Markov chain Monte Carlo—General Principles*

Aaron A. King

May 27, 2009

Contents

1 The Metropolis-Hastings algorithm

A Markov chain Monte Carlo algorithm is just a systematic way of drawing random numbers from a complicated distribution. Its extreme flexibility is its primary appeal. The heart of MCMC is the Metropolis-Hastings algorithm, developed in the 1940s-1970s by a group of physicists associated with the Manhattan Project. In this lecture, I will focus on the essence of the Metropolis-Hastings algorithm. Once you understand this, there are an infinity of variations you can tailor to the questions you ask and the data you have.

Suppose you wish to draw samples from a probability distribution $f(\theta)$. If you're doing Bayesian inference, then this distribution will generally be the posterior distribution of some model parameters. What we're about to learn about MCMC doesn't depend on *why* you are interested in the distribution f, however. Metropolis-Hastings (MH) is a recipe for drawing a sequence of random values $\theta_1, \theta_2, \ldots, \theta_k, \ldots$. This is achieved by repeating one basic step many times: a move from θ_k to θ_{k+1}. In this step, you first "propose" a value of θ_{k+1} and then decide whether to accept the proposal or reject it. One has great freedom in the choice of the random proposal. The useful fact is that (after some initial transient, or "burn-in", phase), the values θ_k will be samples from $f(\theta)$.

Algorithm 1 (Metropolis-Hastings). Ingredients: (a) the *target distribution* $f(\theta)$, (b) a *proposal distribution* $q(\theta \to \theta^*)$, which gives the probability that you make the proposal θ^* given that you are at θ, and (c) a starting value θ_0 .

Procedure:

1. Propose θ^* according to the proposal distribution q.
2. Compute
$$\alpha = \frac{f(\theta^*)\, q(\theta^* \to \theta_k)}{f(\theta_k)\, q(\theta_k \to \theta^*)} \tag{1}$$
3. Set $\theta_{k+1} = \theta^*$ with probability $\min\{\alpha, 1\}$.
4. Go back to step 1 and repeat many, many times.

Remark 2. Notice that we have only used the fact that f is a positive function. In fact, since the only time MH sees f at all is in the ratio $f(\theta*)/f(\theta)$, we only need to have that $f(\theta)$ is *proportional* to the target distribution.

Let's see how this works using a simple example. Let's suppose we have a distribution which is proportional to some mystery function, `mystery.fun`. In this example, this function is just serving as a stand-in for a problem of interest. If you're using MCMC for Bayesian inference, this function would be the likelihood times the prior, i.e., the numerator of Bayes' theorem. We'll use a simple proposal distribution: given θ, propose a random jump to $\theta^* = \theta + U$, where U is a uniform random number on the interval $[-10, 10]$ (we write $U \sim \text{uniform}(-10, 10)$ as shorthand for the latter statement). Because, left to itself, this proposal would wander randomly up and down the line, it is what is called a *random-walk* proposal. In the exercises below, you'll get a chance to explore different proposals. The code in Box 1 implements the Metropolis-Hastings algorithm in this case.

The fundamental fact about MH is expressed in the following

Theorem 3. If the distribution of the θ_k converges, it converges to the distribution f (or the distribution proportional to f).

The upshot of this theorem is that if we can verify that the distribution of the samples θ_k has converged, then we know that it has converged to the target distribution.

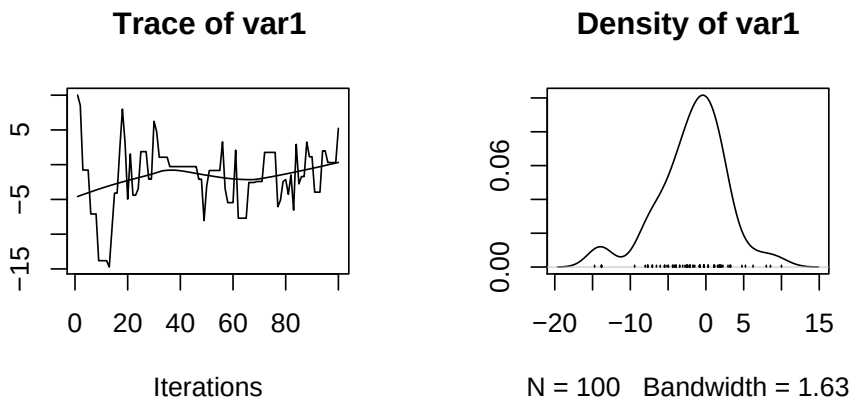
Remark 4. Note that we are quite free to choose the proposal distribution without changing the result. In effect, the Metropolis-Hastings acceptance probability makes just the right correction.

