$$E(f(\theta)|D) = \frac{\int f(\theta)P(\theta)P(D|\theta)d\theta}{\int P(\theta)P(D|\theta)d\theta}$$

However, the exact computation of this expectation is often impossible for real world data. Among other methods, Markov Chain Monte Carlo offers a way of numerically approximating this quantity.

III. GENERALIZING THE PROBLEM

Bayesian inference is the most popular use of expectations of high-dimensional distributions, but the problem can be generalized for use in inter-disciplinary fields. The focus of this paper is on the computation of expectation of a function of interest, f , of random variables, X , that arise from the distribution $\pi(\cdot)$.

$$E(f(X)) = \frac{\int f(x)\pi(x)dx}{\int \pi(x)dx}$$

[2]

The denominator of this formula is the normalization constant which is required to ensure that $E(f(x))$ is a probability density function. Recall that a probability density function has the property that the the probability of the entire sample space is 1. Often this value is not actually known and is too difficult to compute. Often we can only calculate this expectation up to a constant of proportionality.

$$E(f(x)) \propto \int f(x)\pi(x)dx$$

The amazing thing is that with Markov Chain Monte Carlo it is not necessary to compute the normalizing factor for the computation of the expectation.

IV. MONTE CARLO INTEGRATION

As previously stated, Markov Chain Monte Carlo is based on Monte Carlo integration, a method of numerically approximating integrals. Let X denote a vector of k random variables with distribution $\pi(\cdot)$. If we can accurately sample from $\pi(\cdot)$ then we can use Monte Carlo integration to approximate the expectation as follows:

$$E(f(x)) \approx \frac{1}{n} \sum_{t=1}^n f(X_t)$$

This formula relies on the law of large numbers to state that the *maximum likelihood estimator* of the sample mean of $f(X)$ can be approximated by the arithmetic mean of a sufficiently large number of samples of $f(X)$.

That is, if we can accurately draw $\{X_1, X_2, \dots, X_n\}$ then we could apply Monte Carlo integration to find the desired expectation.

Since $\pi(\cdot)$ may be very complicated, drawing these samples may be non-trivial. To address this problem we turn to the concept of *Markov Chains*.

V. MARKOV CHAIN MONTE CARLO

A Markov Chain is a sequence of random variables $\{X_0, X_1, \dots, X_t\}$ such that X_t depends only upon X_{t-1} . Maintaining our notation from above, X is a collection of k random variables

If we can build a Markov Chain having $\pi(\cdot)$ as its stationary distribution then we can use this Markov Chain to draw the necessary samples to perform Monte Carlo integration. Hence the name, Markov Chain Monte Carlo.

The chain must be initialized with some initial X_0 . We must run this chain for a sufficiently long time such that this initial value is “forgotten” and eventually the chain will begin to draw dependent samples from its stationary distribution, $\pi(\cdot)$. This process is referred to as *burn-in*. The values obtained after the burn-in may then be used to perform the approximation.

For a Markov Chain of length n with burn-in m , we can write the *ergodic average* [2] of f as

$$\bar{f} = \frac{1}{n-m} \sum_{t=m+1}^n f(X_t)$$

Which will converge to our desired expectation, $\hat{\mu}$ by the ergodic theorem [2].

The values of m and n must both be chosen to be sufficiently large by analyzing the output of the Markov Chain.

Notice that computing the expectation using Monte Carlo integration does not require the difficult computation of the normalization factor.

Now we must develop a method of constructing a Markov Chain with $\pi(\cdot)$ as its stationary distribution. One method of doing this is to use the Metropolis-Hastings Algorithm.

VI. THE METROPOLIS-HASTINGS ALGORITHM

The algorithm first introduced by Metropolis, et al. and later generalized by Hastings allows for construction of these Markov Chains based upon a *proposal distribution*, $q(\cdot|X_t)$. It is a *rejection sampling* algorithm that only requires being able to evaluate the density of the target distribution $\pi(\cdot)$. It draws a sample from the proposal and accepts it with probability α . If the sample is rejected the chain remains unchanged at state $t + 1$. The proposal distribution should resemble $\pi(\cdot)$ but may depend on the current sample X_t . Choosing a good proposal distribution will be covered later in this paper, but a multivariate normal distribution usually provides a good approximation.

The algorithm draws a point Y from the proposal distribution given the current state of the chain. Y is accepted as the next point in the chain with probability α

$$\alpha(X_t, Y) = \min \left(1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)} \right)$$

If it is rejected then current point becomes the next point, $X_{t+1} = X_t$. This process is repeated for $t = 1 \dots n$. Note that normalizing constant need not be known because it drops out of the ratio $\pi(Y)/\pi(X)$ [3].

