Understanding the Metropolis-Hastings Algorithm

RK

June 7, 2015

Abstract

This document summarized the main points of the paper, "Understanding the Metropolis-Hastings Algorithm" by Siddhartha Chib and Edward Greenberg. The document also contains `R` code that replicates the results for all the examples mentioned in the paper.

Contents

1. Introduction

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 specific case of the canonical ensemble. W.K. Hastings who extended it to the more general case in 1970. The algorithm is versatile and gives rise to Gibbs sampler as a special case. This paper is meant to provide a simple or intuitive exposition of the M-H algorithm.

2. Acceptance-Rejection Sampling

The objective is to generate samples from the target density $\pi(x) = f(x)/K$ where $x \in \mathcal{R}^d$, $f(x)$ is the unnormalized density and K is the normalizing constant. Let $h(x)$ be a density that can be simulated by some known method, and suppose that there is a known constant c such that $f(x) \leq ch(x) \forall x$. Then to obtain a random variate from $\pi(\cdot)$, we have

1. Generate a candidate Z from $h(\cdot)$ and a value u from $U(0,1)$
2. If $u \leq f(Z)/ch(Z)$ return $Z = y$, else go to step 1.

The expected number of draws is obtained as follows

$$E\left[1_{u \leq f(Z)/ch(Z)}\right] = E\left[E\left[1_{u \leq f(Z)/ch(Z)}|Z\right]\right] = E\left[f(Z)/ch(Z)\right]] = 1/c$$

This means that the sampling method can be optimized by setting

$$c = \sup_x \frac{f(x)}{h(x)}$$

It can be easily seen that the above procedure indeed generates a sample from $f(x)$

$$P(X < z) = \frac{P(X < z, u \leq f(z)/(ch(z)))}{P(u \leq f(z)/ch(z))} = \frac{\int_{-\infty}^{z} h(z)f(z)/ch(z)dz}{1/c} = \int_{-\infty}^{z} f(z)dz$$

3. Markov chain Monte Carlo Simulation

Let $P(x, A)$ be the transition kernel for $x \in \mathcal{R}^d$ and $A \in \mathcal{B}$, the Borel sigma field on $\mathcal{R}^d$. If the transition kernel converges to an invariant distribution, the the following holds good

$$\pi^*(y) = \int_{\mathcal{R}^d} \pi(x)P(x, dy)dx$$

MCMC methods turns the CTMC theory around; the invariant density $\pi(\cdot)$ is known but the transition kernel is unknown. To generate samples from $\pi(\cdot)$, the methods find a transition kernel that converges to $\pi(\cdot)$. Finding the transition kernel might appear like an infinite search problem but it becomes a tractable problem once the detailed balance conditions are used. If a function $p(x, y)$ satisfies the reversibility condition, then the kernel converges to the target density

$$\pi(x)p(x, y) = \pi(y)p(y, x)$$

4. Metropolis Hastings Algorithm

The basic idea is that the any candidate generating density $q(x, y)$ that is used in Accept-Reject method will not automatically satisfy detailed balance conditions. The classic AR procedure needs to be tweaked accordingly so that the transitions from x to y satisfy reversibility condition. MH algo introduces *probability of a move* denoted by $\alpha(x, y)$. This is the critical probability of acceptance of a new value from a candidate generating density

$$\alpha(x,y) = \begin{cases} \min\left[\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right], & \text{if } \pi(x)q(x,y) > 0 \\ 1, & \text{otherwise} \end{cases}$$

The kernel should have a nonzero probability that the process remains at x. This probability is

$$r(x) = 1 - \int_{\mathcal{R}^d} q(x,y)\alpha(x,y)dy$$

Hence the transition kernel of M-H chain denoted by

$$P_{MH}(x, dy) = q(x,y)\alpha(x,y)dy + \left[1 - \int_{\mathcal{R}^d} q(x,y)\alpha(x,y)dy\right]\delta_x(dy)$$