Remark 5. In fact, it is very easy to choose the proposal distributions so that the sequence will converge, at least in theory. Not to put to fine a point on it, if the proposal distribution is chosen so that any value of θ can eventually be proposed, then the chain will eventually converge.

Remark 6. *Eventually* can be a long time.

Let's run the algorithm and plot the resulting trajectory of θ_k:

```
theta <- mcmc.mh(theta=10,fun=mystery.fun,niter=100)
plot(theta)
```



On the left we have the sequence of values of θ_k plotted against k. On the right, we see the package `coda`'s attempt to guess the distribution of θ on the basis of these 100 samples. Obviously, it's going to be a pretty bad guess: we expect that we'll have to take a lot more samples to even be able to verify that the distribution has converged and we'd have to sample even more to get a good idea as to the shape of the distribution. How many samples will we need? The `coda` package provides a number of tools we can use to help answer this question.

```
theta <- mcmc.mh(theta=10,fun=mystery.fun,niter=10000)

effectiveSize(theta)

   var1
1813.793

raftery.diag(theta)

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

 Burn-in  Total Lower bound  Dependence
 (M)      (N)   (Nmin)       factor (I)
 13       14170 3746         3.78
```

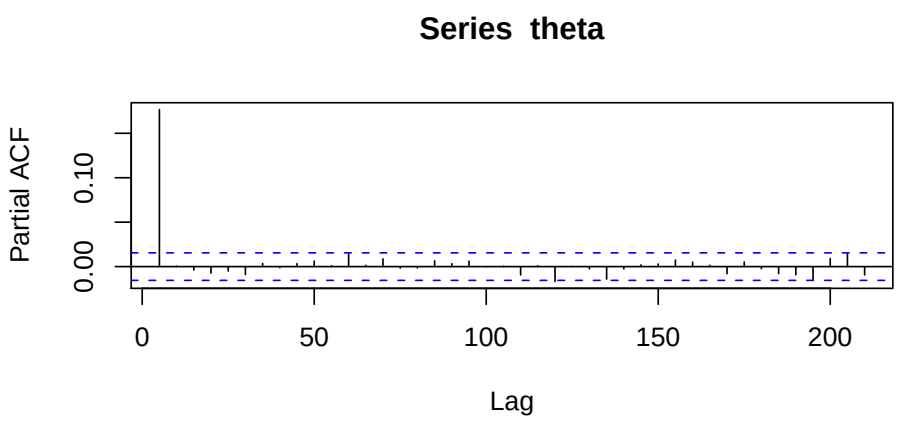
The samples we draw are obviously correlated with one another, so the effective sample size here is less than 10000. The `effectiveSize` diagnostic tries to figure out roughly what this effective sample size is. The `raftery.diag` diagnostic uses the Raftery & Lewis method to try and figure out roughly how long a run we'll need to have, how long a burn-in period we'll need, and how strong the autocorrelation among samples is.

```
theta <- mcmc.mh(theta=10,fun=mystery.fun,niter=1000)  ## burn-in: discard these
theta <- mcmc.mh(theta=tail(theta,1),fun=mystery.fun,niter=16000,thin=5)

effectiveSize(theta)
```

```
    var1
11195.25
```

We can check to see how autocorrelated successive samples are.

```
pacf(theta)
```



Let's thin a bit more aggressively to get rid of this autocorrelation.

```
theta <- mcmc.mh(theta=tail(theta,1),fun=mystery.fun,niter=10000,thin=10)
```

There are a number of other diagnostics we can use to assess convergence, all documented in the `coda` package.

```
geweke.diag(theta)
```

```
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

    var1
-0.01357
```

```
pnorm(q=-abs(geweke.diag(theta)$z))
```

```
     var1
0.4945858
```

```
effectiveSize(theta)
```