VII. PROPOSAL DISTRIBUTIONS AND MIXING

In theory, when using Metropolis-Hastings it does not matter what proposal distribution you pick, the Markov chain will always converge to the stationary distribution $\pi(\cdot)$. For a detailed explanation of why this is the case the reader is referred to *Gilks*, 7 as cited in this paper.

However, picking a good proposal distribution is essential to obtaining reliable results with a chain of practical running time. A poor choice of a proposal distribution may result in slow convergence and slow mixing. The *mixing* of the

chain, refers to how the chain moves about the support set of $\pi(\cdot)$. Ideally, the chain will possess *rapid mixing*, drawing samples rapidly moving around the support set. Under this condition, the chain is more likely to continuously draw samples throughout the support in the correct proportions.

On the other hand, chains with slow mixing will draw many samples from one region of the support and then slowly move to another region. Thus, chains with slow mixing must be run for much longer in order to counteract the bias that is introduced by focusing on select portions of the support.

Another important consideration is the computational efficiency associated with the proposal distribution. To prevent bottle necks in computation, a distribution should be chosen that can be quickly sampled and evaluated.

Large sample theory states that the posterior distribution of the parameters approaches a multivariate normal distribution [4]. This is particularly convenient because it is easy to quickly sample from such a distribution. When using a single or multivariate normal proposal, the means and covariance matrix should be tuned to obtain good mixing for the problem at hand. It is common to use either the means directly from X_t or a function thereof as the means in the proposal distribution for X_{t+1} . The purpose of this is to allow the proposal distribution to draw its next sample near the last accepted sample with some specified variance.

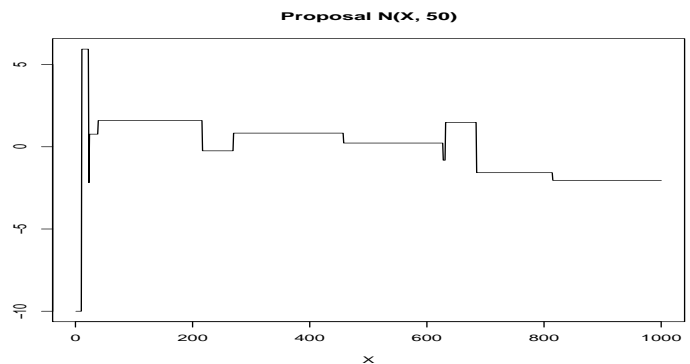
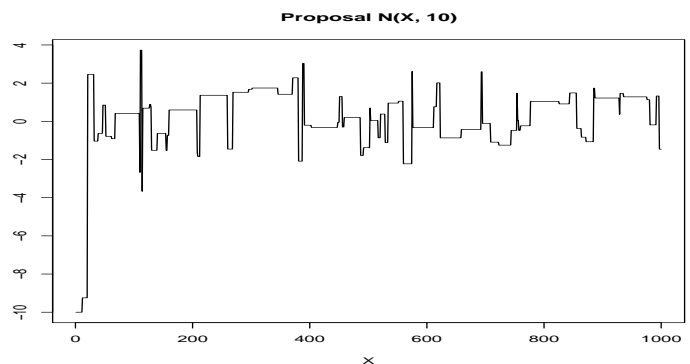
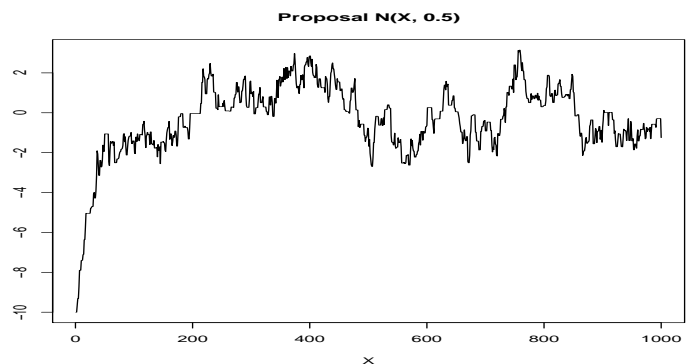
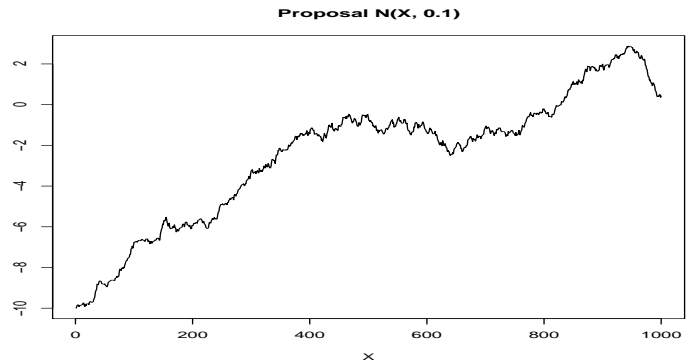
Proposal distributions with larger variance will make larger jumps around the support at the cost of each sample having a higher probability of being rejected. So proposals with especially large variance are prone to perform $X_{t+1} = X_t$ very frequently. On the other hand, proposals with small variances are more likely to result in the chain accepting the proposed point more frequently. Likewise, too small of variance will again result in slow mixing. Even though the proposed samples will be accepted frequently, the chain will take too small of steps from the last accepted sample to quickly move around the support.