- The algorithm is specified by its candidate generating density $q(x, y)$
- If a candidate value is rejected, the current value is taken as the next item in the sequence
- The calculation of $\alpha(x, y)$ does not require the knowledge of normalizing constant $\pi(\cdot)$
- If the candidate generating density is symmetric, the probability of the move reduces to $\pi(y)/\pi(x)$

4.1. Psuedocode

1. Initialize the chain with $x^{(0)}$
2. Repeat for $j = 1, 2, \ldots, N$
3. Generate y from $q(x^{(j)}, \cdot)$ and u from $U(0, 1)$
4. If $u \leq \alpha(x^{(j)}, y)$ set $x^{(j+1)} = y$. Else set $x^{(j+1)} = x^{(j)}$
5. Return the values $\{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\}$

The draws are regarded as a sample from the target density $\pi(x)$ only after the chain has passed the transient stage and the effect of the fixed starting value has become so small that it can be ignored. The regularity conditions are irreducibility and aperiodicity. One has to take care of following questions while running MCMC

- How large an initial size of the sample should be discarded ?
- How long the chain should be run ?
- Are the trace plots looking ok ?
- Is there autocorrelation amongst the samples ?
- Is the chain sticky ?
- Is the chain showing a periodicity pattern ?

5. Implementation Issues

To implement M-H algorithm, it is necessary to specify a suitable candidate-generating density. Typically this density is selected from a family of distributions that require the specification of tuning parameters such as location and scale. Some of the choices that can be made are :

1. Metropolis Random walk with uniform increments

$$q(x, y) = q_1(y - x)$$

   where $q_1(\cdot)$ is a multivariate uniform
2. Metropolis Random walk with normal increments

$$q(x, y) = q_1(y - x)$$

   where $q_1(\cdot)$ is a multivariate normal
3. Exploit the form of $\pi(\cdot)$ in specifying candidate-generating density
4. Use A-R method with *pseudo*dominating density
5. Use autoregressive chains. These chains can be used to induce negative autocorrelation in the chain

$$q(x, y) = q_1(y - a - B(x - a))$$

   This essentially means that the new value and existing value are related by $y = a + B(x - a) + z$

The spread of the candidate generating density affects the behavior of the chain in at least two dimensions : one is the "acceptance rate" and the other is the region of the sample space that is covered by the chain. Research done in this aspect has given rise to some pointers. If the target and proposal densities are normal, then the scale of the latter should be tuned so that the acceptance rate is approximately 0.45 in one dimension and 0.23 in high dimension space.

6. Applications of M-H Algorithm

This section talks about two uses of the algorithm, one involving the A-R method and the other for implementing the algorithm with block-at-a-time scans.

6.1. An M-H Acceptance-Rejection Algorithm

The A-R algo works well if one can find a blanketing function. In practice, find a blanketing function in high dimensions results in low acceptance rate. What is needed is an algo that works with local blanket covering function. The idea is that if the current state is x and the new value is y, there can be four cases :

- $x \in C, y \in C$
- $x \in C, y \notin C$
- $x \notin C, y \in C$
- $x \notin C, y \in C$

where C is the set where domination occurs. In each of the above cases, the probability distribution of an accepted value is analysed and subsequently adjusted so that reversibility condition is satisfied.

Pseudocode

1. Generate y from the AR algorithm
2. Let $C1 = \{f(x) < ch(x)\}$ and $C2 = \{f(y) < ch(y)\}$
3. Generate u from $U(0,1)$ and
   - If $C1 = 1$, then $\alpha = 1$
   - If $C1 = 0$ and $C2 = 1$, then let $\alpha = ch(x)/f(x)$
   - If $C1 = 0$ and $C2 = 0$, then let $\alpha = \min(f(y)h(x)/f(x)h(y), 1)$
4. If $u \leq q$ return y else return x

6.2. Block-at-a-Time Algorithms

The basic idea behind these algorithms is this :

- Divide the parameters in to blocks.
- Obtain the conditional transition kernels for each block so that they converge conditional invariant distributions
- Use the transition kernels for the blocks to generating samples from the joint distribution.
- The product of conditional transition kernels converges to the invariant distribution of the entire chain.

