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

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

Introduction to problem. Include within here the distribution the shots are fired under and the first prior distribution. Might be worth showing some similar figures to the ones in Paul's Woods Hole powerpoint here to illustrate points. Also would be a a place to explain the idea of a conjugate prior and why they are useful.

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
```

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:

```
num_arrows <- 6
mu <- 0.0
var <- 1
```

```
arrows <- rnormal(num_arrows, mu, true_var)
```

3.3 Getting Started with MCMC

Explain what MCMC does here.

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 Real likelihood(mu_prime){
       likelihood_mean = 1
       for(i in 1:num_arrows){
              likelihood_mean *= dnormal(arrows[i], mu_prime, sd, log = false, min =
                  0, max = 400)
       }
       return likelihood_mean
}
```

Prior function:

```
function Real priorMean(mu_prime){
       prior_mean = dnormal(mu_prime, prior_mu, sd, log=false, min = 0.0, max = 400.0)
       return prior_mean
}
```

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

```
mu <- rnorm(1, mean = 1, sd = 1, min = 0.0, max= 400)
```

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

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

Metropolis Hastings Algorithm:

```
for(rep in 1:reps){

       mu_prime = mu + rnormal(n=1, 0.0, delta, min = 0.0)[1]
       R = (likelihood(mu_prime) / likelihood(mu)) * ( priorMean(mu_prime) / priorMean
           (mu))

       u = runif(1,0,1)[1]
       if(u < R){
              # accept proposal
              mu = mu_prime
       }
       else{
              # reject proposal
              # nothing to do here
       }
       write(rep,mu,"\n",file="output/archery_MH.log",append=TRUE)
}
```

Here we used a uniform move.

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 <- 0.0   # 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){
       ar[i] ~ dnNormal(mu, sd, min = 0, max = 300)
       ar[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()
```

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.

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 Defining different MCMC moves

Previously, we had been using random draws from (whatever) distribution for the prior. There are lots of possibilities for doing this. We can test how efficient these moves are at sampling our target distributions. Some are far more efficient than others. Code up all of the moves in `RevBayes`.

4.2.1 Random draw

introduce weights here. (maybe have an aside explaining using a 2 parameter model?)

4.2.2 Sliding move

Explain about δ (the tuning parameter). What values are good/bad for this? Explain about how to tune and the purpose of it.

4.2.3 Scaling move

Similarly, explain the tuning parameter for this move. Also, this move is asymmetric so explain more about the calculation of the Hastings ratio.

Compare the three different moves in your two parameter model in tracer. Questions include: is one move better than the other according to ESS? what happens when the tuning parameter is set very high? very low? why does this happen?

After show the autotuning option in `RevBayes`

Version dated: September 6, 2017