The following figures are plots of Metropolis-Hastings Markov chains with stationary distribution $N(0, 1)$ with varying proposal distributions. Each chain has a starting value of -10 . The figures are squished, but at least they fit in one column.

VIII. DETERMINING SUFFICIENT BURN-IN

Precisely determining sufficient burn in a mathematical manner is difficult. It requires a thorough knowledge of Markov chain concepts related to sampling algorithms. For a brief survey on this topic, the reader is referred to Gareth Roberts section on this topic in *Markov Chain Monte Carlo in Practice*. Here the reader can find information on determining the rate of convergence by first proving that the chain in question is geometrically ergodic and then analyzing the eigenvalues that describe the transition probabilities (pp 48-49).

However, in practice it is often unnecessary to perform this analysis. Instead, the values of the chain can be plotted



against time. this allows for the analysis of both the rate of convergence and the mixing of the chain.

These plots allow us to guess a minimum burn-in with reasonable certainty and also determine how well the chain mixes for the given proposal distribution.

The first figure with proposal distribution $N(X, 0.1)$ mixes too slowly to be considered convergent with 1000 iterations.

The second figure with proposal distribution $N(X, 0.5)$ mixes considerably better. We would take its burn-in to be about 200 iterations.

The third figure with proposal distribution $N(X, 10)$ also mixes well. We would take its burn-in to be about 170 iterations.

The last figure with proposal $N(X, 50)$ does not mix well at all. The variance is too high so most of the proposed samples are rejected. It almost always maintains its current state and hence stays in particular regions of the support set.

IX. DETERMINING STOPPING TIME

Once burn-in is determined, the next decision that needs to be made is for how long to run the chain. The chain needs to be run for long enough to obtain enough precision for the estimator of interest. In this paper we focus on calculating the expectation of a probability distribution, however, computing the Monte Carlo variance is complicated by the lack of independence of the samples [2].

A simple way of determining stopping time is to run the simulation several times with different starting values for the same number of iterations. If the estimates differ by a statistically significant amount then the running time should be increased. If even very long running times result in varying approximations then the proposal distribution may warrant revision.

In general, the longer the chain is run, the better the estimation will be. However, there is certainly a point of diminishing returns. Once the chain has converged to the stationary distribution, future samples should be drawn in the correct proportions. It can be seen that simply increasing the running time does not necessarily mean that the estimate will improve. In fact, by looking at the Monte Carlo Standard Error (to be introduced shortly), we can see that adding samples can decrease the confidence in the estimate if the samples demonstrate high variance from $\hat{\mu}$.

X. STARTING VALUES

Each chain in the preceding figures have a starting value of -10 . Each of the well-mixing chains move away from this starting value quickly and stay fall into the stationary distribution of the chain.

Picking a starting value near the mode of the chain may seem like a good method of reducing the required burn-in. According to *Gilks, et al.*, even if the chain appears to converge immediately, it is still necessary to run the burn-in long enough to let the chain forget its starting value.

However, since these are Markov chains, the next value in the chain is only dependent on the prior value in the chain. So

if the chain is started within the support set of the stationary distribution, it may only take a few steps for the chain to forget its initial value.

XI. NUMBER OF CHAINS

Since MCMC relies on drawing random samples, one should not expect repeated runs to yield the exact same results. However, one should hope for repeated runs to yield results with very small deviation from one another. To prevent incorrect estimations, we suggest verifying this by running several chains in parallel.

