Markov Chain Monte Carlo method using Metropolis–Hastings Algorithm

Abhinav Gupta-150123001,
Arkadeep Das-150123053,
Chiranjiv Goyal-150123008,
Hiten Sethiya-150123015,
Manas Daruka-150123020

April 12, 2017

Abstract

This article is a self-contained introduction to the Metropolis-Hastings algorithm, this ubiquitous tool for producing dependent simulations from an arbitrary distribution. The document illustrates the principles of the methodology on simple examples with R codes and provides entries to the recent extensions of the method. Key words and phrases: Bayesian inference, Markov chains, MCMC methods, Metropolis–Hastings algorithm, intractable density, Gibbs sampler, Langevin diffusion, Hamiltonian Monte Carlo.

1 Introduction

There are many reasons why computing a integral like

$$\mathbf{I(x)} = \int_{\mathbf{x}} \mathbf{d\pi(x)dx}$$

where d π is a 0probability measure, may prove intractable, from the shape of the domain X to the dimension of X (and x), to the complexity of one of the functions h or π. Standard numerical methods may be hindered by the same reasons. Similar difficulties (may) occur when attempting to find the extrema of π over the domain X .

This is why the recourse to Monte Carlo methods may prove unavoidable: exploiting the probabilistic nature of π and its weighting of the domain X is often the most natural and most efficient way to produce approximations to integrals connected with π and to determine the regions of the domain X that are more heavily weighted by π.
The Monte Carlo approach (Hammersley and Handscomb, 1964; Rubinstein, 1981) emerged with computers, at the end of WWII, as it relies on the ability of producing a large number of realisations of a random variable distributed according to a given distribution, taking advantage of the stabilisation of the empirical average predicted by the Law of Large Numbers.

However, producing simulations from a specific distribution may prove near impossible or quite costly and therefore the (standard) Monte Carlo may also face intractable situations. An indirect approach to the simulation of complex distributions and in particular to the curse of dimensionality met by regular Monte Carlo methods is to use a Markov chain associated with this target distribution, using Markov chain theory to validate the convergence of the chain to the distribution of interest and the stabilisation of empirical averages (Meyn and Tweedie, 1994).

It is thus little surprise that Markov chain Monte Carlo (MCMC) methods have been used for almost as long as the original Monte Carlo techniques, even though their impact on Statistics has not been truly felt until the very early 1990s. A comprehensive entry about the history of MCMC methods can be found in Robert and Casella (2010). The paper is organised as follows: in Section

2, we define and justify the Metropolis–Hastings algorithm, along historical notes about its origin. In Section 2, we provide the history of the Metropolis Hastings Algorithm.

A peek at Markov Chain Theory is provided in Section 3. Section 4 includes detailed explanation of the standard Metropolis–Hastings algorithm is provided with examples, while Section 5 and Section 6 includes the code of Assymetric and Symmetric Walks respectively in R along with their respective outputs. Section 7 provides the various Applications of Metropolis-Hastings Algorithm.

2 History

The algorithm was named after Nicholas Metropolis, who was an author along with Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller of the 1953 paper Equation of State Calculations by Fast Computing Machines which first proposed the algorithm for the case of symmetrical proposal distributions, and W. K. Hastings who extended it to the more general case in 1970.
There is controversy over the credit for discovery of the algorithm. Edward Teller states in his memoirs that the five authors of the 1953 paper worked together for "days (and nights)". M. Rosenbluth, in an oral history recorded shortly before his death credits E. Teller with posing the original problem, himself with solving it, and A.W. Rosenbluth (his wife) with programming the computer.
According to M. Rosenbluth, neither Metropolis nor A.H. Teller participated in any way. Rosenbluth's account of events is supported by other contemporary recollections. According to Roy Glauber and Emilio Segrè, the original algorithm was invented by Enrico Fermi and reinvented by Stan Ulam.

