Markov Chain Monte Carlo

Felipe José Bravo Márquez

July 1, 2021

Markov Chain Monte Carlo

- This class introduces estimation of posterior probability distributions using a stochastic process known as **Markov chain Monte Carlo** (MCMC).
- Here we'll produce samples from the joint posterior without maximizing anything.
- We will be able to sample directly from the posterior without assuming a Gaussian, or any other, shape.
- The cost of this power is that it may take much longer for our estimation to complete.
- But the benefit is escaping multivariate normality assumption of the Laplace approximation.
- More advanced models such as the generalized linear and multilevel models tend produce non-Gaussian posterior distributions.
- In most cases they cannot be estimated at all with the techniques of earlier classes.
- This class is based on Chapter 9 of [McElreath, 2020] and Chapter 7 of [Kruschke, 2014].

Markov Chain Monte Carlo

- The essence of MCMC is to produce samples from the posterior $f(\theta|d)$ by only accessing a function that is proportial to it.
- This proportial function is the product of the likelihood and the prior $f(d|\theta) * f(\theta)$, which is always available in a Bayesian model.
- So, merely by evaluating $f(d|\theta) * f(\theta)$, without normalizing it by $f(d)$, MCMC allows us to generate random representative values from the posterior distribution.
- This property is wonderful because the method obviates direct computation of the evidence $f(d)$, which, as you'll recall, is one of the most difficult aspects of Bayesian inference.
- It has only been with the development of MCMC algorithms an software that Bayesian inference is applicable to complex data analysis.
- And it has only been with the production of fast and cheap computer hardware that Bayesian inference is accessible to a wide audience.
- The question then becomes this: How does MCMC work? For an answer, let's ask a politician.

A politician stumbles upon the Metropolis algorithm

- Suppose an elected politician lives on a long chain of islands.
- He is constantly traveling from island to island, wanting to stay in the public eye.
- At the end of a day he has to decide whether to:

    1. stay on the current island
    2. move to the adjacent island to the west
    3. move to the adjacent island to east

- His goal is to visit all the islands **proportionally** to their **relative population**.
- But, he doesn't know the total population of all the islands.
- He only knows the population of the current island where he is located.
- He can also ask about the population of an adjacent island to which he plans to move.

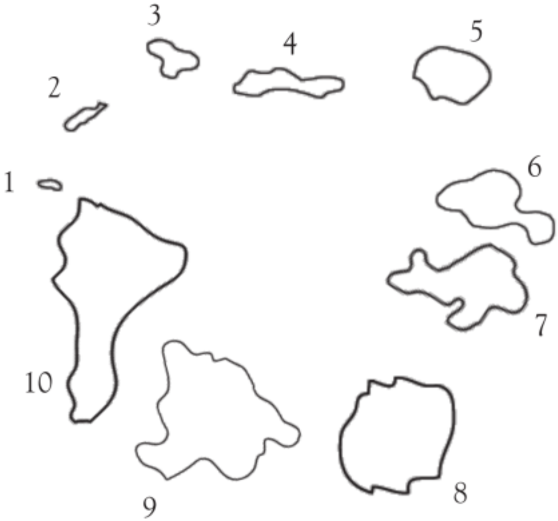
The Metropolis algorithm

- The politician has a simple heuristic for travelling accross the islands called the **Metropolis** algorithm [Metropolis et al., 1953].
- First, he flips a (fair) coin to decide whether to propose the adjacent island to the left or the adjacent island to the right.
- If the proposed island has a larger population than the current island ($P_{proposed} > P_{current}$), then he goes to the proposed island.
- If the proposed island has a smaller population than the current island ($P_{proposed} < P_{current}$), then he goes to the proposed island with probability $p_{move} = P_{proposed}/P_{current}$.
- In the long run, the probability that the politician is on any one of the islands exactly matches the relative population of the island!

The Metropolis algorithm

- Let’s analyze the Metropolis algorithm in more detail.
- Suppose there are 10 islands in total.
- Each island is neighbored by two others, and the entire archipelago forms a ring.
- The islands are of different sizes, and so had different sized populations living on them.
- The second island is twice as populous as the first, the third three times as populous as the first.
- And so on, up to the largest island, which is 10 times as populous as the smallest.