Computationally, it makes sense to run one chain for every processor available unless there are other tasks waiting for the processors. When using multiple chains it is important to be careful of how the multiple estimates are handled. For example it may seem like a good idea to take the arithmetic mean of the estimates to get a better estimator This may, in principle, yield a numerically better approximation of the expectation but makes error reporting more complicated. Since the variance of each chain may differ significantly, the MCSE of one chain may not be representative of the others. Instead of attempting to combine the results of multiple chains together one should consider accepting the chain that has the lowest Monte Carlo Standard Error.

XII. MONTE CARLO STANDARD ERROR

We are using observed data to estimate population mean μ . We do not actually know the mean and would therefore like to report an error estimate. Without this estimate it is impossible to good (or bad) of an estimate we have attained. Monte Carlo estimates reported without standard errors have statistically little meaning [5]. How much confidence one should place in the estimate is completely underspecified.

Geyer provides a good explanation of MCSE in slides available at <http://www.stat.umn.edu/geyer/mcmc> [5]. If we know the true population variance, σ^2 , we can use a typical error estimator.

By the law of large numbers

$$\hat{\mu} \longrightarrow \mu, \quad n \rightarrow \infty$$

Then by the Central Limit Theorem

$$\sqrt{n}(\hat{\mu} - \mu) \longrightarrow N(0, \sigma^2), \quad n \rightarrow \infty$$

where $\sigma^2 = \text{var}(f(X_i))$. This is the actual population variance. Since we don't know the population variance σ^2 we need to estimate this quantity. The Monte Carlo Standard Error is a consistent estimate of the standard deviation σ such as

$$\hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^n (g(X_i) - \hat{\mu})^2}$$

With this estimator we can use Slutsky's Theorem to attain

$$Z = \frac{\hat{\mu}_n - \mu}{\frac{\hat{\sigma}_n}{\sqrt{n}}} \longrightarrow N(0, 1), \quad n \rightarrow \infty$$

Applying basic transformations we see

$$\hat{\mu} \sim N\left(\mu, \frac{\hat{\sigma}^2}{n}\right), \quad n \rightarrow \infty$$

The MCSE tells us how much we can expect our $\hat{\mu}$ to deviate from the true population μ . Note that as more samples are added, the MCSE does not necessarily decrease. In fact, it only decreases if the samples are close to the estimated mean $\hat{\mu}$. Consider the case of $\pi(\cdot)$ having many peaks and the chain has attained a good estimate $\hat{\mu}$ after t iterations. Then if at $t+1$ iterations the chain begins to explore a peak far from $\hat{\mu}$ the MCSE will increase as more samples from this peak are added.

By reporting the MCSE, $\hat{\sigma}$, and the number of samples used, n , enough information is provided to create confidence intervals for the expectation $\hat{\mu}$ at any α value. Taking $\alpha = 0.05$ gives $1 - \alpha = 0.95$ or 95 percent confidence interval.

Let Z represent the standardized distribution of our variable. To construct a 95 percent confidence interval for $\hat{\mu}$ we find z such that

$$P(-z \leq Z \leq z) = 0.95$$

By the definition of CDF:

$$\Phi(z) = P(Z \leq z) = 1 - \frac{\alpha}{2} = 0.975$$

$$z = \Phi^{-1}(\Phi(z)) = \Phi^{-1}(0.975) = 1.96$$

Then we construct the confidence interval as

$$P(-1.96 \leq Z \leq 1.96) = 0.95$$

$$P\left(\hat{\mu} - 1.96 \frac{\hat{\sigma}}{\sqrt{n}} \leq \mu \leq \hat{\mu} + 1.96 \frac{\hat{\sigma}}{\sqrt{n}}\right) = 0.95$$

XIII. CHAIN LENGTH REVISITED

In light of an understanding of MCSE, we propose a method for determining stopping time programatically. Instead of arbitrarily choosing to run the chain for an arbitrarily large number of samples, we propose running the chain until the desired confidence is reached.

Determining when to stop the chain programatically will yield much more precise results. This can be achieved by

- Keeping track of $\hat{\mu}$ while the chain is running.
- Computing MCSE after every M iterations for some M .
- Stopping the chain if the MCSE $< \epsilon$ for some ϵ .

The MCSE should be chosen depending on the confidence level desired. Smaller MCSE results in tighter confidence intervals for a given α level.

This method has surley been used before, but we have not found reference to it in the the literature reviewed thus far.