3 A peek at Markov chain theory

A Markov chain X(t) is a sequence of dependent random variables

$$\mathbf{((0), X(1), X(2), \ldots, X(t), \ldots)}$$

such that the probability distribution of X(t) given the past variables depends only on X(t−1). This conditional probability distribution is called a transition kernel or a Markov kernel K; that is,

$$\mathbf{X(t+1) \mid X(0), X(1), X(2), \ldots, X(t) \sim K(X(t), X(t+1))}$$

For example, a simple random walk Markov chain satisfies X(t+1) = X(t) + t , where t $\sim$ N (0, 1), independently of X(t) ; therefore, the Markov kernel $\mathbf{K(X(t)\ ,\ X(t+1))}$ corresponds to a N (X(t) , 1) density. For the most part, the Markov chains encountered in Markov chain Monte Carlo (MCMC) settings enjoy a very strong stability property. Indeed, a stationary probability distribution exists by construction for those chains; that is, there exists a probability distribution f such that if $\mathbf{X(t) \sim f}$, then $\mathbf{X(t+1) \sim f}$.

The existence of a stationary distribution (or stationarity) imposes a preliminary constraint on K called irreducibility in the theory of Markov chains, which is that the kernel K allows for free moves all over the stater-space, namely that, no matter the starting value X(0), the sequence X(t) has a positive probability of eventually reaching any region of the state-space. (A sufficient condition is that K(x, ·) $>$ 0 everywhere.) The existence of a stationary distribution has major consequences on the behavior of the chain X(t), one of which being that most of the chains involved in MCMC algorithms are recurrent, that is, they will return to any arbitrary non-negligible set an infinite number of times

4 Analysis of the Algorithm

The **Metropolis–Hastings** algorithm can draw samples from any probability distribution **P(x)**, provided you can compute the value of a function **f(x)** that is proportional to the density **P(x)**. The lax requirement that **f(x)** should be merely proportional to the density, rather than exactly equal to it, makes the Metropolis–Hastings algorithm particularly useful, because calculating the necessary normalization factor is often extremely difficult in practice.

Metropolis algorithm, a special case of the Metropolis–Hastings algorithm where the proposal function is symmetric.

- In symmetrical walk:

  Calculate the acceptance ratio $\alpha = \frac{f(x')}{f(x_t)}$, which will be used to decide whether to accept or reject the candidate. Because f is proportional to the density of P, we have that $\alpha = \frac{f(x')}{f(x_t)} = \frac{P(x')}{P(x_t)}$.

  Note that the acceptance ratio α indicates how probable the new proposed sample is with respect to the current sample, according to the distribution **P(x)**. If we attempt to move to a point that is more probable than the existing point (i.e. a point in a higher-density region of **P(x)**), we will always accept the move. However, if we attempt to move to a less probable point, we will sometimes reject the move, and the more the relative drop in probability, the more likely we are to reject the new point. Thus, we will tend to stay in (and return large numbers of samples from) high-density regions of **P(x)**, while only occasionally visiting low-density regions. Intuitively, this is why this algorithm works, and returns samples that follow the desired distribution **P(x)**.

  Compared with an algorithm like adaptive rejection sampling that directly generates independent samples from a distribution, Metropolis–Hastings and other MCMC algorithms have a number of **disadvantages**:

- The samples are correlated. Even though over the long term they do correctly follow **P(x)**, a set of nearby samples will be correlated with each other and not correctly reflect the distribution. This means that if we want a set of independent samples, we have to throw away the majority of samples and only take every nth sample, for some value of n (typically determined by examining the autocorrelation between adjacent samples). **Autocorrelation can be reduced by increasing the jumping width** (the average size of a jump, which is related to the variance of the jumping distribution), but this will also increase the likelihood of rejection of the proposed jump. Too large or too small a jumping size will lead to a **slow-mixing Markov chain**, i.e. a highly correlated set of samples, so that a very large number of samples will be needed to get a reasonable estimate of any desired property of the distribution.

- Although the Markov chain eventually converges to the desired distribution, the initial samples may follow a very different distribution, especially if the starting point is in a region of low density. As a result, a **burn-in period** is typically necessary, where an initial number of samples (e.g. the first 1,000 or so) are thrown away.

