Markov Chain Monte Carlo

Oliver J. Maclaren | oliver.maclaren@auckland.ac.nz

26 May 2016

MCMC - the basic idea

The problem of generating samples

It is surprisingly difficult to 'draw samples' from a given distribution, even when it has a known probability mass/density function.

For example - how would you draw a sequence of observations such that each outcome occurs with probability given by a Normal distribution?

What about a Cauchy distribution

$$f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$$

In each case *we can calculate the probability (or probability density) of a given observation but we struggle to generate observations occurring with a given probabilities.* The difference is subtle but important!

The solution

In general, no randomness in means no randomness out. What we can (and typically have to) do is use simple known distributions - the most fundamental is probably the *uniform* distribution - to give us some 'elemental randomness' (or deterministic chaos!) and then 'leverage' this to *create randomness of a desired sort.*

We will look at generating Normally distributed samples using the uniform distribution as our basic 'random element' (without directly using the CLT and in such as way that we could also use the procedure for e.g. the Cauchy distribution above).

The key idea of the MCMC approach is to *construct a Markov process* - which here will be a modified simple random walk - *which has a limiting distribution* equal to our (known) *target distribution*, which we will call $f(x)$.

After the chain converges to its limiting distribution, the subsequent realisations $X_i, X_{i+1}, ...$ etc *will be samples from our target distribution.*

Simple MCMC - Metropolis-Hastings algorithm

Note - there are a number of variations of this algorithm. Here we will first look at one *type* of Metropolis-Hastings (MH) algorithm called a 'random-walk chain' or 'random-walk-Metropolis-Hastings'. I will try to point out how you might construct other variations on MH. There are also a number of MCMC methods which use different approaches to MH (though quite a few can in fact be reformulated as a MH-type algorithm).

Starting point

Here is the basic starting point for the scheme

- Choose a starting state (sample) X_0 arbitrarily
- Suppose we have generated samples $X_0, X_1, ...X_i$ for some $i \geq 0$

Generating new candidates

We now want to generate X_{i+1} in such a way that it only depends on X_i (even if this dependence is complicated!).

To do this we first need a way of *randomly proposing new candidates.* This is typically called the *proposal distribution* and denoted by the conditional probability density/mass function $q(y|x)$ which we can evaluate for any x and y.

Note that this is *simply a transition matrix in the discrete case.* Here we are also allowing ourselves to work with a continuous state space if we want, so are going a bit beyond what we've covered in the previous lectures. It all works basically the same, however.

We indicate the process of drawing a sample from this distribution given X_i by $Y \sim q(y|X_i)$. This corresponds to *generating a new sample according to our (proposal) transition probabilities.* Note that the notation is slightly ambiguous - i.e. used to mean both 'drawing a sample' vs 'is distributed as' - but is standard.

It is important that $q(y|X_i)$ is nice and simple to sample from - otherwise we are back to square one! We are using q's 'simple, elementary randomness' to generate more 'complicated, targeted randomness' (effectively, add more structure back).

Luckily, *in principle* it doesn't matter (other than a few minor details) which distribution we use for q, though *in practice* it may make a big difference to e.g. convergence rates. Hence we can (again, in principle) use something simple like a uniform random number generator.

Random walk chains

A *random walk* chain works by simply adding a *random error term* to get our candidate, where the 'random error' is centred at zero and which may be positive or negative. That is we propose X_{i+1} candidates, which we denote by Y, according to

$$Y = X_i + \epsilon$$

where ϵ has a known distribution. In this case each ϵ is sometimes called an *innovation* - simply because it represents a the *difference* between the new candidate sample and the old sample.

For example we might take $\epsilon \sim U(-\alpha, \alpha)$ for some α range (scale parameter) that must typically be set manually (and is therefore part art). More common would be $\epsilon \sim N(0, \sigma)$ where σ is now a scale parameter that must be set manually, but here we are assuming we don't know how to generate Normally distributed samples.

This is clearly *just a random walk* (and a Markov chain).

Exercise: write down a simple discrete example indicating what the (proposal) transition matrix would look like (assuming we 'accept' any proposal generated).

