LAB 5. MARKOV CHAIN MONTE CARLO

This exercise builds your confidence in R programming and helps you understand the concept of a Monte Carlo Markov chain using Metropolis steps implemented in a Gibbs sampler. The most effective ways to learn this stuff is to set up a problem where you know the answer and then see if you can find the answer using the concepts and methods that you seek to understand. This is what you will do in this lab—it is a very powerful way to learn new methods. So, it is really important that you keep your notes on the concepts in front of you when you are doing the coding. That way you can see if you can translate the concept into something more concrete. Please try hard to go back and forth between the concepts we covered in lecture and the coding you are doing in this lab—don't get down in the R weeds and lose the big picture of MCMC.

Here are some general tips for mastering a new method like MCMC. First, always work with a known answer—something you derive analytically (as in the lecture chrytid fungus example) or that you can estimate by another, proven method (as you will do in this lab). First, we will go through the simple chrytid fungus MCMC example. Recall that we are interested in estimating incidence (phi). We have sampled 12 frogs of which 3 are infected. We'll walk through the basic steps.

Define a binomial likelihood function that takes 1 argument, the probability of success, phi and a prior uniform distribution.

```
 # Define the binomial likelihood, assume 3 successes (i.e., infected) on 12 trials.
n=12
k=3
L = function(phi)dbinom(3,size=12, prob=phi)

# Define an uninformative distribution for the prior
pr = function(z) dunif(z,0,1)   # a uniform prior
```

Set up the chain, the proposal distribution, and the decision vector. These need to be the length of the chain.

```
ni = 200000 # number of steps in chain
x = numeric(ni) # vector to hold chain
x[1] =0 .1  # initial value for chain
z = runif(ni,0,1) # proposal distribution
u = (runif(ni,0,1)) #decision vector

for(j in 2:ni){
        #get ratio of likelihoods * priors
        ratio = L(z[j]) *pr(z[j])  / L(x[j-1])*pr(x[j-1])

        #decide whether to keep propoal
        if ( u[j] < ratio) x[j] = z[j] else x[j] = x[j-1]
}
```

Think about what the ratio is: if the new value of z is better, then the likelihood of observing your data given z will be higher than the one at step j-1 as well as the probability of z, so you will accept the proposal. The decision vector came from a uniform distribution so it is not very conservative. You could make your chain more conservative by selecting high value for the decision vector so often it will be greater than the ratio.

Plot results from the direct calculation of the posterior via the conjugate and the calculation from MCMC. Recall that the posterior estimate for a binomial likelihood with *n* trials and *y* successes is a beta distribution with parameters:

$$P(\theta \mid y) = beta(1+y, 1+n-y))$$

```
par(mfrow=c(1,1))

# calculate  posterior analytically assuming uninformative beta prior (i.e., beta(1,1)
alpha = 3 + 1
beta = 12 - 3 + 1

# Set up a sequence of possible value for theta to be able to calculate and plot the posterior
distribution.
s = seq(0,1,.01)

# calculate posterior density of theta directly
b = dbeta(s,alpha,beta)

# plot chain values stored in x as a density
plot(density(x, adjust=1),col="red", main = paste("Simulated and Calculated Distribution,
iterations = ",ni), xlab = expression(phi), ylab = expression(paste("P(", phi, "|data)")),
cex.lab=1.2, cex.main=1.5, ylim=c(0,6))

# and overlay the theoretical distribution from the posterior estimation
lines(s,b)

# plot the rug (i.e., all sampled values of x (theta) at bottom curve
rug(x)

text = c("Calculated", "Simulated")
colors = c("Black","Red")
lines = c(1,1)
legend(.4,5.0,text, lty = lines, col = colors, cex = 1.3)
```

Exercise 1: Normal likelihood, uniform independent proposals, Metropolis algorithm, Gibbs Sampler

We will start with a normal likelihood function, a uniform proposal distribution, and a Metropolis algorithm in a Gibbs sampler. You will move, step by step, to a more challenging problem, a beta log-likelihood function with a gamma proposal distribution.
For this first problem, we will estimate the parameters for a model of exponential decay of leaf litter where we start with M0 litter in the first time step. These functions are used often in ecology. Here are your steps.

1) Simulate 50 data points from a normal distribution using the following model:

$$\mu_t = M_0 e^{-kt}$$
$$y_t \sim \text{normal}(\mu_t, \sigma)$$

Assume M_0 =0.87, k =0.04, σ = 0.03 and t = 1 through 50. Make a nice plot of the simulated data and the model as a function of time (t).

2) Use R's nls() function to estimate means and standard deviations for M_0 and k, realizing of course, that it is using a normal likelihood. The residual standard error corresponds roughly to your estimate of σ, below.

3) Write a function to set up prior distributions for the parameters ((M_0 , k, σ). You can make this informative or uninformative. For instance, here is a really uninformative prior:

```
prior.k = function(k) 1
prior.M0 = function(M0) 1
prior.sigma=function(sigma) 1
```

4) Write a normal likelihood function that takes 3 arguments (M_0 , k, σ)**,** and returns the total likelihood of the data conditional on the parameters. This will be very similar to the exercise for the hemlock trees we did last week.

