Bayesian Statistics X

The Metropolis Hastings Algorithm

Introduction

The Metropolis Hastings algorithm is a beautifully simple algorithm for producing samples from distributions that may otherwise be difficult to sample from.

Suppose we want to sample from a distribution π, which we will call the "target" distribution. For simplicity we assume that π is a one-dimensional distribution on the real line, although it is easy to extend to more than one dimension (see below).

The MH algorithm works by simulating a Markov Chain, whose stationary distribution is π. This means that, in the long run, the samples from the Markov chain look like the samples from π. As we will see, the algorithm is incredibly simple and flexible. Its main limitation is that, for difficult problems, "in the long run" may mean after a *very* long time. However, for simple problems the algorithm can work well.

The MH algorithm

The transition kernel

To implement the MH algorithm, the user (you!) must provide a "transition kernel", Q. A transition kernel is simply a way of moving, randomly, to a new position in space (y say), given a current position (x say). That is, Q is a distribution on y given x, and we will write it $Q(y|x)$. In many applications Q will be a continuous distribution, in which case $Q(y|x)$ will be a density on y, and so $\int Q(y|x)dy = 1$ (for all x).

For example, a very simple way to generate a new position y from a current position x is to add an $N(0,1)$ random number to x. That is, set $y = x + N(0,1)$, or $y|x \sim N(x,1)$. So

$$Q(y|x) = 1/\sqrt{2\pi}\exp[-0.5(y-x)^2].$$

This kind of kernel, which adds some random number to the current position x to obtain y, is often used in practice and is called a "random walk" kernel.

Because of the role Q plays in the MH algorithm (see below), it is also sometimes called the "proposal distribution". And the example given above would be called a "random walk proposal".

The MH algorithm

The MH algorithm for sampling from a target distribution π, using transition kernel Q, consists of the following steps:

- Initialize, $X_1 = x_1$ say.
- For $t = 1, 2, \dots$
  - sample y from $Q(y|x_t)$. Think of y as a "proposed" value for x_{t+1}.
  - Compute
    $$A = \min\left(1, \frac{\pi(y)Q(x_t|y)}{\pi(x_t)Q(y|x_t)}\right).$$
    A is often called the "acceptance probabilty".
  - with probability A "accept" the proposed value, and set $x_{t+1} = y$. Otherwise set $x_{t+1} = x_t$.

The Metropolis Algorithm

Notice that the example random walk proposal Q given above satisfies $Q(y|x) = Q(x|y)$ for all x, y. Any proposal that satisfies this is called "symmetric". When Q is symmetric the formula for A in the MH algorithm simplifies to:

$$A = \min\left(1, \frac{\pi(y)}{\pi(x_t)}\right).$$

This special case of the algorithm, with Q symmetric, was first presented by Metropolis et al, 1953, and for this reason it is sometimes called the "Metropolis algorithm".

In 1970 Hastings presented the more general version – now known as the MH algorithm – which allows that Q may be assymmetric. Specifically Hastings modified the acceptance probability by introducing the term $Q(x_t|y)/Q(y|x_t)$. This ratio is sometimes called the "Hastings ratio".

Toy Example

To help understand the MH algorithm we now do a simple example: we implement the algorithm to sample from an exponential distribution:

$$\pi(x) = \exp(-x) \qquad (x \geq 0).$$

Of course it would be much easier to sample from an exponential distribution in other ways; we are just using this to illustrate the algorithm.

Remember that π is called the "target" distribution, so we call our function to compute π `target`:

```
target = function(x){
  return(ifelse(x<0,0,exp(-x)))
}
```

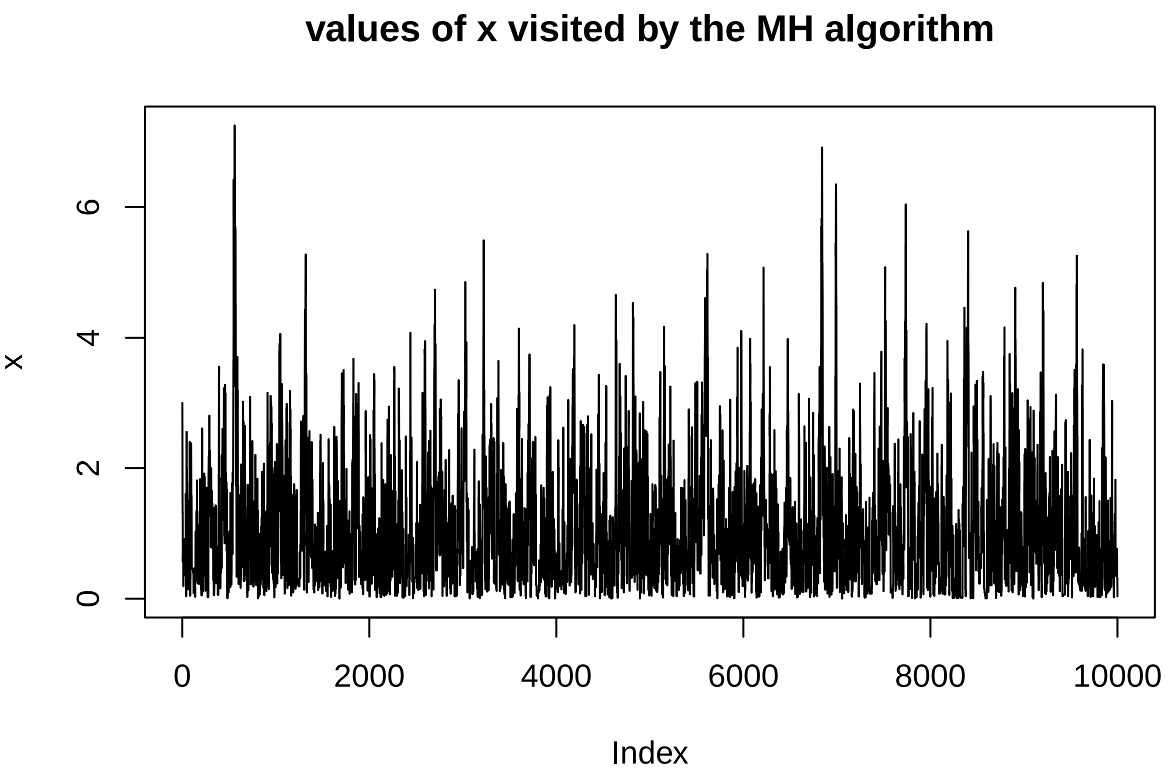
Now we implement the MH algorithm, using the simple normal random walk transition kernel Q mentioned above. Since this Q is symmetric the Hastings ratio is 1, and we get the simpler form for the acceptance probability A in the Metropolis algorithm.