A special case of this algo is "Gibbs sampling method". This method requires that it be possible to generate independent samples from each of the full conditional densities. Another special case of M-H algo is to use general form of MH for a few parameters and use Gibbs for the rest of the parameters. Some call this method, "M-H within Gibbs".

7. Examples

The paper mentions two examples. One relates to simulating bivariate normals and second relates to bayesian analysis of AR(2) process. I have written `R` code to replicate the results mentioned in the paper

7.1. Simulating a Bivariate Normal

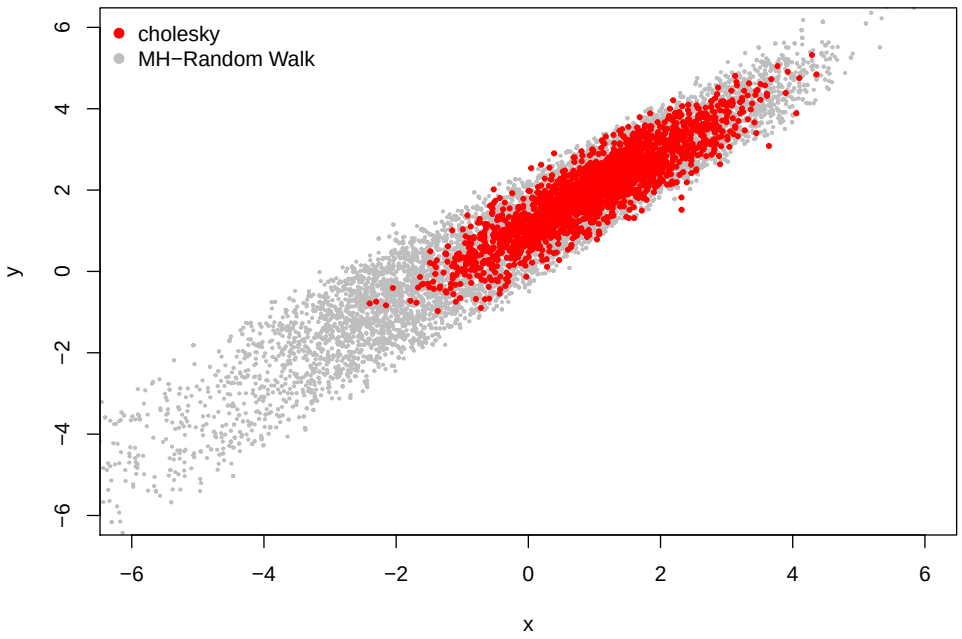
```
mu       <- c(1,2)
cov_mat  <- matrix(c(1, 0.9, 0.9, 1), 2, 2, TRUE)
set.seed(1)
sample_chol  <- rmvnorm(2000, mu, cov_mat)
```

7.1.1. Method 1 : MH Random walk with uniform increments

```
alpha        <- function(x, y) {
  num        <- exp(-0.5 * ((y-mu)%*% solve(cov_mat)%*%(y-mu)))
  den        <- exp(-0.5 * ((x-mu)%*% solve(cov_mat)%*%(x-mu)))
  min(num/den, 1)
}
sample_val    <- function(x) {
```