The Metropolis algorithm



The Metropolis algorithm

- We are going to show an implementation of this algorithm in R.
- But before that, we will combine combine the two possibilities for the probability of moving into a single expression: the proposed island having a 1) higher or 2) lower population than the current island.

$$p_{move} = \min(1, P_{proposed}/P_{current}). \tag{1}$$

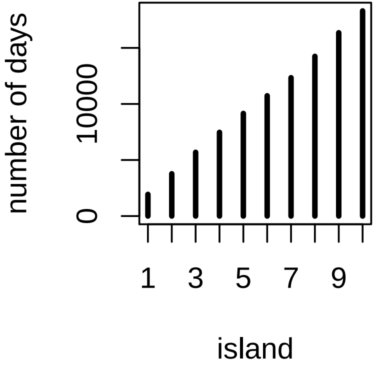
- So, if $P_{proposed} > P_{current}$, $P_{proposed}/P_{current} > 1$ and $p_{move} = 1$.
- For example, $current = 4$ and $proposed = 5$, $5/4 > 1$ so we move to the proposed island (with probability 1).
- On the other hand, if $P_{proposed} < P_{current}$, $P_{proposed}/P_{current} < 1$, and $p_{move} = P_{proposed}/P_{current}$.
- For example, $current = 4$ and $proposed = 3$, $3/4 < 1$ so we move to the proposed island with probability $3/4$.

The Metropolis algorithm

```
num_days <- 1e5
positions <- rep(0,num_days)
current <- 10
for ( i in 1:num_days ) {
  # record current position
  positions[i] <- current
  # flip coin to generate proposal
  proposal <- current + sample( c(-1,1) , size=1 )
  # now make sure he loops around the archipelago
  if ( proposal < 1 ) proposal <- 10
  if ( proposal > 10 ) proposal <- 1
  # move?
  prob_move <- min(proposal/current,1)
  decision <- rbinom(1,1,prob_move)
  current <- ifelse( decision == 1 , proposal , current )
}

library(rethinking)
simplehist(positions,xlab="island",ylab="number of days")
```

The Metropolis algorithm



The time spent on each island is proportional to its population size.

The Metropolis algorithm

- The first three lines of the method just define the number of days to simulate, an empty history vector, and a starting island position (the biggest island, number 10).
- Then the for loop steps through the days.
- Each day, it records the politician's current position.
- Then it simulates a coin flip to nominate a proposal island.
- The only trick here lies in making sure that a proposal of "11" loops around to island 1 and a proposal of "0" loops around to island 10.
- Finally, a random binary number is generated with a binomial distribution with probability of success (or moving)$= \min(1, P_{proposed}/P_{current})$.
- If this random number is 1 we move, otherwise we stay.

The Metropolis algorithm

- In real applications, the goal is not to help a politician, but instead to draw samples from an unknown and usually complex posterior probability distribution.
- The "islands" in our objective are parameter values θ, and they need not be discrete, but can instead take on a continuous range of values as usual.
- The "population sizes" in our objective are the posterior probabilities (or densities) at each parameter value: $f(\theta|d)$
- The "days" in our objective are samples taken from the posterior distribution.
- The Metropolis algorithm will eventually give us a collection of samples from the posterior.
- We can then use these samples just like all the samples we have already used in this course.