XIV. PSEUDO-CODE FOR METROPOLIS-HASTINGS

Metropolis-Hastings Markov Chain Monte Carlo is easy to describe in pseudo-code. The algorithm consists of 2 steps:

- Construct the Markov chain
- Perform Monte-Carlo Integration on the chain

```
#Construct the Markov Chain
X0 = ...           #initialize X0
chain = [X0]      #initialize chain
for i in range(1, max_iterations):
    Y = random sample from q(.|chain[i-1])
    U = random sample from Uniform(0,1)

    if U <= alpha(chain[i-1], Y):
        chain[i] = Y
    else:
        chain[i] = chain[i-1]

#Monte-Carlo Integration
total = 0
for i in range(burnin, max_iterations):
    total = total + chain[i]

total = total/(max_iterations-burnin)
```

XV. THE SOFTWARE IMPLEMENTATION

The software implamentation provided as a supplement to this paper is intended to provide a demonstration of Markov Chain Monte Carlo in a single dimension. It is written in C and must invoked from within the open source statistical software package *R*. The implementation relies upon several R functions to efficiently sample from the proposal distribution and evaluate $\pi(\cdot)$ at a given x . To make calling the source code easier we provide an R script that wraps the call to C in a function, calls the function with specified arguments, and plots a histogram of the chain after excluding the burn-in.

This Metropolis-Hastings sampler draws samples according to the pseudo-code in the preceding section. It outputs the approximated expectation and displays a histogram of chain.

To demonstrate the sampler we have run this to approximate a standard normal and a Chi-square distribution with 6 degrees of freedom.

A. Using the Software Implementation

To use the software implamentation you must be using a Unix system with R installed. It may work in Windows, but this has not been tested. First copy the files *MCMC.c* and *MCMC.r* into a directory and then open a terminal at the same location.

Compile the C code with

```
R CMD SHLIB MCMC.c
```

Then open R at this location enter into the terminal

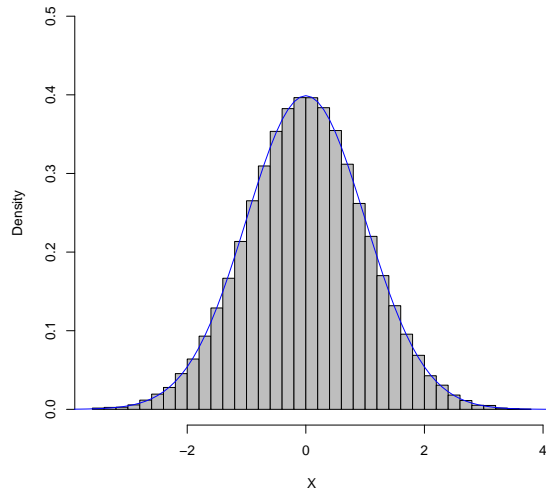


Fig. 1. Approximation of standard normal with 15,000 samples.

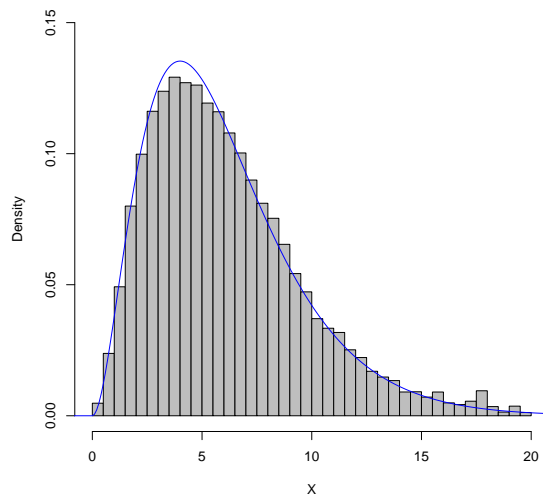


Fig. 2. Approximation of Chi-square with 6 degrees of freedom with 30,000 samples.

R

Now in R, to run the R script enter

```
source("MCMC.r")
```

B. Changing the Number of Iterations and Burn-In

To change the number of iterations and burn-in the file `MCMC.r` must be edited. Open this file in your favorite text editor and change the first 2 lines specifying the chain size and burn-in.

```
chainSize = 1000
burnin = 200
```

C. Changing the Distributions

To modify the distributions changes must be made to the C code itself. The first 2 functions of `MCMC.c` are responsible for the distributions.

The first function, `pi`, returns the value of the proposal distribution when evaluated at some point X . We have set this as a normal distribution with

```
return dnorm(X, 0, 1, FALSE);
```

where X is the point, 0 is the mean, 1 is the variance, and the `log` option is set to false.

The second function, `q`, returns a random sample from the proposal distribution. We have set this as a single-variate normal with

```
return rnorm(Y,10);
```

where Y is the mean and 10 is the variance.