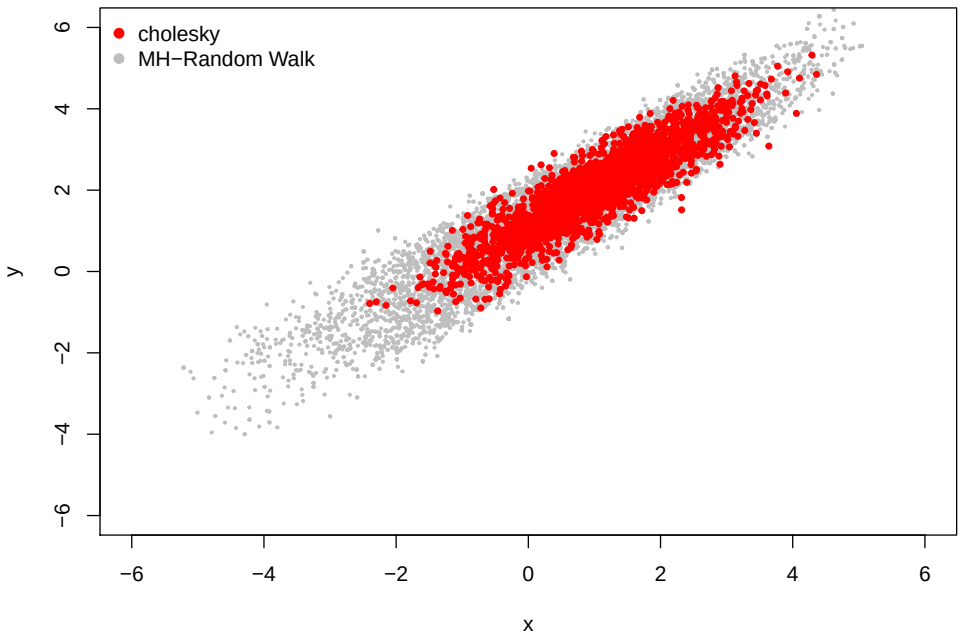
```
  delta_1     <- 0.75
  delta_2     <- 1
  return( x + c(runif(1, -delta_1, delta_1), runif(1, -delta_2, delta_2)))
}

N             <- 10000
x_start       <- 0
results_1     <- matrix(NA, nrow = N, ncol = 2)
results_1[1,] <- x_start
i             <- 2
for( i in 2:N ) {
  new_val     <- sample_val(results_1[i - 1, ])
  alpha_x_y   <- alpha(results_1[i - 1], new_val)
  if(runif(1) < alpha_x_y){
    results_1[i, ] <- new_val
  }else{
    results_1[i,]  <- results_1[i-1, ]
  }
}
results_1     <- results_1[round(0.1*N):N,]
```



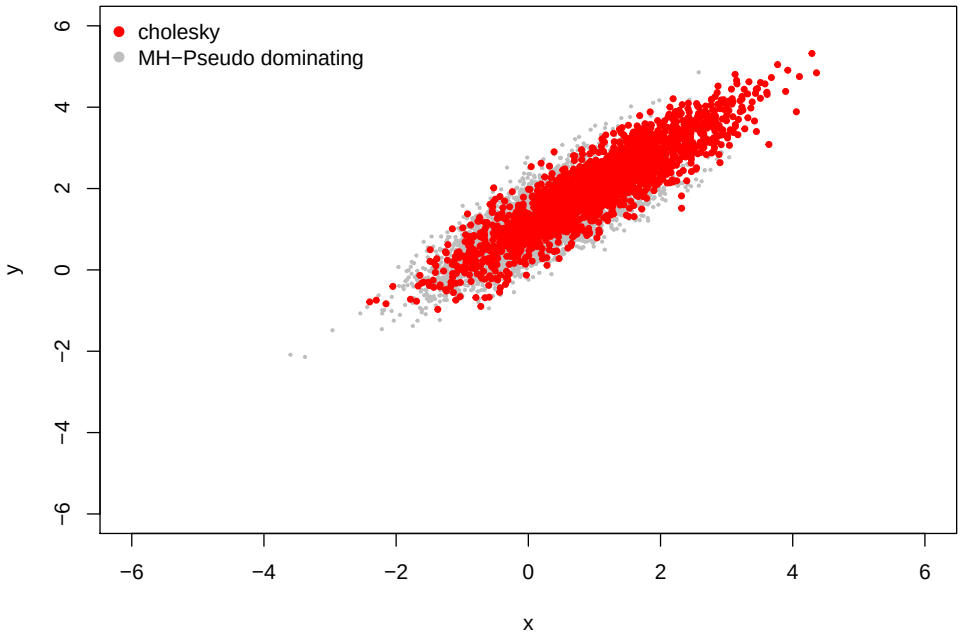
7.1.2. Method 2 : MH Random walk with normal increments

```
sample_val    <- function(x) {
  delta_1     <- 0.6
  delta_2     <- 0.4
  return(x + c(rnorm(1, 0, sqrt(delta_1)), rnorm(1, 0, sqrt(delta_2))))
}
results_2     <- matrix(NA, nrow = N, ncol = 2)
results_2[1,] <- x_start
i             <- 2
for( i in 2:N ) {
  new_val     <- sample_val(results_2[i - 1, ])
  alpha_x_y   <- alpha(results_2[i - 1], new_val)
  if(runif(1) < alpha_x_y){
    results_2[i, ] <- new_val
  }else{
    results_2[i,]  <- results_2[i-1, ]
  }
}
results_2     <- results_2[round(0.1*N):N,]
```