1. The Metropolis–Hastings algorithm involves designing a Markov process (by constructing transition probabilities) which fulfils the two above conditions, such that its stationary distribution $\pi(x)$ is chosen to be **P(x)**. The derivation of the algorithm starts with the condition of detailed balance:

$$P(x'|x)P(x) = P(x|x')P(x')$$

2. $\frac{P(x'|x)}{P(x|x')} = \frac{P(x')}{P(x)}$

3. he proposal distribution **g(x'|x)** is the conditional probability of proposing a state x' given x, and the acceptance distribution **A(x'|x)** the conditional probability to accept the proposed state x'. The transition probability can be written as the product of them:

$$P(x'|x) = g(x'|x)A(x'|x)$$

4. $\frac{A(x'|x)}{A(x|x')} = \frac{P(x')}{P(x)}\frac{g(x|x')}{g(x'|x)}$

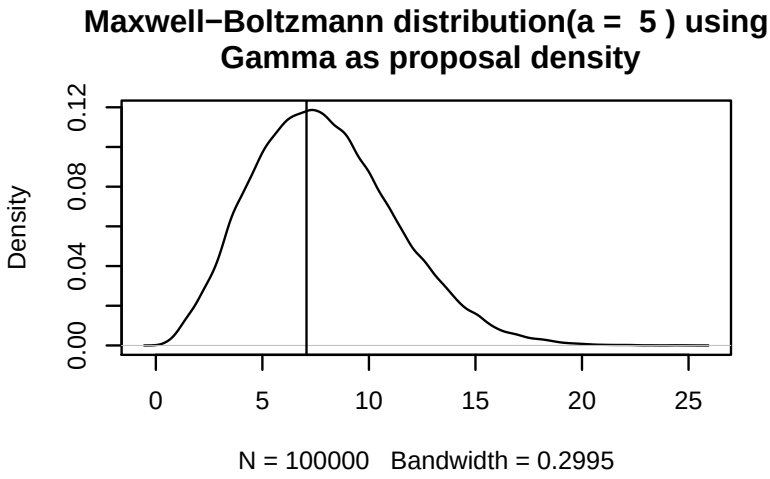
5. A(x'|x) $= \min(1, \frac{P(x')}{P(x)}\frac{g(x|x')}{g(x'|x)})$

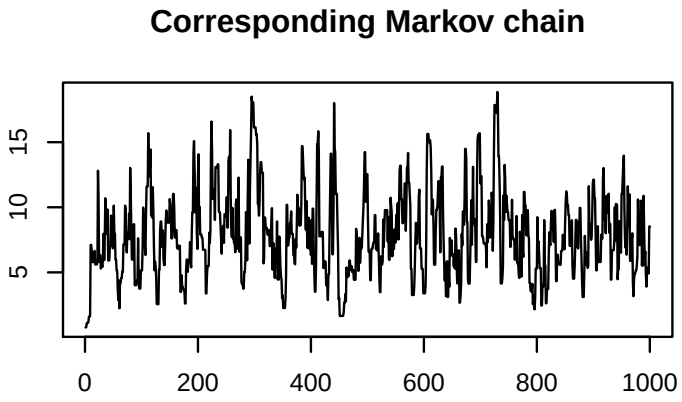
From the graph we can judge how stationary distribution exists for the markov chain and is $\pi(x)$.

