Phylogenetic Inference using `RevBayes`

A Simple Binomial Model

Mike May and Brian Moore

1 Overview

This very basic tutorial provides and introduction to Bayesian inference and Markov chain Monte Carlo (MCMC) algorithms. The tutorial explains the fundamental concepts of an MCMC algorithm, such as *moves* and *monitors*, which are ubiquitous in every other tutorial. After the tutorial you should be somewhat familiar with Bayesian inference (*e.g.,* what is a prior distribution, posterior distribution, and likelihood function) and MCMC simulation (*e.g.,* what are moves and monitors and why do we need them).

2 A Coin Flipping (Binomial) Model

We'll begin our exploration of Bayesian inference with a simple coin-flipping model. In this model, we imagine flipping a coin n times and count the number of heads, x; each flip comes up heads with probability p. This model gives rise to the Binomial probability distribution, with parameters n and p:

$$P(x \mid n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

Simple intuition suggests that, given that we observe x heads in n coin tosses, the maximum-likelihood estimate (MLE) of p is simply $\frac{x}{n}$: if we flip a coin 100 times and observe 70 heads, we assume the probability the coin comes up heads is $\frac{70}{100} = 0.7$. This is indeed the maximum likelihood estimate!

From Bayes' theorem, the *posterior distribution* of p given x, $P(p \mid x)$, is:

$$\overbrace{P(p \mid x)}^{\text{posterior distribution}} = \frac{\overbrace{P(x \mid p)}^{\text{likelihood}} \times \overbrace{P(x)}^{\text{prior}}}{\underbrace{P(x)}_{\text{marginal likelihood}}}$$

The take-home message here is that, if we're interested in doing Bayesian inference for the coin flipping model, we need to specify a *likelihood function* and a *prior distribution* for p. In virtually all practical cases, we cannot compute the posterior distribution directly and instead use numerical procedures, such as a Markov chain Monte Carlo (MCMC) algorithm. Therefore, we will also have to write an MCMC algorithm that samples parameter values in the frequency of their posterior probability.

We'll use a simple beta distribution as a prior on the parameter of the model, p. The beta distribution has two parameters, α and β (Figure 1). The graphical model for the binomial model is depicted in Figure 2.

3 Writing an MCMC from Scratch

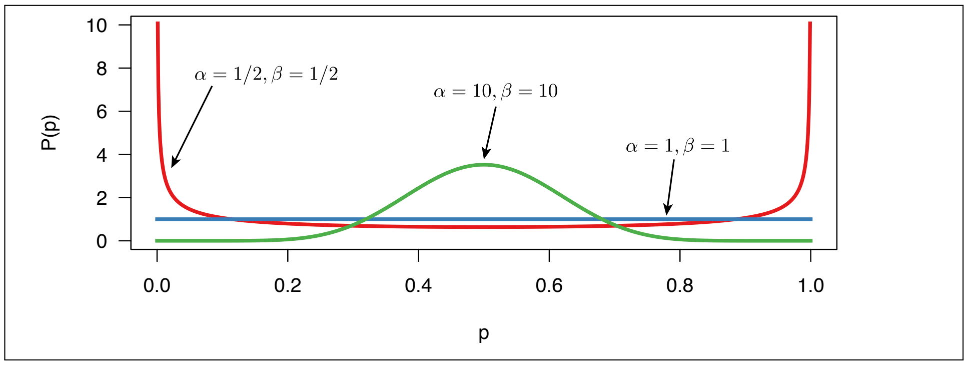
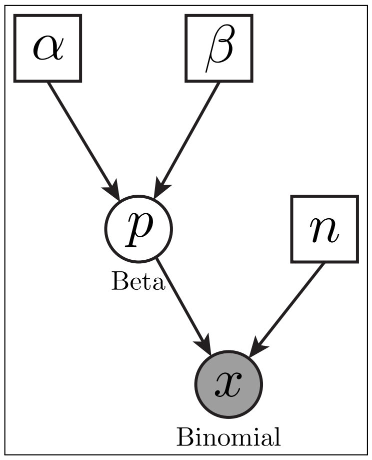


Figure 1: A beta distribution with two parameters, α and β. This distribution is used as a prior distribution on the probability parameter p of observing a head. Here we show different curves for the beta distribution when using different parameters.



3.1 The Metropolis-Hastings Algorithm

Though `RevBayes` implements efficient and easy-to-use Markov chain Monte Carlo algorithms, we'll begin by writing one ourselves to gain a better understanding of the moving parts. The Metropolis-Hastings MCMC algorithm proceeds as follows:

1. Generate initial values for the parameters of the model (in this case, p).
2. Propose a new value for some parameters of the model, based on their current values (which we'll call p')
3. Calculate the acceptance probability, R, according to:

$$R = \min\left\{1, \frac{P(x \mid p')}{P(x \mid p)} \times \frac{P(p')}{P(p)} \times \frac{q(p)}{q(p')}\right\}$$

4. Generate a uniform random number between 1 and 0. If it is less than R, accept the move (set $p = p'$). Otherwise, keep the current value of p.
5. Return to step 2 many many times, keeping track of the value of p.