Note that in this case it needs to make sense for X_n to have a state space corresponding to arbitrarily positive and negative values since we could in principle wander away to infinity. We could e.g. transform to a new scale using logs or implement 'boundary conditions' etc but we will just ignore this subtlety here.

Accepting or rejecting proposed candiates p.I

If we just accepted every randomly generated candidate we would have a simple random walk. *We want our random walk to be more 'directed' than this, however.*

In particular, we want its limiting distribution to correspond to our *target* distribution $f(x)$. This will ensure that, once our chain converges to its limiting distribution, each sample (or individual realisation) will correspond to a sample from our desired probability distribution.

So now we introduce an additional wrinkle. Rather than just accept Y straightaway, we associate a 'score' $r(X_i, Y)$, called the *acceptance probability*, designed to *compare which of the candidate Y and the current X_i are more likely to be a sample from our desired distribution $f(x)$*. Note that the score also depends on our known $f(x)$ - i.e. it needs to know which samples are 'likely' samples from $f(x)$. We'll discuss this more below.

The score is constructed so as to be a probability and hence we need to ensure $0 \leq r(X_i, Y) \leq 1$. Furthermore, it will be constructed so that it is closer to one when Y is 'better' (more likely to be a sample from our target) relative to the current X_i and closer to zero if X_i is 'better' than Y.

Important note: r will be constructed so that it depends only on the X_i and Y (and Y itself depends on the state X_i, i.e. not on X_{i-1} etc). Hence r is also a valid 'current-state' quantity which we can use to construct a Markov chain/process.

In particular, we can use X_i, our (known) innovation distribution for ϵ and our (known, but not yet specified, score $r(X_i, Y)$) to construct a *new* Markov process

$$X_{i+1} = \begin{cases} Y, & \text{with probability } r \\ X_i, & \text{with probability } 1 - r \end{cases}$$

With the right choice of r and a known probability density/mass function $f(x)$ we wish to sample from we can show that the limiting distribution of X_i exists, is unique and converges to the distribution given by $f(x)$. Hence our sequence of observed states $X_i, X_{i+1}, ...$ will (eventually) correspond to samples from our target distribution.

Note on accepting with given probability

Suppose we have a desired probability p and we want to accept (or get a one, say) with probability p and reject (or get a zero, say) with probability $1 - p$.

First note that a uniformly-distributed variable $u \sim U(0, 1)$ will have $u < p$ with probability p since the cumulative distribution function $F(x)$ is $\frac{x-0}{1-0} = x$ and so $F(p) = p$.

Then the following procedure clearly gives 'acceptance' with probability p:

- Generate a *new* uniformly distributed variable $u \sim U(0, 1)$.
- Accept if $u < p$, reject otherwise.

Accepting or rejecting proposed candiates p.II

We have already chosen a sub-type of the general Metropolis-Hastings scheme, based on modifying a simple random walk.

Now we consider a further specialisation, *based on choosing a particular type of proposal distribution.*

In particular, it can be shown that if we choose a *symmetric proposal distrbution* - which corresponds to choosing symmetric 'innovations' or 'errors' about zero - then a sufficient choice (i.e. sufficient to ensure that our chain has limiting distribution = target distribution) for $r(X_i, Y)$ is

$$r = \min\left\{\frac{f(y)}{f(x)}, 1\right\}$$

Note that the main part is simply a comparison of the relative probabilities of being a sample from f for y and x, while the 'min' part ensures we don't exceed 1. When we choose a symmetric proposal distribution the algorithm is typically referred to as a Metropolis algorithm (no Hastings).

There is a tradeoff in choosing good proposals - the scale of the error - such that we get good r values.

- If we choose a scale such that we almost always accept - i.e. r is typically ≈ 1 - then this typically corresponds to taking very small steps (small innovations) along good directions and hence not exploring enough space to ensure our chain 'settles' down to its equilibrium.
- If we choose a scale such that we almost always reject - i.e. r is typically ≈ 0 - then this typically corresponds to taking wild guesses (large innovations), rarely stepping but taking large steps when we do. This also means we don't typically settle down to equilibrium well.