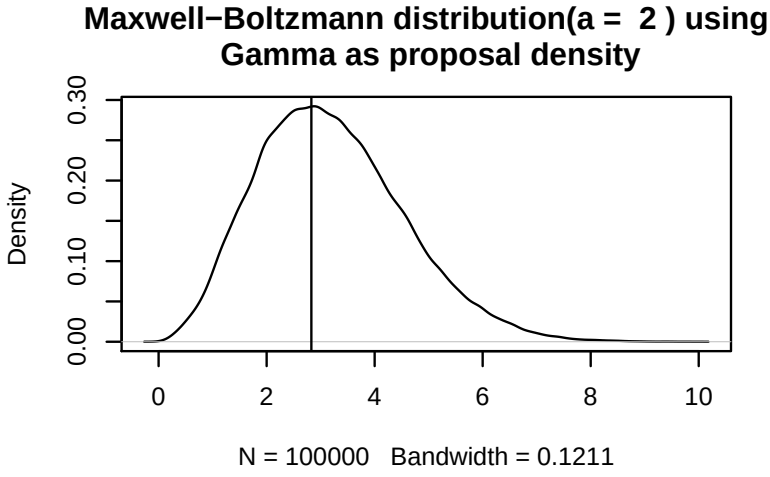
5 Code of Asymmetric Walk in R

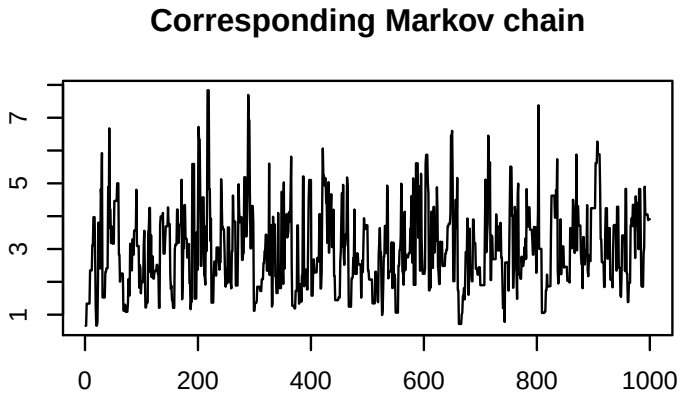
```
sampler <- function()
{
  trials = 100000
  y <- as.array(trials)
  y[1] <- rgamma(1, shape=1, rate = 1)
  u <- runif(trials)
  count <- 0
  for (i in 2:trials)
  {
   y.p <- rgamma(n=1, shape=y[i-1], rate=1)
   f_ratio <- dist(y.p)/dist(y[i-1])
   q_ratio <- dgamma(x = y[i-1],shape = y.p, rate = 1)/dgamma(x = y.p,shape = y[i-1],
       rate = 1)
   acc_prob <- f_ratio*q_ratio

   if (u[i] <= acc_prob)
   {
    y[i] <- y.p
    count = count +1
    }
   else
   {
    y[i] <- y[i-1]
   }
  }
  cat(count)
  return(y)
}

dist <- function(x){
  return(x*x*exp(-(x^2)/fctr))
}
driver <- function(fctr)
  {
  a= (sqrt(fctr/2))
  heading = paste("Maxwell-Boltzmann distribution(a = ",a,") using\n Gamma as proposal
      density")
  result=sampler()
  plot(density(result),main = heading)
  abline(v = sqrt(fctr))
  plot(result[1:1000],type = 'l',main = "Corresponding Markov chain",xlab = '',ylab = '')
}
```

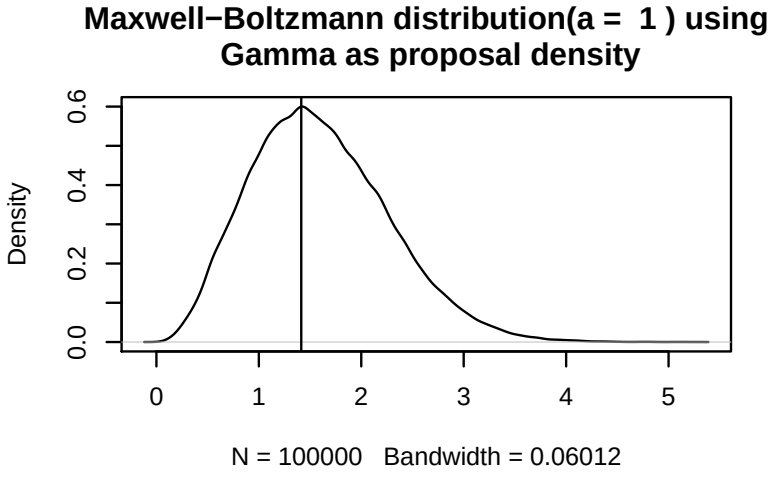
```
par(mfrow = c(3,2))


fctr <- 2*5*5
driver(fctr)

fctr <- 2*2*2
driver(fctr)

fctr <- 2
driver(fctr)
dev.off(0)
```