Why it works

- Now, let's try to understand why the algorithm works.
- Consider two adjacent positions and the probabilities of moving from one to the other.
- We'll see that the relative transition probabilities, between adjacent positions, exactly match the relative values of the target distribution.
- Extrapolate that result across all the positions, and you can see that, in the long run, each position will be visited proportionally to its target value.
- Suppose we are at position θ.
- The probability of moving to $\theta + 1$, denoted $P(\theta \rightarrow \theta + 1)$, is the probability of proposing that move times the probability of accepting it if proposed, which is:

$$P(\theta \rightarrow \theta + 1) = 0.5 \times \min(P(\theta + 1)/P(\theta), 1)$$

Why it works

- On the other hand, if we are presently at position $\theta + 1$, the probability of moving to θ is:

$$P(\theta + 1 \to \theta) = 0.5 \times \min(P(\theta)/P(\theta + 1), 1)$$

- The ratio of the transition probabilities is:

$$\begin{aligned}
\frac{p(\theta \to \theta+1)}{p(\theta+1 \to \theta)} &= \frac{0.5 \min\left(P(\theta+1)/P(\theta), 1\right)}{0.5 \min\left(P(\theta)/P(\theta+1), 1\right)} \\
&= \begin{cases} \frac{1}{P(\theta)/P(\theta+1)} & \text{if } P(\theta + 1) > P(\theta) \\ \frac{P(\theta+1)/P(\theta)}{1} & \text{if } P(\theta + 1) < P(\theta) \end{cases} \\
&= \frac{P(\theta+1)}{P(\theta)}
\end{aligned}$$

Why it works

- The last equation tells us that during transitions back and forth between adjacent positions, the relative probability of the transitions exactly matches the relative values of the target distribution.
- That might be enough to get the intuition that, in the long run, adjacent positions will be visited proportionally to their relative values in the target distribution.
- If that's true for adjacent positions, then, by extrapolating from one position to the next, it must be true for the whole range of positions.
- In more mathematical terms, this means that the transition probabilities form a Markov chain that has the target distribution as its equilibrium or stationary distribution. [Wikipedia, 2021]
- Hence, one can obtain a sample of the desired distribution by recording states from the chain.

The Metropolis Algorithm more Generally

- So far, we have only considered the case with a single discrete parameter θ that can only move to the left or right.
- The general Metropolis algorithm allows working with multiple continuous parameters $\theta_1, \theta_2, \ldots, \theta_n$ and more general proposal distributions.
- The essentials of the general method are the same as for the simple case.
- First, we have some target distribution $P(\theta)$ (θ can be a vector of parameters) from which we would like to generate representative sample values.
- We must be able to compute the value of $P(\theta)$ for any candidate value of θ.
- The distribution, $P(\theta)$, does not have to be normalized, however.
- Just needs needs to be nonnegative.

The Metropolis Algorithm more Generally

- In our Bayesian inference application $P(\theta)$ is the unnormalized posterior distribution on θ, which is the product of the likelihood and the prior: $f(d|\theta) * f(\theta)$.
- This is a very important property of MCMC, as it allows us to draw samples from the posterior without having to calculate the evidence $f(d)$.
- Sample values from the target distribution are generated by taking a random walk through the parameter space.
- Proposal distributions can take many different forms, the goal being to use a proposal distribution that efficiently explores the regions of the parameter space where $P(\theta)$ has most of its probability area.
- The generic case is using a Gaussian distribution centered at the current position.
- So the proposed move will typically be near the current position, with the probability of proposing a more distant position dropping off according to the normal curve.
- For multivariate target distributions, we can use a Multi-variate Gaussian to propose multi-dimensional points in each step.

Gibbs Sampling

- The Metropolis algorithm works whenever the probability of proposing a jump to B from A is equal to the probability of proposing A from B, when the proposal distribution is symmetric (such as a Gaussian distribution).
- There is a more general method, known as Metropolis-Hastings, that allows asymmetric proposals.
- This would mean, that the politician's coin were biased to lead him clockwise on average.
- Asymmetric proposal distributions allows us to explore the posterior distribution more efficiently (i.e., acquire a good image of the posterior distribution in fewer steps).
- Gibbs sampling is a variant of the Metropolis-Hastings algorithm that uses clever proposals and is therefore more efficient.
- The improvement arises from adaptive proposals in which the distribution of proposed parameter values adjusts itself intelligently, depending upon the parameter values at the moment.