A good goal is typically to get an r around 30-50%. Luckily, we have the acceptance probabilities for each run and so can use this as a diagnostic tool.

Let's code one up!

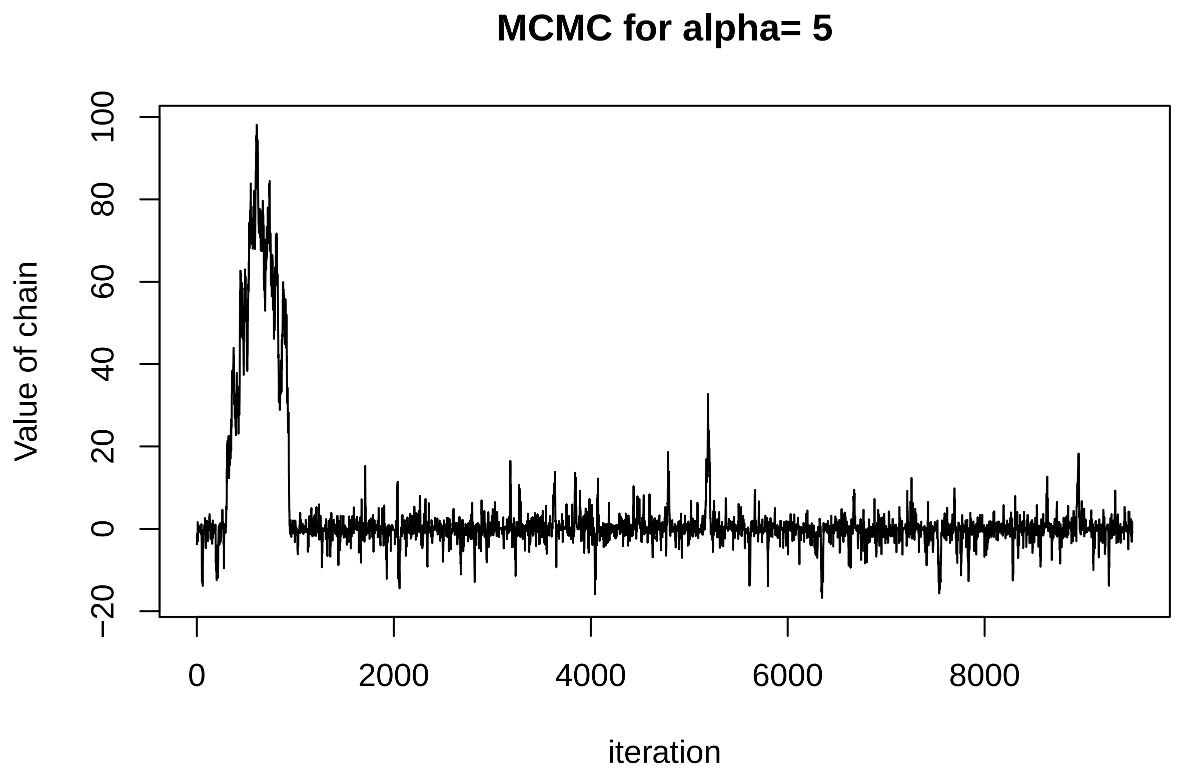
Here is an R code block implementing a simple random-walk MH algorithm.

```
#choice of target distribution from which we wish
#to generate samples.
target <- function(x) {
  #t = dnorm(x)
  t = dcauchy(x)
  #t = dexp(x)
  #t = (1/pi)*1/(1+x^2) #a roll-your-own Cauchy!
  return(t)
}

#Random-walk MH algorithm
alpha = 5 #'innovation' scale - try running for 0.001,0.01,0.1, 1.0, 10, 100, 1000...
n = 10000 #number of samples/steps to take
chain = vector("numeric",n)
scores = vector("numeric",n)
x = 0 #initial value
chain[1] = x
for (i in 2:n) {
  candidate = x + runif(1,-alpha,alpha)
  score = target(candidate)/target(x) #see target above.
  scores[i] = score #for diagnostic checking
  #accept with probability = score (for score <=1).
  #if score > 1 then always accepts - we're good either way!
  u = runif(1)
  if (u < score )
    x = candidate
  chain[i] = x #score next value of chain
}
```