D. Implementation Shortcomings

The implementation provided here is by no means perfect. It is provided as an example of how MCMC works. When calling C code from within R all variables must be passed to a shared library as pointers and be directly modified within C as such. You cannot create a new value and return it to R. Since we return the whole chain to allow for the generation of histograms, memory for the entire chain must be preallocated. Since the elements of the chain are stored in a type double array, the size of the chain is limited to $\frac{1}{16}$ the number of bytes of available memory after all overhead. This could be fixed by only storing the most recent element and immediately adding its value to the ergodic average. However this would require removing support for histograms and all other user access to the chain.

XVI. MULTI-DIMENSIONAL CONSIDERATIONS

The example implementation described above performs MCMC in one dimension. The real purpose of MCMC is to be able to attain estimation approximations in high-dimensional spaces in which numerical evaluation may difficult and/or inaccurate.

The multi-dimensional version is nearly identical. Instead of X being a point at which the posterior is evaluated, it is regarded as a vector. For example, the posterior distribution may depend on n parameters, so we would have $X = x_1, x_2, \dots, x_n$. Then it would be evaluated as $\pi(X) = \pi(x_1, x_2, \dots, x_n)$. The goal is still to approximate $E(f(X))$ but we must do this by varying each $x_i \in X$.

Obviously, as the number of dimensions are increased so too will be the necessary running time. This is especially true if a random sample of n dimensions is drawn and then accepted or rejected as a whole. To counteract this we introduce single-component Metropolis-Hastings.

A. Single-Component Metropolis-Hastings

Single-component Metropolis-Hastings works by dividing multi-dimensional X into $X_{.1}, X_{.2}, \dots, X_{.h}$ and drawing a proposal sample for each which is then accepted or rejected. There is no restriction on the dimensions of the components $X_{.1}, X_{.2}, \dots, X_{.h}$. They may all be single-dimensional values, multi-dimensional with the same dimension, or multi-dimensional with varying dimensions.

One iteration of single-component Metropolis-Hastings takes h update steps to complete. A total of h candidates must be drawn for X and the distribution from which they are drawn may depend on the new values of the already updated components and the prior values of the non-updated components. That is, the proposal distribution for $Y_{.i}$ may depend on $X_{t,-1} = X_{t+1,1}, \dots, X_{t+1,i-1}, X_{t,i}, \dots, X_{t,h}$.

The i^{th} proposal is then accepted with probability

$$\alpha(X_{.-i}, X_{.i}, Y_{.i}) = \min\left(1, \frac{\pi(Y_{.i}|X_{.-i})q_i(X_{.i}|Y_{.i}, X_{.-i})}{\pi(X_{.i}|Y_{.-i})q_i(Y_{.i}|X_{.i}, X_{.-i})}\right)$$

XVII. OTHER MCMC ALGORITHMS

The Metropolis-Hastings Algorithm (1970) was the first MCMC algorithm applicable to arbitrary probability distributions. It uses the ratio described in α to either accept or reject a new sample. This works well but is not the only way to solve the problem. In fact, any Markov chain with stationary distribution π will do as long as the chain is:

- 1) *irreducible*: for any state in the Markov chain there is a positive probability of visiting all other states [6].
- 2) *aperiodic*: The Markov chain does not fall into cycles.

Despite the existence of many other MCMC algorithms, Metropolis-Hastings works well and is still widely used in modern-day applications. However, there are many other MCMC algorithms that may be better suited to particular problems. We offer a brief overview of some of these algorithms to help the reader pick the most appropriate method for his or her particular problem.

A. Gibbs Sampler

Gibbs sampling is the most simple method of MCMC. It is useful when the n conditional distributions of the n parameters of interest can be sampled from directly. One sample is drawn from each of the n conditional distributions. These n values are stored as the new state of the chain. Unlike Metropolis-Hastings MCMC, this method does not require any tuning as there is no proposal distribution to tune. The only decisions to make are how many chains to run and how long to run the chains.

If we have data D , parameters θ_1, θ_2 and can sample from

$$p(\theta_1|\theta_2, D) \text{ and } p(\theta_2|\theta_1, D)$$

then we can construct a Markov chain

$$(\theta_1^{(1)}, \theta_2^{(1)}), (\theta_1^{(2)}, \theta_2^{(2)}), \dots, (\theta_1^{(n)}, \theta_2^{(n)})$$

by sampling

$$\theta_1^{(k)} \sim p(\theta_1|\theta_2^{(k-1)}, D) \quad \text{and} \quad \theta_2^{(k)} \sim p(\theta_2|\theta_1^{(k)}, D)$$

for $k = 1, \dots, n$. The stationary distribution of the Markov chain will then be the desired $p(\theta_1, \theta_2|D)$. Modern-day Superman to some yet-to-be-found Lois Lane, part-time voider of warranties, and full-time engineer with a penchant This can be generalized to any dimension by sampling each of the parameters as

$$\theta_i^{(k)} \sim p(\theta_i|\theta_1^{(k)}, \dots, \theta_{i-1}^{(k)}, \theta_{i+1}^{(k-1)}, \dots, \theta_n^{(k-1)}, D).$$