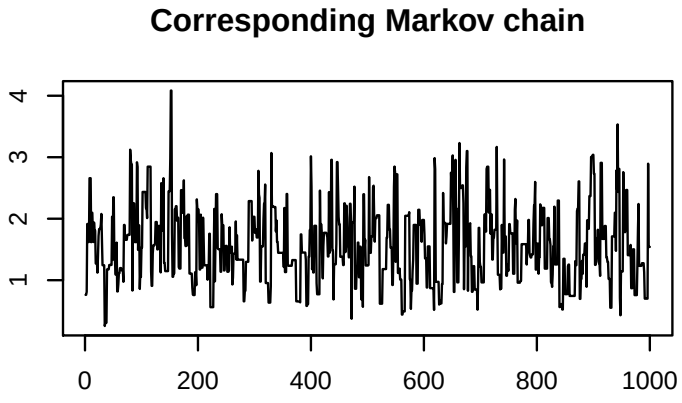

Maxwell−Boltzmann distribution(a = 5) using Gamma as proposal density
Density
0.00
0.04
0.08
0.12
0
5
10
15
20
25
N = 100000 Bandwidth = 0.2995


Corresponding Markov chain
5
10
15
0
200
400
600
800
1000


Maxwell−Boltzmann distribution(a = 2) using Gamma as proposal density
Density
0.00
0.10
0.20
0.30
0
2
4
6
8
10
N = 100000 Bandwidth = 0.1211

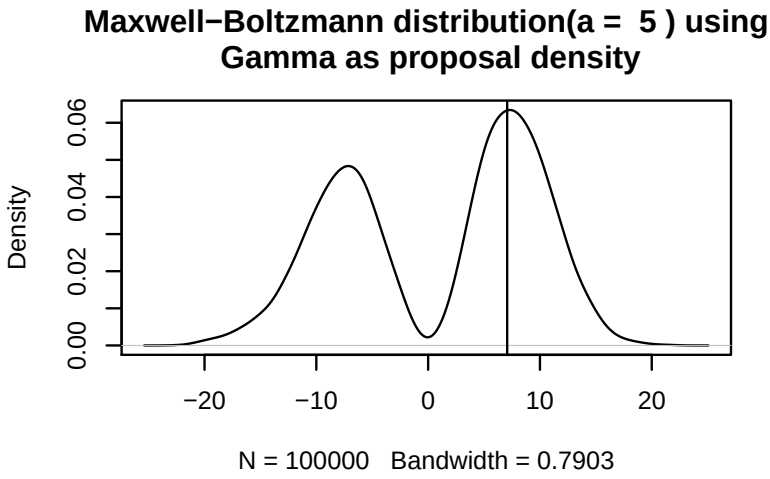

Corresponding Markov chain
1
3
5
7
0
200
400
600
800
1000

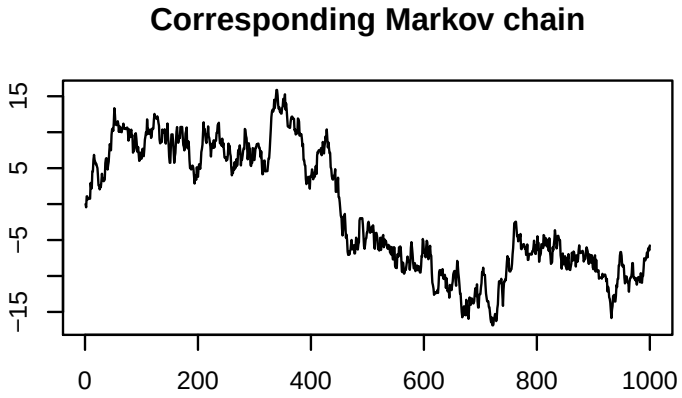

Maxwell−Boltzmann distribution(a = 1) using Gamma as proposal density
Density
0.0
0.2
0.4
0.6
0
1
2
3
4
5
N = 100000 Bandwidth = 0.06012

