Introduction to MCMC using RevBayes

Wade Dismukes, Tracy Heath, Walker Pett

1 Overview

This tutorial is intended to provide a introduction to the basics of Markov chain Monte Caro (MCMC) using the Metropolis-Hastings algorithm. This will provide a brief introduction to MCMC moves as well as prior distributions. We begin with a simple example of estimating the probability distribution of an archer's ability to shoot at a target, and the distance those arrows land from the center. We will simulate data using this example and attempt to estimate the posterior distribution using a variety of MCMC moves.

1.1 Learning Outcomes

- Understand and implement the Metropolis-Hastings MCMC algorithm
- Begin to develop an intuition regarding the use of different priors
- Understand the difference and utility of various MCMC moves

1.2 Required Software

This tutorial requires that you download and install the latest release of `RevBayes` (`?`), which is available for Mac OS X, Windows, and Linux operating systems. Directions for downloading and installing the software are available on the program webpage: http://revbayes.com.

The exercise provided also requires additional programs for editing text files and visualizing output. The following are very useful tools for working with `RevBayes`:

- A good text editor – if you do not already have one that you like, we recommend one that has features for syntax coloring, easy navigation between different files, line numbers, etc. Good options include `Sublime Text` or Atom, which are available for Mac OSX, Windows, and Linux.
- `Tracer` – for visualizing and assessing numerical parameter samples from `RevBayes`

2 Introduction

3 Modeling an archer's shots on a target

Suppose you are interested in estimating the distribution of an archer's shots on a target. For simplicity we will assume that these n shots are all taken from 10 meters away from the target by the same experienced archer. One way that we could do this is to measure how far each arrow lands from the bullseye. Let's assume that these distances follow a truncated normal distribution with some mean, μ, and some standard deviation, σ, and a minimum of 0 meters, and a maximum of 300 meters. For Bayesian inference it is often more convinient to reparameterize the standard deviation as $\tau = \frac{1}{\sigma^2}$. The following represents the data model for n arrows shot where a is a data point and Φ is the cumulative distribution function of the standard normal distribution:

$$p(a_i|\mu,\tau,0,300) = \prod_{i=1}^{n} \frac{\frac{\sqrt{\tau}}{\sqrt{2\pi}} exp\left(-\frac{1}{2}\tau(a_i-\mu)^2\right)}{(\Phi(300)-\Phi(0))}$$

Now suppose we don't know what the mean is, but we do know the standard deviation for simplicity. Now we need to place a prior on the mean that describes our uncertainty about the mean. For now let's use another truncated normal distribution and so our prior becomes:

$$p(\mu|\mu_0,\sigma_0,0,300) = \frac{\frac{\sqrt{\tau_0}}{\sqrt{2\pi}} exp\left(-\frac{1}{2}\tau_0(\mu-\mu_0)^2\right)}{(\Phi(300)-\Phi(0))}$$

where μ_0 and σ_0 are set by the scientist. The values of these should reflect one's belief about the parameter that the prior distribution distribution. For example, if we were very confident then we would set both μ_0 and σ_0 to a small numbers. Intuitively, it may be useful to think of the μ_0 as your guess about the parameter and σ_0 as your confidence in that guess. Using Bayes' Rule we can then derive the posterior:

$$p(\mu|y,\sigma) = \frac{\frac{\sqrt{\tau_n}}{\sqrt{2\pi}} exp\left(-\frac{1}{2}\tau_n(\mu-\mu_n)^2\right)}{(\Phi(300)-\Phi(0))}$$

with

$$\mu_n = \tau_n^{-1}(n\tau\bar{a} + \sigma_0\mu_0)$$

$$\tau_n = \tau_0 + n\tau$$

$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

As you can see the posterior distribution has the same distribution family as the prior. When this happens the prior is called the conjugate prior distribution. In addition, the normal prior for this model has the attractive property that its posterior precision is the sum of the prior mean and the data mean, and the posterior mean is the precision weighted average of the prior mean and the data mean.

3.1 Tutorial Format

This tutorial follows a specific format for issuing instructions and information.

The boxed instructions guide you to complete tasks that are not part of the RevBayes syntax, but rather direct you to create directories or files or similar.

Information describing the commands and instructions will be written in paragraph-form before or after they are issued.