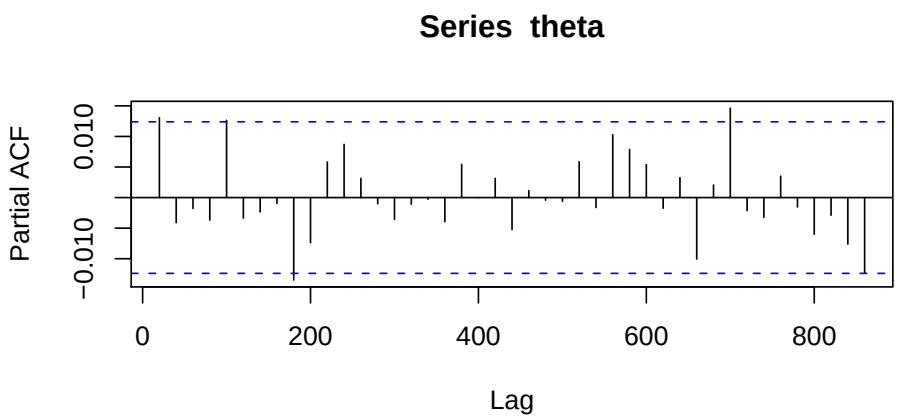
```
   var1
9137.03
```

```
heidel.diag(theta)
```

```
     Stationarity start     p-value
     test         iteration
var1 passed       1         0.204

     Halfwidth Mean   Halfwidth
     test
var1 failed    -0.591 0.0871
```

We're pretty much there, but the Heidelberger-Welch diagnostic is not quite happy, and we still see some autocorrelation, so just to be on the safe side:

```
 theta <- mcmc.mh(theta=tail(theta,1),fun=mystery.fun,niter=25000,thin=20)

 heidel.diag(theta)

     Stationarity start     p-value
     test         iteration
var1 passed       1         0.638

     Halfwidth Mean  Halfwidth
     test
var1 passed    -0.55 0.0512

 geweke.diag(theta)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

   var1
-0.7623

 pnorm(q=-abs(geweke.diag(theta)$z))

     var1
0.2229309

 effectiveSize(theta)

    var1
24354.59

 pacf(theta)
```

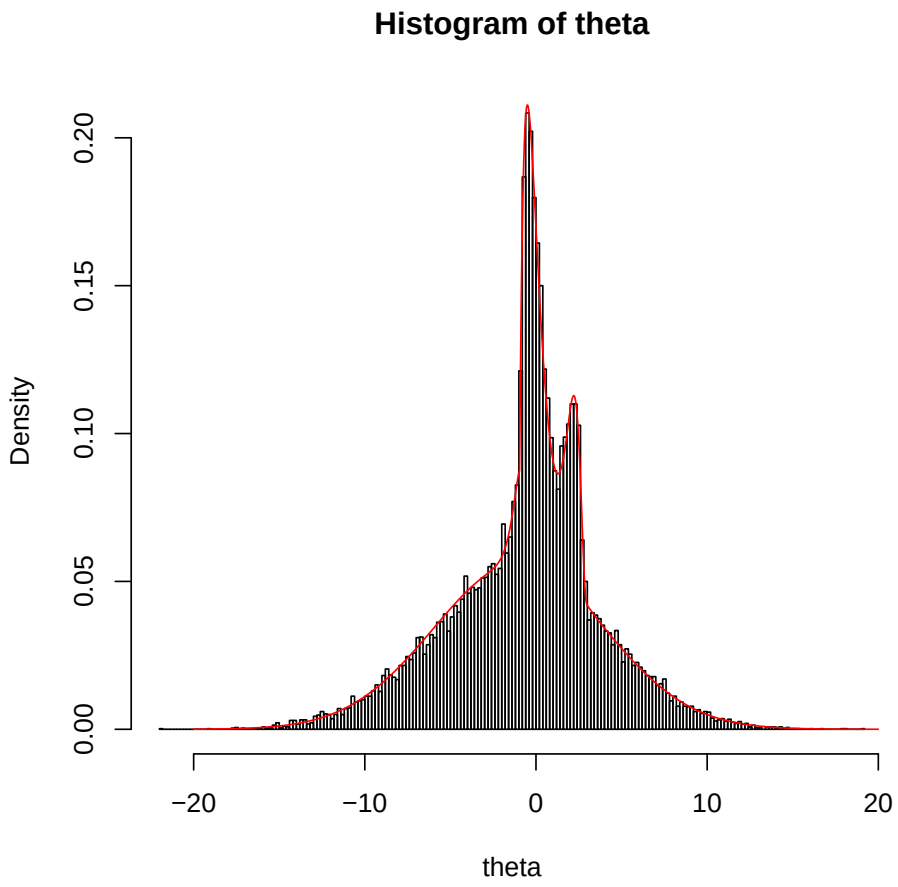


All indications are that we've converged pretty well by this point.