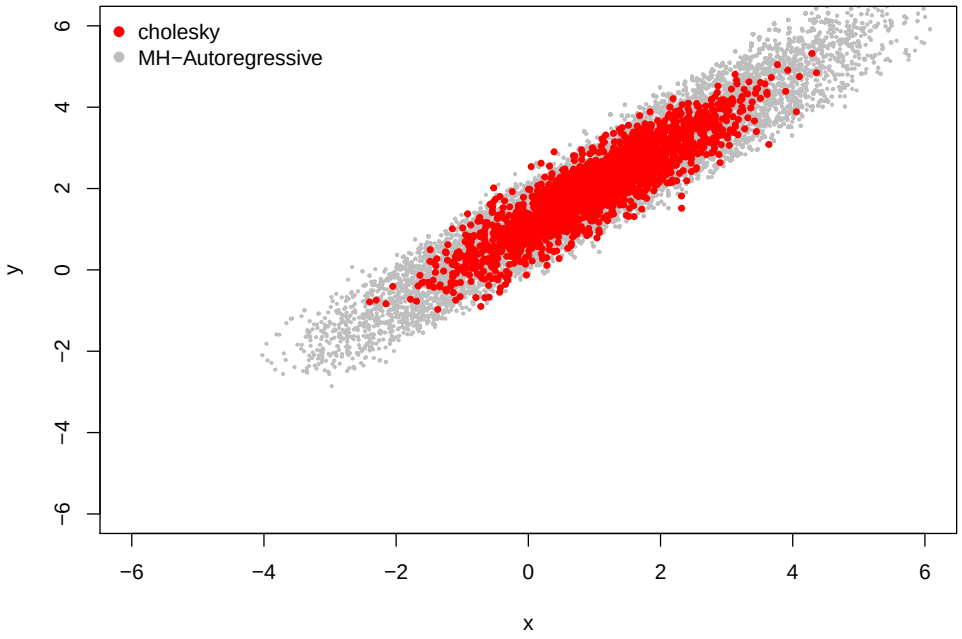
7.1.3. Method 3 : Psuedo Dominating Density method

```
D             <- diag(c(2, 2))
k             <- 0.9
f             <- function(z) 1/(2*pi)*1/sqrt(det(cov_mat)) *
                  exp(-0.5*(z-mu)%*%solve(cov_mat)%*%(z-mu))
h             <- function(z) dmvnorm(z, c(0, 0), D)
results_3     <- matrix(NA, nrow = N, ncol = 2)
results_3[1,] <- x_start
i             <- 2
for(i in 2:N){
  cur_val      <- results_3[i - 1, ]
  accept       <- FALSE
  while(accept == FALSE){
    z          <- c(rmvnorm(1, c(0, 0), D))
    if(runif(1) <= f(z)/(k*(h(z)))) {
      new_val   <- z
      accept    <- TRUE
    }
  }
  C1           <- f(cur_val) < k*h(cur_val)
  C2           <- f(new_val) < k*h(new_val)
  if(C1 == 1)          alpha_mult <- 1
  if(C1 == 0 & C2 ==1) alpha_mult <- k*h(cur_val)/f(cur_val)
  if(C1 == 1 & C2 ==0) alpha_mult <- min(f(new_val)*h(cur_val)/(f(cur_val)*h(new_val)),1)
  if(runif(1) <= alpha_mult){
    results_3[i, ] <- new_val
  }else{
    results_3[i,]  <- results_3[i-1, ]
  }
}
results_3     <- results_3[round(0.1*N):N,]
```