All command-line text, including all Rev syntax, are given in **monotype font**. Furthermore, blocks of Rev code that are needed to build the model, specify the analysis, or execute the run are given in separate shaded boxes. For example, we will instruct you to create a constant node called **rho** that is equal to 1.0 using the <- operator like this:

```
rho <- 1.0 # constant node
alpha ~ dnExponential(rho) # stochastic node
lambda := 1 / alpha # deterministic node
```

It is important to be aware that some PDF viewers may render some characters given as `Rev commands` differently. Thus, if you copy and paste text from this PDF, you may introduce some incorrect characters. Because of this, we recommend that you type the instructions in this tutorial or copy them from the scripts provided.

3.2 Data and Files

On your own computer or your remote machine, create a directory called RB_MyFirstMCMC_Tutorial (or any name you like).

In this tutorial we will be simulating our own data using RevBayes. Explain how to simulate data in Rev. We will be simulating data. Let's assume from the above archery example that our archer's true ability has their arrows landing with a mean of 0 and a variance of 1. Let's say they shoot six arrows. We do this in RevBayes like this:

```
minimum = 0
maximum = 300
num_arrows = 6
true_mu = 0.0
true_var = 1

arrows = rnormal(n = num_arrows, mean = true_mu, sd = true_sd, min = minimum, max =
    maximum)
```

3.3 Getting Started with MCMC

As you saw above, we were able to get a nice analytical posterior distribution for our normal model with known variance and a normal prior on the mean even in the case of a truncated normal. However, in the majority of interesting scientific cases we are unable to do this. This is where MCMC comes in. MCMC algorithms are a set of tools that provide us with a means to estimate the posterior even in very complex scenarios. There are many flavors of MCMC algorithms; however, this tutorial will only cover the Metropolis-Hastings algorithm. The basic idea of this algorithm is to propose new values for our parameter, θ, in some way (here they are termed moves) Then based on our data model and prior we calculate what is called the Hastings ratio:

$$R = \frac{p(y|\theta')p(\theta')}{p(y|\theta)p(\theta)} \times \frac{q(\theta|\theta')}{q(\theta'|\theta)}$$

where θ' represents the new proposed value of the parameter and θ represents the previous iteration's value. $q(\theta'|\theta)$ is called the proposal distribution. However, oftentimes it is convenient to choose our proposals such that $q(\theta'|\theta) = q(\theta|\theta')$ thus eliminating that term from the calculation of R, moves that have this property are called symmetric moves. The Hastings ratio is then used to determine whether the proposed value, θ' is an improvement over the previous iterations value.

The Metropolis-Hastings algorithm is stated as follows:

1. Set an initial value for θ
2. Draw a new value, θ' from $q(\theta'|\theta)$
3. Calculate the Hastings ratio
4. Draw u from $Unif(0, 1)$
5. If $u < R$ then we set $\theta = \theta'$
6. Record the value of θ and go to 2. Repeating N times

3.3.1 Metropolis-Hastings algorithm by hand

Go through outline of algorithm steps. explain proposal distributions. Then go into a step-by-step on how to write a MH-algorithm in Rev. First write the functions for the likelihood and the prior.

Likelihood function:

```
function Natural logLikelihood(mu_prime, sd_zero, mini, maxi){
       likelihood_mean = 0.0
       for(i in 1:num_arrows){
              likelihood_mean += dnormal(arrows[i], mu_prime, sd_zero, min = mini, max
                  = maxi, log=true)
       }
       return likelihood_mean
}
```

Prior function:

```
function Natural logPriorMean(mu_prime, mu_zero, sd_zero, mini, maxi){
       prior_mean = dnormal(mu_prime, mu_zero, sd_zero, min = mini, max = maxi, log=
           true)
       return prior_mean
}
```

Draw an initial value for our mean from the prior distribution:

```
prior_mu = 0.5
stdev = 1
mu <- rnorm(1, mean = prior_mu, stdev, min = minimum, max= maximum)
```

Set number of iterations of our MCMC and setup writing our output:

```
reps = 10000
write("iteration","p","\n",file="output/archery_MH.log")
write(0,v,"\n",file="output/archery_MH.log",append=TRUE)
```