B. Coupling from the Past

Coupling from the past theoretically gives perfect samples from the stationary distribution at the cost of an unbounded, but finite in expectation, running time [7]. For an in-depth look at coupling from the past, the reader is referred to *Coupling from the past: a user's guide*[8].

C. Metropolis-Hastings-Green

Metropolis-Hastings-Green is useful when working with a probability distribution that is a mixture of distributions having supports of different dimension [9]. In M-H-G the unnormalized density $\pi(X)$ is replaced with an unnormalized measure on the state space and the proposal density $q(X|Y)$ is replaced by a proposal kernel $Q(X, A)$. For an informal introduction to M-H-G MCMC the reader is referred to Charles J. Geyer's note on the subject, freely available as cited in this paper [9].

D. Avoiding Random Walks

All of the methods that have been discussed thus far are *random-walk* varieties of MCMC. These move around the support of $\pi(\cdot)$ randomly and therefore the steps have no tendency to follow a particular direction. As a result, the chain may repeatedly explore the same region of the support before moving on to other regions, decreasing the rate of convergence.

Some optimizations exist to avoid random walks. Hybrid Monte Carlo or 'Hamiltonian Monte Carlo' tries to avoid random walks by imposing momentum on the chain using Hamiltonian dynamics [7]. The chain then moves across the sample space more rapidly, decreasing the correlation between the samples and converging to the stationary distribution more rapidly.

The HMC method allows for global moves while keeping the acceptance probability high [?]. Global jumps are not usually made in the Metropolis-Hastings algorithm because the acceptance probability would be prohibitively low. This problem is handled with single-component Metropolis Hastings (discussed earlier) which only updates one component of θ at a time. As a result the update step of HMC is much faster since the update is done at once instead of iteratively updating each individual component.

E. Monte Carlo EM

Expectation Maximization provides an alternative method for estimating the maximum likelihood estimators of a probability model relying on both observed and unobserved data. EM works by performing iterative refinement on the expectation of the parameters θ of a probability model. Let X consist of visible and latent variables, $X = x_v, x_h$. Then the

maximum likelihood $p(x_v|\theta)$ can be found using EM [6]. The algorithm consists of two steps.

- 1) *E Step*. Compute the expected value of the complete log-likelihood function with respect to the latent variables.

$$Q(\theta) = \int_{X_h} \log(p(x_h, x_v|\theta))p(x_h|x_v, \theta^t)dx_h$$

- 2) *M Step*. Perform the maximization

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} Q(\theta)$$

In many practical situations the expectation in the *E* step is an intractable integral. One way to solve this problem is to use MCMC to sample from $p(x_h|x_v, \theta^{(old)})$ by drawing N dependent samples. According to *An Introduction to MCMC for Machine Learning* by Andrieu, *et al.* we can then replace $Q(\theta)$ with

$$\hat{Q}(\theta) = \frac{1}{N} \sum_{j=1}^N \log(p(x_h^{(j)}, x_v|\theta))$$

and using these estimates to estimates to Then the expectation is performed using this estimator.

In summary, MCMC-EM works by sampling from the unobservable data and using this data to approximate the log-likelihood of the parameters of interest. The log-likelihood is then maximized and the process starts over again using the new and improved parameters.

VIII. NUMERICAL INTEGRATION BY QUADRATURES

Quadratures are a long-known and well-loved method of performing numerical integration. Many of the methods work by finding easy to integrate interpolating functions on many subintervals and then integrating each subinterval. These methods work well for integration over a single dimension.

Expanding these methods into multiple dimensions is generally achieved by treating the multidimensional integral as multiple one dimensional integrals using Fubini's Theorem. The problem with this method is that the computational cost increases exponentially with dimension. If 1,000 points are required in 1-D then typically 1,000,000 are required in 2-D and 1,000,000,000 in 3-D [10].

The *curse of dimensionality* is partially overcome by the *sparse grid scheme*. The method is based on 1-D quadratures but combines the results in a more sophisticated manner using Smolyak's quadrature rule [11]. This method works well in many situations and has the advantage of giving precise error estimation for a normalized posterior. Sparse grids do not rely on the concepts of burn-in or chain length but do require precise choice of a basis functions. A wrong basis function will result in a wrong result. Similarly, in MCMC a poor choice in proposal will hinder convergence but not yield a wrong answer.