5) Now we will construct a Gibbs sampler using a Metropolis accept-reject step for each of the three parameters. Here are some hints. Most important, *do the simple things first; get it working; test it; and then move to the next step*. So the first thing to do is to write a likelihood function that takes arguments for the three parameters and the data and returns the product of the likelihoods across all data points. Test this function by plotting likelihood profile for each of the parameters, say k, holding the others constant at their known values (i.e., the ones you used to generate the data). If you proceed before you test this function thoroughly, I can promise you will suffer later.

6) Next, estimate a single parameter using a Gibbs sampler, say, k, treating the other parameters as known in all steps—you know their values from the nls( ) analysis you did in step 2. Get this working first, then add a step for the second parameter and estimate the first and second parameter. Get this working, then add the third one. Follow the example from the hemlock trees we covered in lecture.

7) To achieve this, you will need to create vectors of proposals of length = total number of simulations using uniform proposal distributions for each parameter. Store the values of the chain for each parameter in vectors of length = number of simulations. To save you some time (and suffering), I found that the following proposal distributions worked well:

```
#set up parameters bounds for uniform proposal distributions
k.low=0.01
k.up = 0.1
M0.low = 0.70
M0.up = 0.95
sigma.low = 0.01
sigma.up = 0.06

#create vectors of proposals for model parameters
prop.k = runif(n.sim, k.low, k.up)
prop.M0 = runif(n.sim,M0.low,M0.up)
prop.sigma = runif(n.sim,sigma.low,sigma.up)
```

You can make them broader, but the broader they are, the more iterations you will need to get good estimates. You will also need to provide the first value of the chain for each parameter and vectors that store random draws from a uniform distribution for each of the parameters. See lecture notes for this.

8) The acceptance rule for Gibbs sampler with Metropolis algorithm for parameter M0 is:

```
R = min(1, ratio) #prevents R from going above 1
if(u.M0[j] < R) M0[j] = prop.M0[j] else M0[j] = M0[j-1]
```

9) Think about how to deal with convergence. Think about throwing away some number of members of the chain to represent burn in. You can visually inspect this by plotting the running average of your iterations. Talk this over with you lab mates and write a brief description of your discussion. You will probably need 200,000 or more iterations to get a really good chain for your final estimates, one that will produce fairly smooth densities. However, when you are developing this code, you can use something more like 10,000 burn in + 25,000 iterations you keep. This is good practice---you can get a good sense of how the model is working using a relatively small number of iterations. Even if the plotted densities are jagged, you will find that the means and the standard errors are well estimated relative to the nls() estimates. This accelerates model development and debugging. When all is running well, then you can add multiple chains for each parameter and increase the number of iterations.

10) Estimate the means of the parameters and their standard deviations based on Gibbs sampling and compare them with the estimated values from nls(). Talk with you lab mates about how to obtain these quantities. You will know you have succeeded when *M0* and *k* are very close to the nls() estimates. Your estimate of σ using Gibbs will always be slightly greater than the estimate of the residual error from nls(). Why?

11) Plot the chains for each parameter. Plot the posterior distributions of your estimates of each parameter.. Estimate 95% credible intervals on all parameters and quantities of interest (hint—use the R quantile() function with the vector that stores the parameters values that were kept (e.g., M0, k, or sigma). Write a brief description of the difference between a credible interval and a confidence interval. Plot a histogram of the chain and overlay the density representing the posterior distribution. What the density() function is doing is explained nicely at http://en.wikipedia.org/wiki/Kernel_density_estimation

A technical note: The adjust = option for the density( ) function is a way to "smooth" the plot of the density. Alternatively, you can get a smoother plot with adjust = 1 if you do more iterations. Try playing with it. If you set adjust > 1 in a plot that you include in a publication you need to say something like: "A kernel estimator with a narrow bandwidth was used to smooth marginal posteriors produced by MCMC simulation."

Exercise 2: Beta log-likelihood, uniform proposal, Metropolis algorithm, Gibbs Sampler

Now assume that the data (i.e., decomposition, think about it as a proportion) can only take on values between 0-1 such that a beta likelihood is a better choice than the normal. The model still takes the form:

$$\mu_t = M_0 e^{-kt}$$
$$y_t \sim \text{beta}(a,b)$$

but the y's are bounded between 0 and 1.

Begin by simulating data from a beta distribution using the same parameters as above and modify your code to use a beta distribution in your likelihood function. You will need to use the method of moments to come up with shape parameters for the beta distribution.

Follow the steps outlined above to come up with functions for the prior distributions and the likelihoods. For this to work, you will need to use log-likelihoods instead of likelihoods, but be careful about how you calculate the accept-reject ratio. With the beta, you will also need to pay attention to illegal operations—the beta is more fussy than the normal. So, the following lines of code will be useful:

```
L.vector = dbeta(y,a,b, log=TRUE)
like=sum(L.vector[is.finite(L.vector)])
```

Think about what is going on here. We first create a vector of likelihoods of the data (y), some of which turn out to be infinite, generating those NaN warnings. We exclude those from the sum using is.finite( ), which is analogous to the is.na( ) function.