Remark 7. In this simple example, we've been content to run a single Markov chain. In practice, you'll probably want to run multiple chains, to verify that they all converge to the same distribution. There are other convergence diagnostics that can be applied once you've got multiple chains to work with.

Let's unveil the mystery distribution and compare it with the distribution we've obtained by MCMC.

```
hist(theta,breaks=250,freq=F)
z <- integrate(mystery.fun,lower=-20,upper=20)$value
curve(1/z*mystery.fun(theta=x),from=-20,to=20,col='red',add=T,n=1001)
```



In closing this discussion of MH, I note that although the example is univariate, there is nothing stopping us from using MCMC to study multivariate, even very high-dimensional, distributions.

Exercise 1. Modify the code of Box 1 to reproduce the example using a uniform random-walk proposal $\theta^* \sim \text{uniform}(\theta - 1, \theta + 1)$. How does the sampler behave with this choice of proposal? That is, what choice of burn-in period, length of sample, and thinning ratio do you find that you need? Now repeat the exercise using the proposal $\theta^* \sim \text{uniform}(\theta - 20, \theta + 20)$.

Exercise 2. Modify the code of Box 1 to use an *independent* proposal, i.e., one in which the proposal θ^* is independent of the current value θ. Specifically, try $\theta^* \sim \text{normal}(\mu = 0, \sigma = 10)$. Since the proposal is no longer symmetric, you'll have to modify the accept/reject rule (see Eq. 1).

2 Completion (AKA data augmentation)

The MCMC approach, as we've seen, is not to compute the distribution directly but rather to sample from it and build up an understanding of its shape that way. All we need to be able to do this is to compute the numerator of Bayes' theorem (likelihood times prior). However, it is frequently the case that we can't even compute likelihoods because some data are missing.

As as example, let's return to the chain-binomial. In that model, each incidence is related to the incidence

one generation before via

$$I_{t+1} \sim \text{binomial}(S_t, 1 - e^{-\beta I_t}) \qquad \text{where} \qquad S_t = S_0 - \sum_{u=1}^{t} I_u$$

We can easily write the likelihood for β and S_0. Let I represent the entire sequence of cases $\{I_t\}_{t=0}^n$. Then the likelihood is just the probability we observed the data I given the parameters β, S_0:

$$f(I|\beta, S_0) = \prod_{t=0}^{n-1} \binom{S_t}{I_{t+1}} (1 - e^{-\beta I_t})^{I_t} (e^{-\beta I_t})^{S_t - I_{t+1}}$$

If, say, I_3 were missing, then we'd not be able to compute S_t for $t > 2$, and so we'd therefore be unable to compute the likelihood of I_t for $t > 3$. To get around this problem, we could treat the missing value I_3 as a parameter and attempt to estimate it along with β and S_0. It's straightforward to write the likelihood

$$f(\beta, S_0, I_3|\text{data})$$

We're not really interested in estimating I_3 itself, however; rather, we're interested in I_3 only insofar as not knowing it makes our estimates of S_0 and β less certain. To incorporate this uncertainty, we need to integrate over all possible values of I_3:

$$f(\beta, S_0|\text{data}) = \int f(\beta, S_0, I_3|\text{data})\, dI_3$$

This integral might be very difficult to compute. If we adopt an MCMC approach, however, we don't ever need to compute this integral. We get the distribution we're interested in by sampling *beta*, S_0, and I_3 from the full chain using MCMC and simply throwing away the values of I_3.

This procedure of "augmenting the data" to make computation of the likelihood possible is called "demarginalization" or "completion" of the target distribution, and is a very important tool in our toolbox.

3 The Gibbs sampler

Using MH is something of an art: once you've figured out how to write the likelihood and chosen priors, it takes a little practice to be able to figure out how long the chains have to be run and how long a burn-in period to use. More artful still is the choice of the proposal distribution. There's no optimal proposal for all situations. The Gibbs sampler is a special case of MH where we let the model supply the proposal distribution. Here's the basic idea. Suppose we have two parameters we want to estimate, θ_1 and θ_2. We'll do each round of MH in two steps: first we'll make a move in the θ_1 direction, then we'll change θ_2. Since $f(\theta_1, \theta_2) = f(\theta_1|\theta_2)\, f(\theta_2)$, we can write the MH acceptance probability for the $\theta_1 \to \theta_1^*$ move as

$$\alpha = \frac{f(\theta_1^*, \theta_2)\, q(\theta_1^* \to \theta_1)}{f(\theta_1, \theta_2)\, q(\theta_1 \to \theta_1^*)} = \frac{f(\theta_1^*|\theta_2)\, f(\theta_2)\, q(\theta_1^* \to \theta_1)}{f(\theta_1|\theta_2)\, f(\theta_2)\, q(\theta_1 \to \theta_1^*)} = \frac{f(\theta_1^*|\theta_2)\, q(\theta_1^* \to \theta_1)}{f(\theta_1|\theta_2)\, q(\theta_1 \to \theta_1^*)}.$$