Now before we write out MCMC algorithm. We need to decide what type of proposal we are going to use on our parameter, μ. For new we will simply randomly propose new values for μ by taking our previous value of μ and adding a uniformly distributed variable to that value. The uniform distribution we use will be drawn from the range $\{-\delta, \delta\}$ where δ is set to some value. This parameter, δ, is known as a tuning parameter and can be optimized to allow for the most efficient sampling of our target distribution. Finally, we can write our Metropolis Hastings Algorithm:

```
# first we need to set delta
delta = 0.1

# the actual algorithm
for(rep in 1:reps){

       mu_prime = mu + runif(n=1, -delta, delta)[1]

       R = (logLikelihood(mu_prime, stdev, minimum, maximum) - logLikelihood(mu, stdev
           , minimum, maximum)) +
                ( logPriorMean(mu_prime, prior_mu, stdev, min = minimum, max = maximum)
                    - logPriorMean(mu, prior_mu, stdev, min = minimum, max = maximum))
       u = runif(1,0,1)[1]
       if(ln(u) < R){
              # accept proposal
```

```
            mu = mu_prime
        }

        write(rep,mu, "\n", file="output/archery_MH.log", append=TRUE)
}
```

When this finishes running you should now have a posterior distribution that looks like a truncated normal distribution.

Use `Tracer` to examine the output of your MCMC.

3.3.2 Metropolis-Hastings using RevBayes

Now imagine a new archer arrives on the range and we have no prior belief about what the mean of the distribution of their shots would be, or about the variance of that distribution. Now we need a prior on both the mean and the variance. First, we need to simulate data. Let's say that the new archer is a beginner who tends to shoot to the right of target and with quite high variance:

```
num_arrows = 6    # number of arrows shot
true_mu = 1.5    # true mean
true_var = 2.0    # true variance

arrows = rnormal(num_arrows, true_mu, true_sd, min = 0.0, max = 300)
```

Now that we have data we can precede once again with setting up our model. However, this time we will use the builtin RevBayes tools. Here we use a stochastic node to put a gamma distribution on our precision and then use a deterministic node to transform the precision into variance:

```
alpha <- 1
beta <- 1
precision ~ dnGamma(a,b)
sd := sqrt(1 / precision)

moves[1] = mvSlide(precision, delta = 0.1, tune = false, weight = 2.0)
```

Now for convience sake let's assume that the mean (conditional on our variance) follows a normal distribution similar to our data model now this prior has one parameters we need to specify, the prior mean, which we will set to 0 for simplicity, again we use a stochastic node to draw the mean from a normal distribution:

```
prior_mean <- 0.5   # mean for the prior distribution
# prior distribution
mu ~ dnNormal(prior_mean, sd, min = 0, max = 300)

# move for our mu
moves[2] = mvSlide(mu, delta = 0.1, tune = false, weight = 2.0)
```

Now set the data model and then clamp the data to that node.

```
# specify our data model and clamp the data to it
for(i in 1:num_arrows){
       shot[i] ~ dnNormal(mu, sd, min = 0, max = 300)
       shot[i].clamp(arrows[i])
}
```

Now we construct our model:

```
my_model = model(a)
```

We still need to define moves on our parameters. Ensure that you place the move on the stochastic nodes only. Moves cannot be performed on deterministic or clamped nodes. Here we will use what is called a sliding moves (explain what that is) on the mean, μ, and the precision. The weights here represent how often these moves are performed on average per iteration of our MCMC so this case each move is done on average 0.5 times per iteration.

Monitors to keep trach of our MCMC

```
monitors[1] = mnModel(filename = "output/archery_MCMC.log", printgen = 10, separator =
    TAB)
monitors[2] = mnScreen(printgen = 1000, sd)
```

Finally, assemble our mcmc analysis and run it.

```
mymcmc = mcmc(my_model, monitors, moves)

mymcmc.run(100000, tuningInterval = 0)


mymcmc.operatorSummary()
```

As you can see `RevBayes` is much faster at doing MCMC than the algorithm we implemented in the previous section. Again we can use `Tracer` to examine our output.

4 Exercises

4.1 Using different priors

Suppose we know little about the variance of some archer. In that case, we have little prior belief and a very flat prior might be a good choice. Our conjugate prior is convenient here as the Inverse-Gamma distribution is relatively flat when $\alpha = \beta$ are very small. Estimate the posterior density with alpha and beta as very small values (perhaps try 0.001). compare with your previous posterior distribution.