Method 4 : MH Random walk with autoregressive chains

```
sample_val    <- function(x) {
  delta_1     <- 0.6
  delta_2     <- 0.4
  return(2*mu - x + c(runif(1, -delta_1, delta_1), runif(1, -delta_2, delta_2)))
}
results_4     <- matrix(NA, nrow = N, ncol = 2)
results_4[1,] <- x_start
i             <- 2
for( i in 2:N ) {
  new_val     <- sample_val(results_4[i - 1, ])
  alpha_x_y   <- alpha(results_4[i - 1], new_val)
  if(runif(1) < alpha_x_y){
    results_4[i, ] <- new_val
  }else{
    results_4[i,]  <- results_4[i-1, ]
  }
}
results_4     <- results_4[round(0.1*N):N,]
```



7.1.4. Autocorrelation of the chain across four methods

```
corr_chain <- function(temp){
  n        <- dim(temp)[1]
  cor(temp[1:(n-1)], temp[2:n])
}
cor_results <- data.frame( "Method 1" = corr_chain(results_1),
                           "Method 2" = corr_chain(results_2),
                           "Method 3" = corr_chain(results_3),
                           "Method 4" = corr_chain(results_4))
cor_results <- t(cor_results)
colnames(cor_results) <- c("$\\rho$")
```

	ρ
Method.1	0.99
Method.2	0.96
Method.3	0.75
Method.4	-0.47

7.2. Simulating a Bayesian Posterior for AR(2)

```
set.seed(1)
n           <- 100
ar2_data    <- arima.sim( n = n, list( order=c(2,0,0), ar = c(1,-0.5)), sd = 1)

Y           <- ar2_data[1:2]
N           <- 5000
params      <- matrix(NA, nrow = N, ncol = 3)
y           <- cbind(ar2_data[3:n])
wt          <- cbind(ar2_data[2:(n-1)], ar2_data[1:(n-2)])
V_inv_mat   <- function(phi){
  matrix( c( 1-phi[2]^2, -phi[1]*(1+phi[2]),
             -phi[1]*(1+phi[2]),1-phi[2]^2) , 2, 2, T)
}
Psi         <- function(phi, sig2) {
  V_inv     <- V_inv_mat(phi)
  1/sig2 *1 /sqrt(abs(det(V_inv))) * exp(-1/(2*sig2) * t(Y)%*%V_inv%*%Y)
}
params[1,]  <- c(2, 0.5, 0.2)
i           <- 2
for(i in 2:N) {
  #simulate gamma rv
  cur_phi     <- params[(i-1), 2:3]
  V_inv       <- V_inv_mat(cur_phi)
  params[i,1] <- 1/rgamma(1, n/2, 0.5*(Y%*%V_inv%*%Y + sum((y - wt%*%cur_phi)^2)))

  G           <- t(wt)%*%wt
  phi_mu      <- solve(G) %*%(t(wt)%*%y)
  phi_sig     <- params[i,1]*solve(G)
  new_phi     <- c(rmvnorm(1, phi_mu, phi_sig))
  alpha_mult  <- min(Psi(new_phi, params[i, 1])/ Psi(cur_phi, params[i, 1]), 1)
  if(runif(1) < alpha_mult){
    params[i, c(2,3)] <- new_phi
  }else{
    params[i, c(2,3)] <- cur_phi
  }
}
params       <- params[round(0.1*N):N,]
params       <- cbind(params, sqrt(params[,1]))
results      <- apply(params,2, function(z){
c(mean(z), sd(z), median(z), quantile(z, prob=c(0.025, 0.975)),
  cor(z[2:4501],z[1:4500]))
```

```
})
results       <- t(results)
colnames(results) <- c("mean","sd", "median", "lower", "upper","correlation")
results           <- as.data.frame(results)
results$"o.b.m sd"  <- sqrt(c(apply(params, 2,function(z) olbm(z, 50))))
results           <- results[-1,]
rownames(results) <- c("$\\phi_1$","$\\phi_2$","$\\sigma$")
results <- results[,c("mean", "o.b.m sd", "sd",
                      "median", "lower", "upper","correlation")]
```

