Introduction to MCMC

Helen Johnson

June 7, 2014

1 Markov Chain Monte Carlo (MCMC) inference

1.1 Recap. on Bayesian inference

We saw above that the joint probability

$$p(y, x, \theta) = p(y|x, \theta)p(x|\theta)p(\theta)$$

encodes everything about our model.

For model fitting and inference, y is the observed data and our aim is to calculate the *posterior* probabilities

$$p(x, \theta|y) = p(\theta|y)p(x|\theta, y)$$

In this section, we introduce **Monte Carlo approximation methods**. The main idea is to generate *unweighted samples* from the posterior distribution and to use these to compute any quantity of interest e.g. .

The more samples we generate, the more accurate our estimates will be. The challenges lie in how to:

- generate samples from a probability distribution
- effectively cover parameter space (often high-dimensional)
- know when to stop!

A particularly good reference on sampling is Robert and Casella [**?**].

Often the function we want to learn about (e.g. $P(x) = p(\theta|y)$ in the case of parameter fitting)is complicated, making it difficult to fit a standard distribution such as a Gaussian and sample from it directly. So we apply instead *Monte Carlo* methods (so-called because they make use of random numbers, like dice throws). A simple Monte Carlo method is **rejection sampling**.

1.2 Rejection sampling

We assume that the distribution of interest $P(x) = P^*(x)/Z$ (where Z is a normalisation constant) is too difficult to sample from directly.

Our first step is to identify a **proposal distribution** $Q(x)$, which we can evaluate and from which we are able to draw samples. We then assume that we can scale up the proposal distribution by a constant c such that

$$cQ^*(x) \geqslant P^*(x)$$

for all x. That is, that it provides an upper envelope for the distribution from which we want to sample.

Insert figure here
Rejection sampling proceeds as follows:

- Sample $x^* \sim Q(x)$.
  (This is equivalent to choosing a random location on the x-axis.)
- Pick another random number $u \sim Uniform[0, cQ^*(x)]$
  (This is equivalent to choosing a random location on the y-axis.)
- Evaluate $P^*(x^*)$
- If $P^*(x^*) < u$ accept x^*
  (This means we add x^* to our set of samples)
- Otherwise reject x^*.
- Discard u
- Repeat

Rejection sampling works best if $Q(x)$ is a good approximation to $P(x)$. If Q and P are very different shapes then c needs to be very large to ensure that $cQ^*(x)$ exceeds $P(x)$ everywhere. In this case the rejection rate becomes very high.

In higher-dimensional problems e.g. when trying to infer several parameters, this limitation becomes even more troublesome. The requirement for $cQ^*(\mathbf{x}) \geqslant P^*(\mathbf{x})$ will force c to be so huge that very few of our samples will ever be accepted. So rejection sampling is not really a practical option for building up a picture of a high-dimensional distribution $P(\mathbf{x})$. This leads us on to **Markov Chain Monte Carlo**.

1.3 What is Markov Chain Monte Carlo?

The concept underlying Markov Chain Monte Carlo (MCMC) is to perform a random walk around the state space $\mathcal{X}$ in such a way that the amount of time we spend in each state $\mathbf{x}$ is proportional to $P(\mathbf{x})$.

This random walk should have the properties of a **Markov chain**. That is, that knowing where we are should be sufficient for choosing the next state, without having to look back at all the previous states we took to get here.

Interlude: Markov chains

Consider a sequence of observations, $X_1, ..., X_T$ of aribtrary length T. These could be points in time or in (state-)space. The idea of a Markov chain is to assume that X_t contains all the necessary information to predict the future, without reference to the past. Thus the joint distribution is written as:

$$P(X_{1:T}) = P(X_1)P(X_2|X_1)P(X_3|X_2)... = P(X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})$$

If $P(X_t|X_{t-1})$ is independent of time then we say the chain is **stationary**.

We are generally interested in the long-term distribution over all states, this is called the **stationary distribution** and it occurs when the chain has forgotten its original state and found an equilibrium (think of walking around in a very complicated loop!).

Two algorithms are widely used for constructing the Markov chain: **Metropolis-Hastings** and **Gibbs sampling**. We shall focus here on the former but see the Bibliography for good references for further reading.

1.4 Metropolis-Hastings

The greatest limitation of rejection sampling is the need to identify a proposal distribution $Q(\mathbf{x})$ that matches the distribution of interest $P(\mathbf{x})$ closely. In high dimensions and with complex problems it is difficult to find a single density to fulfil this requirement.

The Metropolis-Hastings (M-H) algorithm circumvents the need to find one proposal density $Q(\mathbf{x})$ to cover the whole of $P(\mathbf{x})$. Instead it makes use of a proposal density that depends on the current state $\mathbf{x}$.

In each step, we propose to move from where we are now $\mathbf{x}$ to another state

$\mathbf{x}'$ with probability $Q(\mathbf{x}'|\mathbf{x})$, where Q is again the proposal distribution. Q can be any fixed distribution from which we can take samples - this makes the M-H algorithm very flexible. We discuss the choice of proposal distribution below.