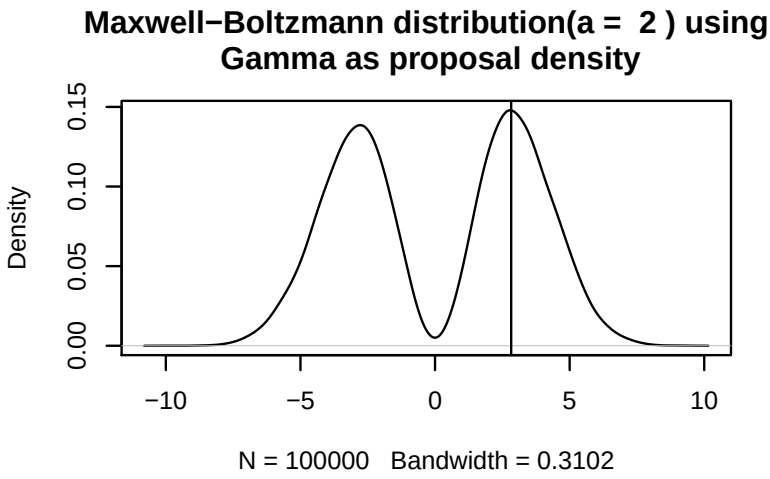

Corresponding Markov chain
1
2
3
4
0
200
400
600
800
1000

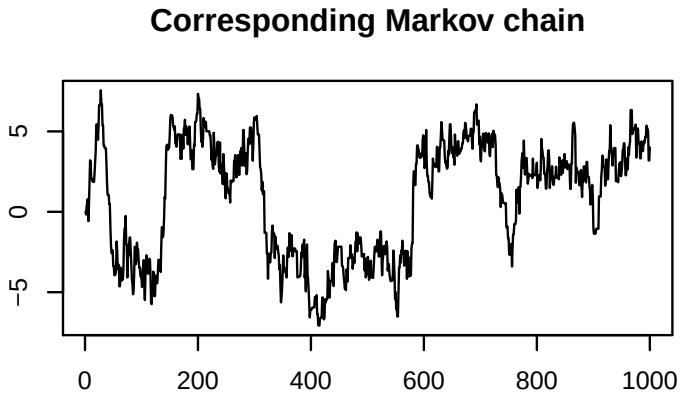
6 Code of Symmetric Walk in R

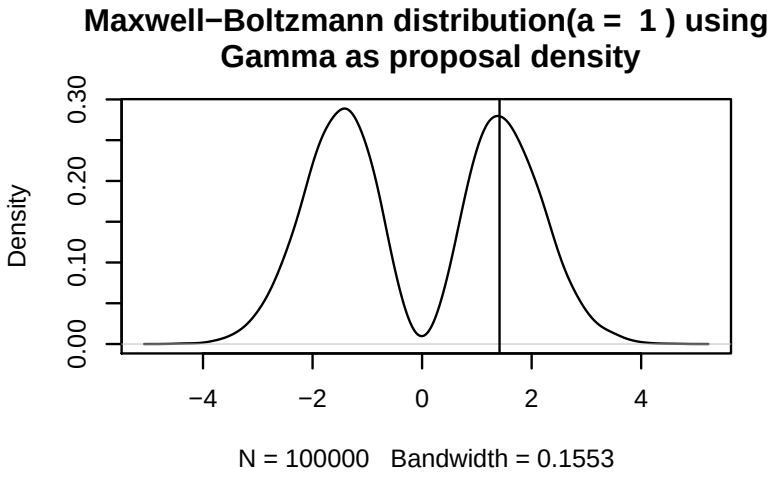
```
sampler<- function()
{
  trials <- 100000
  y<-as.array(trials)
  y[1]=0
  count = 0
  u <- runif(trials)
  for (i in 2:trials)
  {
    y.p <- prop(y[i-1])
    f_ratio <- dist(y.p)/dist(y[i-1])
    acc_prop <- f_ratio

    if(u[i] <= acc_prop)
    {
      y[i] = y.p
      count = count+1
    }
    else
    {
      y[i]=y[i-1]
    }
  }
  cat(count)
  return(y)
}

dist <- function(x)
{
  return (x*x*exp(-(x^2)/fctr))
}

prop <- function(x)
{
  return (rnorm(1,x,1))
}

driver <- function(fctr)
{
  a= (sqrt(fctr/2))
  heading = paste("Maxwell-Boltzmann distribution(a = ",a,") using\n Gamma as proposal
      density")
  result = sampler()
  plot(density(result),main = heading)
  abline(v = sqrt(fctr))
  plot(result[1:1000],type = 'l',main = "Corresponding Markov chain",xlab = '',ylab = '')
}


par(mfrow = c(3,2))

fctr = 2*5*5
driver(fctr)

fctr = 2*2*2
driver(fctr)

fctr = 2
driver(fctr)
dev.off(0)
```