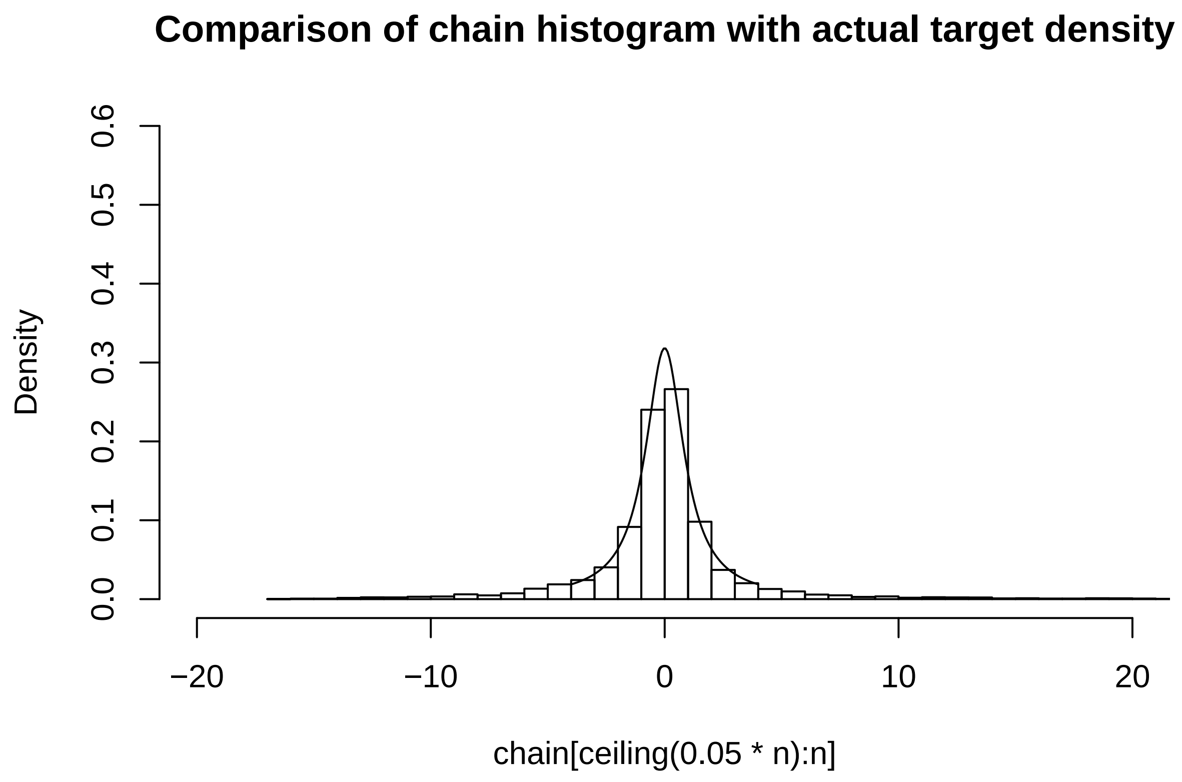
```
#diagnostics - acceptance prob
mean(pmin(scores,1))
```

```
## [1] 0.5027634
```

```
#plot(pmin(scores,1),type="l", xlab="iteration",ylab="acceptance prob.",
#         main=paste("Acceptance prob. for alpha=",
# alpha))

#plots of chain
plot(chain[ceiling(0.05*n):n],type="l", xlab="iteration",ylab="Value of chain",
        main=paste("MCMC for alpha=",
                   alpha))
```



```
hist(chain[ceiling(0.05*n):n],breaks = 100,freq=FALSE,main="Comparison of chain histogram with actual ta
#compare with actual target
x <- seq(-4, 4, length=100)
#hx <- dnorm(x)
hx <- dcauchy(x)
lines(x,hx,ylim=c(0,1))
```


Comparison of chain histogram with actual target density
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−20
−10
0
10
20
chain[ceiling(0.05 * n):n]