The algorithm proceeds as follows:

- Initialise x_0
  for *s = 0,1,2,...* **do**
- Define $x = x_s$;
- Sample $x' \sim Q(x'|x)$;
- Compute acceptance probability
$$\alpha = \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}$$
- Compute $r = min(1, \alpha)$;
- Sample $u \sim U(0, 1)$;
- Set new sample to
$$x_{s+1} = \begin{cases} x', & \text{if } u < r \\ x_s, & \text{if } u \geqslant r \end{cases}$$

Clearly an important component of this algorithm is the formula for deciding whether a certain proposed step should be taken. If the proposal distribution is symmetric then the acceptance probability is given by the formula:

$$r = min(1, \frac{P(\mathbf{x}')}{P(\mathbf{x})})$$

In this case we will definitely move to $\mathbf{x}'$ if it is more probable than $\mathbf{x}$ (since $\frac{P(\mathbf{x}')}{P(\mathbf{x})} > 1$) but if $\mathbf{x}'$ is less probable then we may still move there anyway, depending on the relative probabilities and the random number we draw.

If the proposal is asymmetric i.e. $Q(x|x') \neq Q(x'|x)$ then we need to apply the so-called 'Hastings correction', given by:

$$\frac{P(x')/Q(x'|x)}{P(x)/Q(x|x')} = \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}$$

This compensates for the fact that the proposal distribution itself might favour certain states.

A particular strength of the M-H algorithm is that we do not need to know the normalised version of our target density - any normalisation constant is factored out in the ratio step.

1.5 Proposal distributions

Any type of distribution can potentially be used as a proposal distribution; the only requirement is that it allow for the possibility of any state which may be included in the target distribution. A **multivariate Gaussian** distribution is a popular choice since it:

- has a non-zero probability density on the whole state space
- is relatively easy to manipulate numerically

However, in order for MCMC to be accurate and efficient, it is important that the proposal distribution has its probability mass distributed appropriately. The key to success is in **tuning** the variance and covariance terms.

Figure X shows an example of two distinct modes - a particularly tricky problem.

If the variance term is too small, then the chain will never be able to move far enough from one mode to be able to identify the other (Figure Xa).

If the variance is too high, then many proposed values will be rejected and the chain will *stick* in one place for many steps (Figure Xb).

If the variance is just right, then the chain will efficiently explore the full shape of the target distribution.

1.6 Choosing a proposal distribution

We saw above that key to the accuracy and efficiency of an MCMC method is the choice of proposal distribution. We need to ensure that we explore the whole of the target distribution but we don't want to try too many unsuccessful steps.

In practice, we tend to try several different proposal distributions, aiming for an acceptance rate between 24% and 40% . (Theory suggests that for a one-dimensional problem 50% is optimal). These initial test are called **pilot runs**.

The most efficient proposals depend not only on the current state but also on the observed data. This is called **data-driven MCMC**. See Tu and Zhu (2002).

A handy trick to improve efficiency is to change the parameters of the proposal distribution while the chain is running - this is known as **adaptive MCMC**. Thus we can start, for example, with a broad covariance, which allows us to scan around the state-space, until we find a mode. At this point we can tighten up the covariance, making the proposal distribution more specific, and explore the region around the mode more carefully.

Figures here showing size and shape

The main caveat is that the parameters of the proposal distribution should not depend on the whole history of the chain as this would violate the *Markov property*. To avoid this, we introduce a **cooling** term, meaning that the parameters only 'remember' a certain number of steps.

1.7 Burn-in

MCMC is started from an arbitrary point, but it is only once we reach the **stationary distribution** that we are sampling from $P(\mathbf{x})$. We generally throw away samples made before reaching this equilibrium in what is commonly known as the **burn-in** phase.

The amount of time it takes to reach the stationary distribution is called the **mixing time** and can be difficult to assess. This is one of the limitations of MCMC. There are computational methods to calculate it exactly, using properties of the covariance matrix, but these are tricky.

One simple heuristic method is to start several chains off in different parts of state-space (these are called **over-dispersed**) and superimpose their output on one plot. If their sample traces overlap, then we assume they have converged to the stationary distribution.

1.8 Accuracy of MCMC

By its very nature, MCMC produces auto-correlated samples. This mean that we need more of them than we would do if we were drawing independent samples from our target distribution.

We can reduce the degree of auto-correlation in our samples by **thinning** and only retain every n^{th} sample. This doesn't reduce the computational time but it does mean that we don't need to store many highly-correlated sampled.

The information content of a set of MCMC samples is given by the **effective sample size (ESS)**, which is defined as the Monte Carlo estimate of the variance if the samples weren't correlated, divided by the variance in the MCMC samples. A handy R package for thinning samples and calculating the ESS is *coda*.

Some thought has been given to the advantage of running several short chains, rather than one long one but the jury is still out. It seems there's no strong evidence that one is preferable to the other.

1.9 MCMC vocabulary

- **chain:**
- **mixing:**
- **burn in:**
- **tuning:**
- **stationary distribution:**
- **proposal distribution:**
- **target distribution:**
- **acceptance rate:**