Here is the code:

```
x = rep(0,10000)
x[1] = 3     #initialize; I've set arbitrarily set this to 3
for(i in 2:10000){
  current_x = x[i-1]
  proposed_x = current_x + rnorm(1,mean=0,sd=1)
  A = target(proposed_x)/target(current_x)
  if(runif(1)<A){
    x[i] = proposed_x       # accept move with probabily min(1,A)
  } else {
    x[i] = current_x        # otherwise "reject" move, and stay where we are
  }
}
```

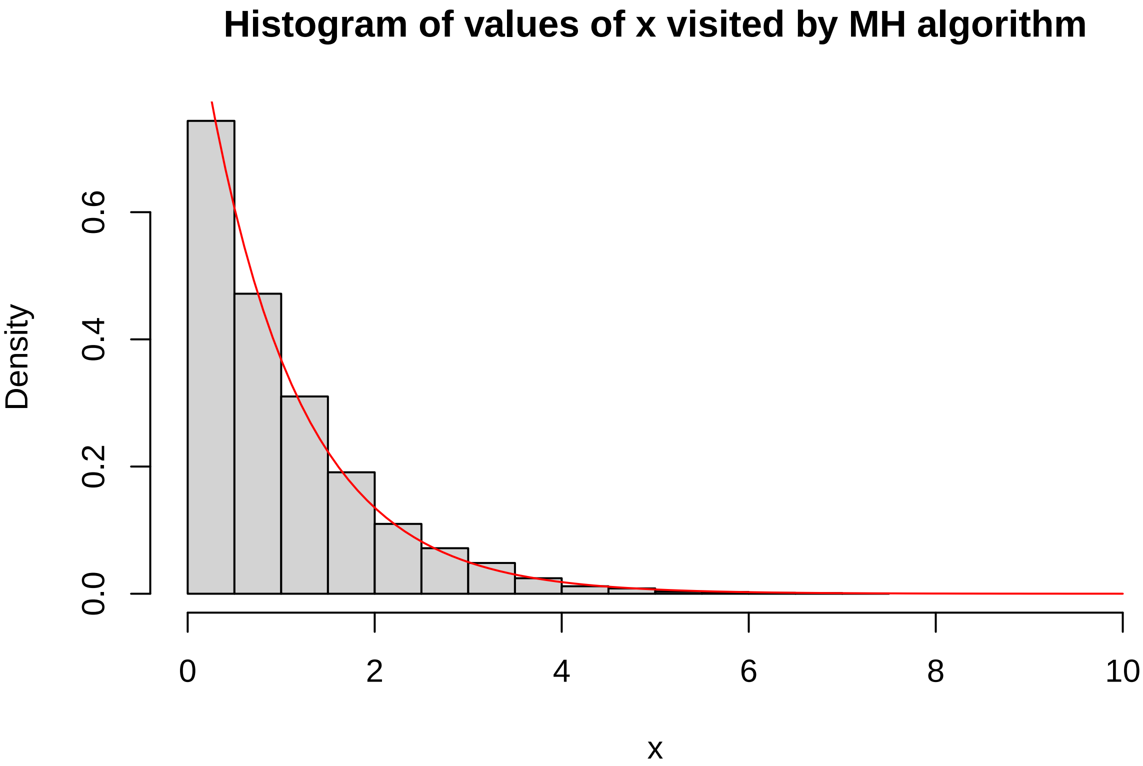
Having run this code we can plot the locations visited by the Markov chain x (sometimes called a trace plot).

```
plot(x,main="values of x visited by the MH algorithm", type = 'l')
```



Remember that we designed this algorithm to sample from an exponential distribution. This means that (provided we ran the algorithm for long enough!) the histogram of x should look like an exponential distribution. Here we check this:

```r
hist(x,xlim=c(0,10),probability = TRUE, main="Histogram of values of x visited by MH algorithm")
xx = seq(0,10,length=100)
lines(xx,target(xx),col="red")
```



So we see that, indeed, the histogram of the values in x indeed provides a close fit to an exponential distribution.

Closing remarks

One particularly useful feature of the MH algorithm is that it can be implemented even when π is known only up to a constant: that is, $\pi(x) = cf(x)$ for some known f, but unknown constant c. This is because the algorithm depends on π only through the ratio $\pi(y)/\pi(x_t) = cf(y)/cf(x_t) = f(y)/f(x_t)$.

This issue arises in Bayesian applications, where the posterior distribution is proportional to the prior times the likelihood, but the constant of proportionality is often unknown. So the MH algorithm is particularly useful for sampling from posterior distributions to perform analytically-intractible Bayesian calculations.