	mean	o.b.m sd	sd	median	lower	upper	correlation
ϕ_1	1.09	0.0017	0.09	1.09	0.92	1.26	0.27
ϕ_2	-0.59	0.0019	0.09	-0.59	-0.76	-0.42	0.27
σ	0.89	0.0010	0.06	0.89	0.78	1.03	0.01

7.3. Simulating from a Truncated Normal

This section uses MH algo to simulate samples from a truncated normal distribution.
Consider $f(x) = N(5, 9)I(1 \leq x \leq 6)$

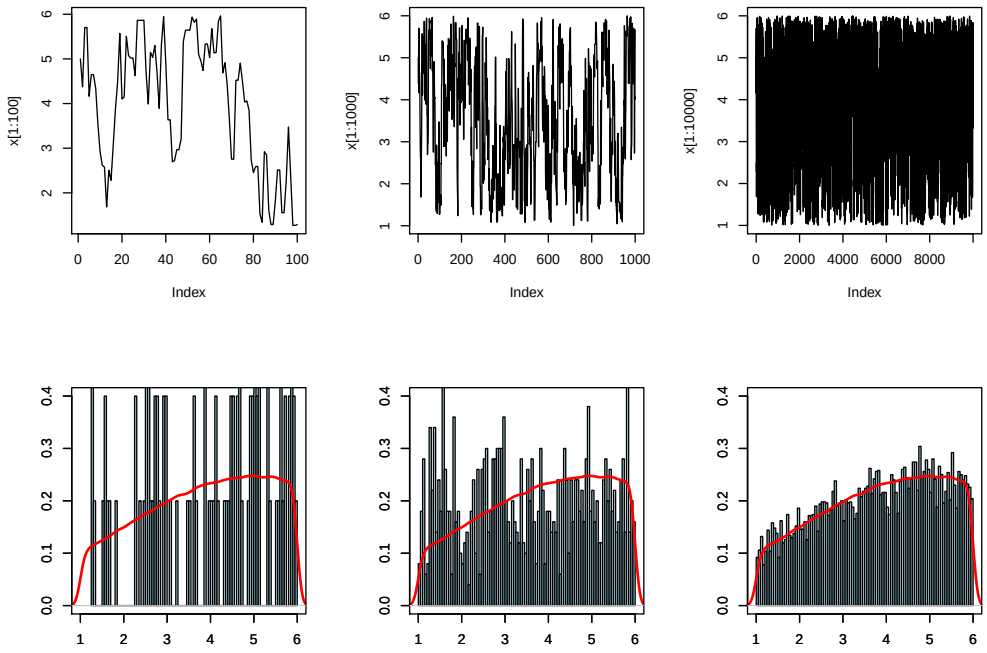
$$f(x) \propto \exp(-(x-5)^2/18)I(1 \leq x \leq 6)$$

and let $x^{(0)} = 5$ and let proposal distribution be $q(x|x^{(j)}) = N(x|x^{(j)}, 1)$ Sample sizes of 100, 1000, 10000 are generated and density plots are superimposed so that one can visually check for distribution fit.

7.3.1. Random Walk Metropolis

```
mh_simulation <- function(N){
prop          <- function(x, y) exp(-(x-y)^2/2)
prop_sample   <- function(x) rnorm(1,x,1)
targ          <- function(x) exp(-(x-5)^2/18)*ifelse(all(x>=1,x<=6),1,1e-10)
x_0           <- 5
results       <- numeric(N)
results[1]    <- x_0
i             <- 2
set.seed(1)
for(i in 2:N){
  current     <- results[i-1]
  x_star      <- prop_sample(current)
  rho         <- min(1,(targ(x_star)*prop(current,x_star))/
                       (targ(current)*prop(x_star,current)))
  results[i] <- ifelse(runif(1) < rho, x_star, current)
  }
```

```
results
}
sample_1   <-  mh_simulation(100)
sample_2   <-  mh_simulation(1000)
sample_3   <-  mh_simulation(10000)
```