3.2 Reading in the data

Actually, in this case, we're just going to make up some data on the spot. Feel free to alter these values to see how they influence the posterior distribution

```
# Make up some coin flips!
# Feel free to change these numbers
n <- 100 # the number of flips
x <- 63 # the number of heads
```

3.3 Initializing the Markov chain

We have to start the MCMC off with some initial parameter values. One way to do this is to randomly draw values of the parameter (just p, in this case) from the prior distribution. We'll assume a "flat" beta distribution; that is, one with parameters $\alpha = 1$ and $\beta = 1$.

```
# Initialize the chain with starting values
alpha = 1
beta  = 1
p <- rbeta(1,alpha,beta)[1]
```

We also need to initialize the likelihood and prior values. We use the binomial probability for the likelihood function:

```
# Compute the initial likelihood and prior probabilities
likelihood <- (p^x) * (1-p)^(n-x)
```

and the beta probability for the prior on p:

```
prior     <- p^(alpha-1) * (1-p)^(beta-1)
```

We also need to initialize a file into which to write the MCMC samples:

```
# Prepare a file to log our samples
write("iteration","p","\n",file="binomial_MH.log")
write(0,p,"\n",file="binomial_MH.log",append=TRUE)
```

(You may have to change the newline characters to `"\r\n"` if you're using a Windows operating system.)

3.4 Writing the MH Algorithm

At long last, we can write our MCMC algorithm. We will repeat this resampling procedure many times (here, 10000), and iterate the MCMC using a `for` loop:

```
# Write the MH algorithm
reps = 10000
for(rep in 1:reps){
```

(remember to close your `for` loop at the end).

The first thing we do in the first generation is generate a new value of p' to evaluate. We'll propose a new value of p from a narrow uniform distribution centered on the current value.

```
        # Propose a new value of p
        p_prime <- p + runif(1,-0.1,0.1)[1]
```

One thing we have to be careful about is making sure p (being a probability) stays between 0 and 1. We can do that by reflecting any proposals outside of that region back into the valid region. (One of the nice things about the real MCMC algorithms implemented in `RevBayes` is that it takes care of this for us).

```
        if (p_prime < 0) {
                p_prime <- abs(p_prime)
        } else if (p_prime > 1) {
                p_prime <- 2 - p_prime
        }
```

Next, we compute the proposed likelihood and prior probabilities, as well as the acceptance probability, R:

```
        # Compute the new likelihood and prior
        likelihood_prime <- (p_prime^x) * (1-p_prime)^(n-x)
        prior_prime     <- p^(alpha-1) * (1-p)^(beta-1)

        # Compute the acceptance probability
        R <- (likelihood_prime/likelihood) * (prior_prime/prior)
```

Then, we accept the proposal with probability R and reject otherwise:

```
        # Accept or reject the proposal
        u <- runif(1,0,1)[1]
       if(u < R){
               # Accept the proposal
               p <- p_prime
               likelihood <- likelihood_prime
               prior <- prior_prime
        }
```

Finally, we store the current value of p in our log file.

```
        # Write the samples to a file
        write(rep,p,"\n",file="binomial_MH.log",append=TRUE)

} # end MCMC
```

If we execute this script, the `.log` file will contain samples from the posterior distribution of the model! We can open the file in `Tracer` to learn about various features of the posterior distribution, for example: the posterior mean or the 95% credible interval. Pretty awesome, right? However, this MCMC algorithm is *very* specific to our binomial model and thus hard to extend (also it's pretty inefficient!). We'll now specify the exact same model in `Rev` using the built-in modeling functionality.

4 The Metropolis-Hastings Algorithm with the *Real* `RevBayes`

It turns out that the `Rev` code to specify the above model is extremely simple. Again, we start by "reading in" (*i.e.*,making up) our data.

```
# Make up some coin flips!
# Feel free to change these numbers
n <- 100 # the number of flips
x <- 63 # the number of heads
```

Now we specify our prior model.

```
# Specify the prior distribution
alpha <- 1
beta  <- 1
p ~ dnBeta(alpha,beta)
```

One difference between `RevBayes` and the MH algorithm that we wrote above is that many MCMC proposals are already built-in, but we have to specify them *before* we run the MCMC. We usually define (at least) one move per parameter immediately after we specify the prior distribution for that parameter.

```
# Define a move for our parameter, p
moves[1] = mvSlide(p,delta=0.1)
```

Next, our likelihood model.

```
# Specify the likelihood model
k ~ dnBinomial(p, n)
k.clamp(x)
```

We wrap our full Bayesian model into one model object (this is a convenience to keep the entire model in a single object, and is more useful when we have very large models):

```
# Construct the full model
my_model = model(p)
```

We use "monitors" to keep track of parameters throughout the MCMC. The two kinds of monitors we use here are the `mnModel`, which writes parameters to a specified file, and the `mnScreen`, which simply outputs some parts of the model to screen (as a sort of progress bar).

```
# Make the monitors to keep track of the MCMC
monitors[1] = mnModel(filename="binomial_MCMC.log", printgen=10, separator = TAB)
monitors[2] = mnScreen(printgen=100, p)
```

Finally, we assemble the analysis object (which contains the model, the monitors and the moves), and execute the run using the `.run` command:

```
# Make the analysis object
analysis = mcmc(my_model, monitors, moves)

# Run the MCMC
analysis.run(100000)
```

Open the resulting `binomial_MCMC.log` file in `Tracer`. Do the posterior distributions for the parameter p look the same as the ones we got from our first analysis?

Hopefully, you'll note that this `Rev` model is substantially simpler and easier to read than the MH algorithm script we began with. Perhaps more importantly, this `Rev` analysis is *orders of magnitude* faster than our own script, because it makes use of extremely efficient probability calculations built-in to `RevBayes` (rather than the ones we hacked together in our own algorithm).

4.1 Exercise

Play around with various parts of the model to develop on intuition for both the Bayesian model and the MCMC algorithm. For example, how does the posterior distribution change as you increase the number of coin flips (say, increase both the number of flips and the number of heads by an order of magnitude)? How does the estimated posterior distribution change if you change the prior model parameters, α and β (*i.e.,* is the model prior sensitive)? Does the prior sensitivity depend on the sample size? Are the posterior estimates sensitive to the length of the MCMC? Do you think this MCMC has been run sufficiently long, or should you run it longer?

Version dated: December 20, 2016