Gibbs Sampling

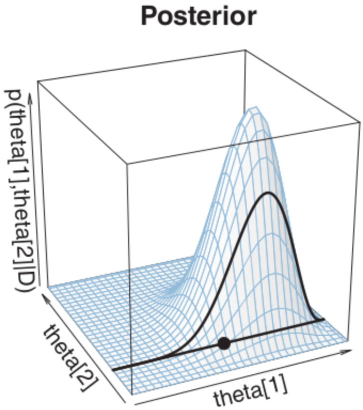
- How Gibbs sampling computes these adaptive proposals depends upon using particular combinations of prior distributions and likelihoods known as conjugate pairs (such as Binomial and the Beta).
- Conjugate pairs have analytical solutions for the posterior distribution of an individual parameter.
- And these solutions are what allow Gibbs sampling to make smart jumps around the joint posterior distribution of all parameters.
- The algorithm works as follows:
- At each point in the walk, the parameters are selected in an iterative cycle: $\theta_1, \theta_2, \theta_3, \dots \theta_1, \theta_2, \theta_3, \dots$.
- Suppose that parameter θ_i has been selected.
- Gibbs sampling then chooses a new value for that parameter by generating a random value directly from the conditional probability distribution of that parameter given all the others and d:

$$f(\theta_i|\theta_1, \dots, \theta_{i-1}, \theta_{i+1}, \dots, \theta_n, d)$$

Gibbs Sampling

- As we are using conjugate pairs, this conditional probabilities distribution has a closed form and is easy to sample random numbers from it.
- The new value for θ_i , combined with the unchanged values of $\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n$, constitutes the new position in the random walk.
- The process then repeats: select the next parameter θ_{i+1} and select a new value for that parameter from its conditional posterior distribution.
- Let's illustrate this process for a two-parameter example: θ_1, θ_2.
- In the first step, we want to select a new value for θ_1 .
- We conditionalize on the values of all the other parameters from the previous step in the chain.
- In this example, there is only one other parameter, namely θ_2.

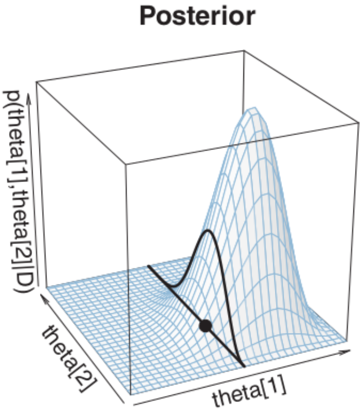
Gibbs Sampling



- The figure shows a slice through the joint distribution at the current value of θ_2.
- The heavy curve is the posterior distribution conditional on this value of θ_2, which is $f(\theta_1|\theta_2, d)$ in this case because there is only one other parameter.

Gibbs Sampling

- Because we are using conjugate pairs a computer can directly generate a random value of θ_1 from $f(\theta_1|\theta_2, d)$.
- Having generated a new value for θ_1, we then conditionalize on it and determine the conditional distribution of the next parameter, θ_2 using $f(\theta_2|\theta_1, d)$ as shown below:



- We generate a new value of θ_2 , and the cycle repeats.

Gibbs Sampling

- Because the proposal distribution exactly mirrors the posterior probability for that parameter, the proposed move is always accepted.
- Hence, the algorithm is more efficient than the standard Metropolis algorithm in which proposals are rejected in many cases.
- But there are some limitations to Gibbs sampling.
- First, there are cases when we don't want to use conjugate priors.
- Second, it can become inefficient with complex models containing hundreds, thousands or tens of thousands of parameters.

Conclusions

- Blabla

References I

- Kruschke, J. (2014).
  Doing bayesian data analysis: A tutorial with r, jags, and stan.

- McElreath, R. (2020).
  Statistical rethinking: A Bayesian course with examples in R and Stan.
  CRC press.

- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953).
  Equation of state calculations by fast computing machines.
  The journal of chemical physics, 21(6):1087–1092.

- Wikipedia (2021).
  Markov chain Monte Carlo — Wikipedia, the free encyclopedia.
  `http://en.wikipedia.org/w/index.php?title=Markov%20chain%20Monte%20Carlo&oldid=1027048003`.
  [Online; accessed 01-July-2021].