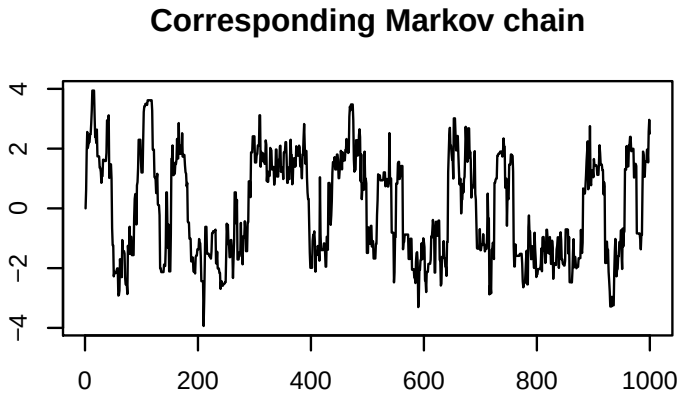

Maxwell−Boltzmann distribution(a = 5) using
Gamma as proposal density
Density
0.00
0.02
0.04
0.06
−20
−10
0
10
20
N = 100000 Bandwidth = 0.7903


Corresponding Markov chain
−15
−5
5
15
0
200
400
600
800
1000


Maxwell−Boltzmann distribution(a = 2) using
Gamma as proposal density
Density
0.00
0.05
0.10
0.15
−10
−5
0
5
10
N = 100000 Bandwidth = 0.3102


Corresponding Markov chain
−5
0
5
0
200
400
600
800
1000


Maxwell−Boltzmann distribution(a = 1) using
Gamma as proposal density
Density
0.00
0.10
0.20
0.30
−4
−2
0
2
4
N = 100000 Bandwidth = 0.1553


Corresponding Markov chain
−4
−2
0
2
4
0
200
400
600
800
1000

7 Applications

- A common use of Metropolis–Hastings algorithm is to compute an integral. Specifically, consider a space $\Omega \subset R$ and a probability distribution P(x) over Ω,$x \epsilon R$. Metropolis-Hastings can estimate an integral of the form of

$$\sigma(x) = \int_{\sigma} A(x)P(x)dx$$

where A(x) is an arbitrary function of interest. estimating **P(E)** can be accomplished by estimating the expected value of the indicator function

$$AE(x) \equiv 1E(x)$$

which is 1 when

$$E(x) \epsilon [E, E + \triangle]$$

and zero otherwise.Metropolis-Hastings can be used here to sample (rare) states more likely and thus increase the number of samples used to estimate **P(E)** on the tails. This can be done e.g. by using a sampling distribution $\pi(x)$ to favor those states .

- Metropolis - Hasting Algorithm satisfies the so-called detailed balance condition

$$f(x)K(y|x) = f(y)K(x|y)$$

, from which we can deduce that f is the stationary distribution of the chain X(t) by integrating each side of the equality in x. That Metropolis-Hasting Algorithm is naturally associated with f as its stationary distribution thus comes quite easily as a consequence of the detailed balance condition for an arbitrary choice of the pair (f, q). In practice, the performance of the algorithm will obviously strongly depend on this choice of q, but Metropolis-Hasting Algorithm can be compared with iid sampling.

.