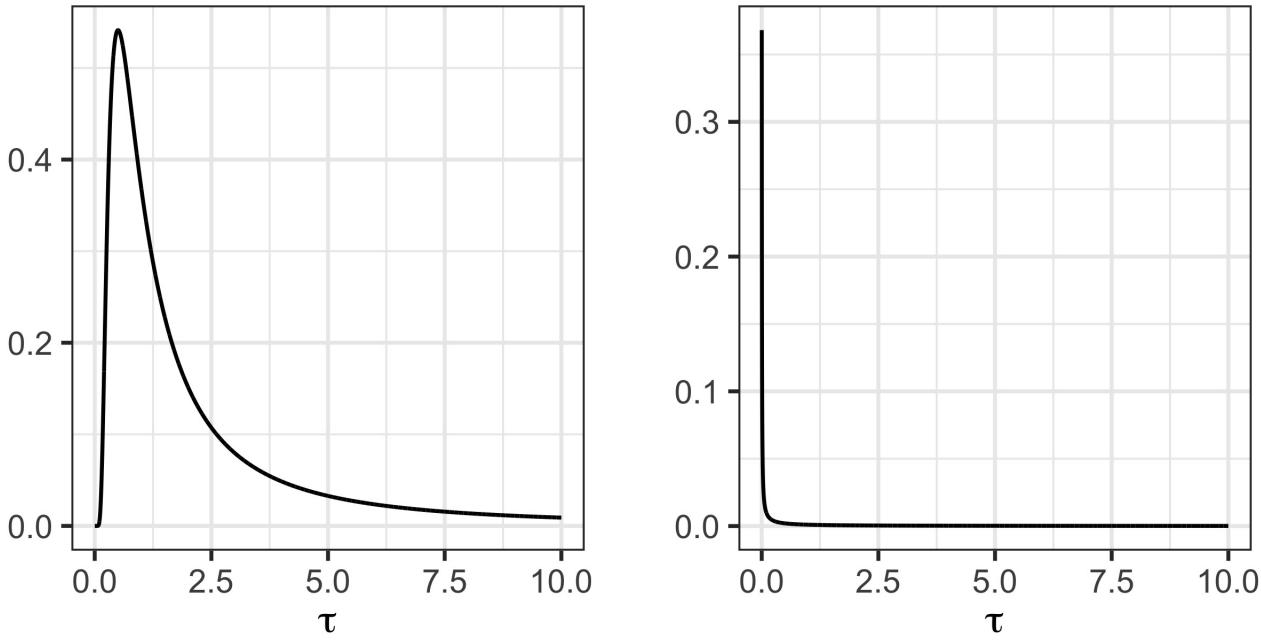

Figure 1: The left shows $InvGamma(1, 1)$, the first prior distribution that was used for the precision, τ. Note the peak around lower parameter values. The right shows $InvGamma(0.001, 0.001)$. Note the much more extreme peak near zero and the relatively flat distribution everywhere else.

The result is not great! Perhaps collecting more data will help, try changing the number of arrows to a larger number (say 100). This still results in a bad estimate this is due to these reasons. Typically when doing MCMC we perform whats called a burn-in where we do some amount of steps and then discard them before continuing. Add this line to the end of your analysis:

```
mymcmc.burnin(generations = 10000, tuningInterval = 0)
```

A little better but still not even close to our true value. In the previous problems all of our moves had a δ that we set to some value. Perhaps the step-size is not ideal. This is where the tuning interval and tune options in our MCMC commands and move specifications come in. `RevBayes` will tune the δ's for these moves to their optimal values if we replace the relevant lines with the following:

```
moves[1] = mvSlide(precision, delta = 0.1, tune = true, weight = 2.0)
moves[2] = mvSlide(mu, delta = 0.1, tune = true, weight = 2.0)

mymcmc.burnin(generations = 10000, tuningInterval = 100)
```

Finally, we arrive at a good estimate of the standard deviation of our distribution. It may have been simpler to use a more well-behaved distribution (see box). Add box about different priors for the standard deviation (e.g. uniform(0,C), Half-Cauchy(0,1))

4.2 Using different moves

So far we have introduced just one move, the sliding move, which takes the current parameter being estimated and adds a uniformly-distributed random variable to that. The width of this uniform distribution is determined by the δ parameter that is set in the move and that the autotune option optimized. Explain about mirror moves, and Depending on the parameter and the model the ability of a move to effectively sample a posterior distribution varies. Fortunately, `RevBayes` comes with many possible moves. For example, we could try placing a scaling move on the precision. Explanation of the scaling move goes here. Note that this move cannot propose a negative number that must be either transformed into the support of the paramter (which is automatically performed in RevBayes) or automatically rejected. This means that the scaling move is more efficient computationally. To use this move in `RevBayes` change the sliding move on the precision.

```
moves[1] = mvScale(precision, lambda = 0.1, tune = true, weight = 2.0)
```

Compare the output of the operatorSummary function for your old move and your new move.

Try the mvUniform move on the precision

Test your new move doing the exercise form the previous section.

Version dated: September 12, 2017