7.3.2. Independent Metropolis Hastings

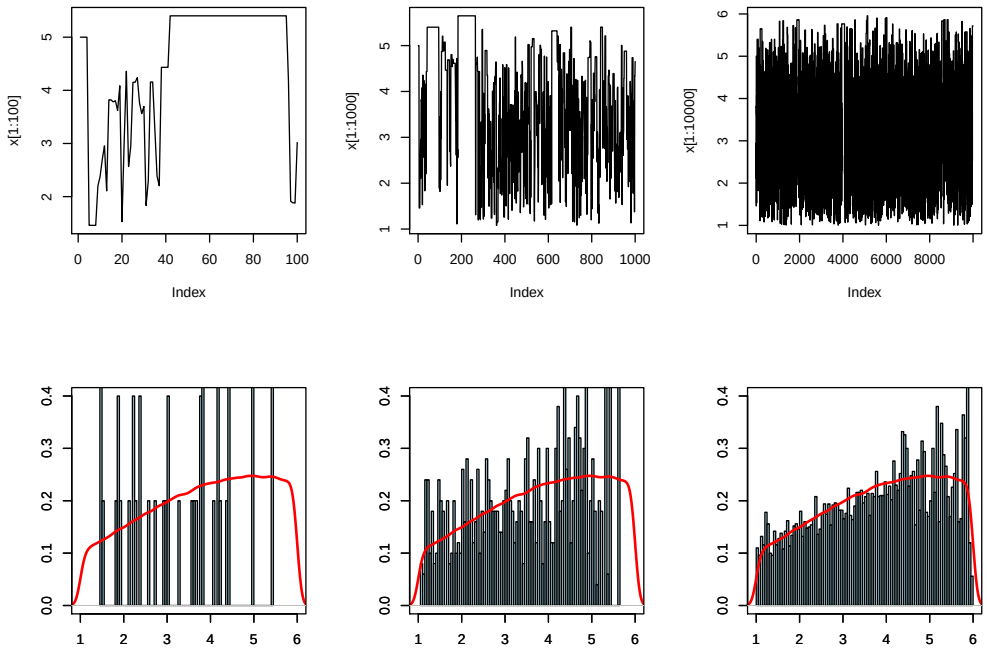
Consider $f(x) = N(5, 9)I(1 \leq x \leq 6)$

$$f(x) \propto \exp(-(x-5)^2/18)I(1 \leq x \leq 6)$$

and let $x^{(0)} = 5$ and let proposal distribution be $q(x|x^{(j)}) = N(x|3, 1)$

```
mh_simulation <- function(N){
prop          <- function(x, y) exp(-(x-3)^2/2)
prop_sample   <- function(x) rnorm(1,3,1)
targ          <- function(x) exp(-(x-5)^2/18)*ifelse(all(x>=1,x<=6),1,1e-10)
x_0           <- 5
results       <- numeric(N)
results[1]    <- x_0
i             <- 2
set.seed(1)
for(i in 2:N){
  current     <- results[i-1]
```

```
  x_star     <- prop_sample(current)
  rho        <- min(1,(targ(x_star)*prop(current,x_star))/
                      (targ(current)*prop(x_star,current)))
  results[i] <- ifelse(runif(1) < rho, x_star, current)
  }
results
}
sample_1   <-  mh_simulation(100)
sample_2   <-  mh_simulation(1000)
sample_3   <-  mh_simulation(10000)
```



8. Takeaway

The paper is wonderfully written. It illustrates the workings of Metropolis Hastings algorithm from scratch. The basic idea of searching for a transition kernel is avoided by tweaking a generic transition kernel so that detailed balance conditions hold good. Thus the algorithm is nothing but an extremely smart extension of Accept-Reject method.