XIX. CONCLUSION

MCMC methods are a popular way of computing expectations and handling various other multi-dimensional integrals. Unfortunately there are not many other methods to compute such integrals that yield accurate results but sparse grids do offer some hope. Among the MCMC methods, the general purpose Metropolis-Hastings sampler is a very diverse algorithm that can be applied to almost all situations. The exception to this is when one needs to handle a mixture of distributions having different dimensions. In this case the even more general and more complicated Metropolis-Hastings-Green method should be used. If the conditional distributions for each variable are known, Gibbs sampling can provide just as good of a chain and requires no tuning.

Like all algorithms, Metropolis-Hastings MCMC is not a silver bullet. Fast convergence requires a well thought-out prior and rapid mixing often requires significant tuning. High-dimensional problems require massive number of samples which can be computationally expensive. This is especially true when a large number of multi-dimensional integrals need to be handled. In this situation it may be advisable to implement the optimizations mentioned in the *Avoiding Random Walks* section of this paper.

Despite these problems, MCMC works surprisingly well. By taking advantage of MCSE, estimates can be made at almost any level of desired precision at the cost of additional computer time. This is what makes an MCMC estimate worth using. It is no surprise that since Metropolis introduced the method in 1952 it has gained popularity in every field that requires the computation of high-dimensional problems.

Regardless of the application, most people using MCMC are interested in computing the expectation of a probability distribution, usually a Bayesian posterior. In physics this may be the expected location of an electron or in genetics it may be the probability that a single nucleotide polymorphism (SNP) is a causal for a given disease model.

In every case, picking an adequate proposal is one of the most important challenges faced. A multivariate normal is often a good choice, but the covariance matrix needs to be tuned for the data at hand. Without strong prior knowledge of the distribution from which the data arise, this requires monitoring how well the chain is mixing for each parameter and making adjustments based on this information. Fortunately, this is not very difficult. Unfortunately, this requires a human in the loop. These modifications can be made programatically at the cost of a more complicated software implementation.

Ensuring that a chain has good mixing and determining a proper burn-in value are important considerations. The mixing of the chain is generally determined by variance of the proposal, also known as "step-size". When using a multi-variate proposal, the step-size for each parameter can be set independently. Mixing and burn-in can be analyzed visually as demonstrated the figures in this paper. However, the parameters must be analyzed one at a time unless 3-D graphing techniques are used.

There are various MCMC implementations available for download on the web. The BUGS (Bayesian inference Using Gibbs Sampling) Project provides a comprehensive software package for Gibbs Sampling. Charles Geyer at the University of Minnesota has released an R package for doing “Simple, but general” random-walk Metropolis-Hastings. It is available at <http://www.stat.umn.edu/geyer/mcmc/>.

REFERENCES

- [1] C. J. Geyer, “Practical markov chain monte carlo,” *Statistical Science*, vol. 7, no. 4, 1992.
- [2] D. J. S. Walter R. Gilks, *Markov Chain Monte Carlo in Practice*. First CRC Press, 1 ed., 1998.
- [3] “xbeta: Metropolis-hastings algorithm.” <http://xbeta.org/wiki/show/Metropolis-Hastings+algorithm>.
- [4] “The mcmc procedure: Tuning the proposal distribution.” http://support.sas.com/documentation/cdl/en/statug/59654/HTML/default/statug_mcmc_sect022.htm, 2008.
- [5] C. Geyer, “Introduction to markov chain monte carlo slides.”
- [6] C. Andrieu, “An introduction to mcmc for machine learning,” 2003.
- [7] Wikipedia, “Markov chain monte carlo — wikipedia, the free encyclopedia,” 2009. [Online; accessed 10-May-2009].
- [8] D. Propp, James; Wilson, “Coupling from the past: a user’s guide,” *Microsurveys in discrete probability*, p. 181192, 1998. MR1630414.
- [9] C. J. Geyer, “The metropolis-hastings-green algorithm.” <http://www.stat.umn.edu/geyer/f05/8931/bmhg.pdf>, December 2003.
- [10] D. H. Bailey, “High-precision, high-dimension integration,” tech. rep.
- [11] Wikipedia, “Numerical integration — wikipedia, the free encyclopedia.” http://en.wikipedia.org/w/index.php?title=Numerical_integration, 2009. [Online; accessed 14-May-2009].