Mattias Villani
Department of Statistics
Stockholm University
Dept of Computer and Information Science
Linköping University

How code up the Random Walk Metropolis Algorithm in R

This little note will help you with coding up the random walk Metropolis algorithm so that the same function, let's call it `RWMSampler`, can be applied to simulate from the posterior of the parameters in *any* model. The trick is to use *function objects* in `R` and the *triple dot* $(\ldots)$ wildcard argument. Note the following:

1. One of the input arguments of your `RWMSampler` function should be `logPostFunc` (or some other suitable name). `logPostFunc` is a *function object* that computes the log posterior density for any value of the parameter vector, which is need for computing the acceptance probability in the Metropolis algorithm. You should always code the *log* posterior density and then evaluate the acceptance probability as

$$\frac{p(\theta_p|\mathbf{y})}{p(\theta^{(i-1)}|\mathbf{y})} = \exp\left[\log p(\theta_p|\mathbf{y}) - \log p(\theta^{(i-1)}|\mathbf{y})\right].$$

   This gives numerical stability since common multiplicative factors in $p(\theta_p|\mathbf{y})$ and $p(\theta^{(i-1)}|\mathbf{y})$ cancel out before we evaluate the exponential function (which can otherwise overflow).

2. The first argument of your (log) posterior function should be `theta`, the vector of parameters for which the posterior density is evaluated. You can of course use some other variable name, but it must be the *first* argument of your posterior density function.

3. The user's posterior density is also a function of the data and prior hyper-parameters and those can can be supplied to the `RWMSampler` function by using the triple dot $(\ldots)$ argument to functions. The triple dot acts like a wildcard for *any* parameters supplied by the user. This makes it possible to use the Metropolis function for any problem, even when you as a programmer don't know what the user's posterior density function looks like, or what kind of data and hyper-parameters will be used in that particular problem. To illustrate the use of the triple dot argument, I give some very simple code below with the log posterior density for the Bernoulli model with a Beta prior. The log posterior density is then used in a useless, but illustrative, function `MultiplyByTwo` that returns 2 times the log posterior density evaluated at $\theta = 0.3$. Note how the `MultiplyByTwo` takes a function object as input and how it uses the triple dot $(\ldots)$ argument to supply the data s and f, and the prior hyper-parameters a and b without explicitly using these symbols inside the function. This makes the `MultiplyByTwo` function applicable for *any* function.

```
# This is the log posterior density of the beta(s+a,f+b) density
LogPostBernBeta <- function(theta, s, f, a, b){
 logPost <- (s+a-1)*log(theta) + (f+b-1)*log(1-theta)
 return(logPost)
}

# Testing if the log posterior function works
s <- 8;f <- 2;a <- 1;b <- 1
logPost <- LogPostBernBeta(theta = 0.1, s, f, a, b)
print(logPost)

# This is a rather useless function that takes the function myFunction,
# evaluates it at x = 0.3, and then returns two times the function value.
MultiplyByTwo <- function(myFunction, ...){
 x <- 0.3
 y <- myFunction(x,...)
 return(2*y)
}
#Let's try if the MultiplyByTwo function works:
MultiplyByTwo(LogPostBernBeta,s,f,a,b)
```