Now of course it's difficult to sample from $f(\theta_1, \theta_2)$ (otherwise why would be using MCMC?) but it may not be hard to draw samples from the conditional distribution $f(\theta_1|\theta_2)$. If we can do this, why not use it as our proposal? Then $q(\theta_1 \to \theta_1^*) = f(\theta_1^*|\theta_2)$, $q(\theta_1^* \to \theta_1) = f(\theta_1|\theta_2)$, and $\alpha = 1$: we always accept the proposal! If we can simulate from $f(\theta_2|\theta_1)$, then we can do the same thing for the other half. We can generalize this to any number of parameters to get

Algorithm 8 (Gibbs Sampler). Ingredients: (a) simulators for each of the conditional distributions $f(\theta_k|\theta_{-k})$. Here θ_{-k} is shorthand for all the elements of the parameter vector θ except for θ_k. (b) an initial guess for all but one of the parameters, say θ_{-1}. .

Procedure:

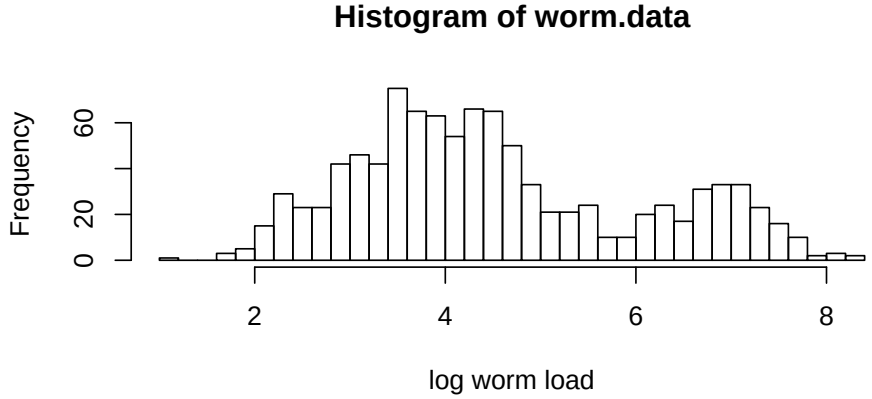
1. simulate $\theta_1 \sim f(\theta_1|\theta_{-1})$
2. simulate $\theta_2 \sim f(\theta_2|\theta_{-2})$

   $\vdots$
3. simulate $\theta_n \sim f(\theta_n|\theta_{-n})$
4. go back to Step 1 and repeat many, many times.

Remark 9. Looking back at the basic MH algorithm, all those proposals that we reject may seem quite wasteful. From this perspective, Gibbs seems very efficient. On the other hand, a very high acceptance fraction may indicate that we're being too conservative in our proposals: we're accepting the proposals frequently because we're not making very interesting proposals. More generally, there is a tradeoff between breadth of exploration at the global scale and carefulness at the local scale.

Let's look at an example. In a large sample of rabbits, we measure helminthic parasite loads. When we log-transform the data, we see the following picture.

```
hist(worm.data,breaks=50,freq=T,xlab="log worm load")
```



We hypothesize that there is an interaction in this system between worm infection intensity and myxovirus coinfection. Specifically, we hypothesize that individual rabbits that are infected with the virus have a different degree of resistance to the worms. On the other hand, we hypothesize that the probability of being infected with the virus does not depend on worm load. Let's model this hypothesis as follows. Let p be the probability of a rabbit's being infected with the virus, i.e., the prevalence of myxoma in this population. For each rabbit k, let $Z_k = 1$ if that rabbit is infected, $Z_k = 2$ if it is virus-free. An infected rabbit ($Z_k = 1$) has a worm load X_k which is log-normally distributed, i.e., $\log X_k \sim \text{normal}(\mu_1, \sigma_1)$. An uninfected rabbit ($Z_k = 2$) also has a log-normal worm load, but with a different mean and variance, $\log X_k \sim \text{normal}(\mu_2, \sigma_2)$. It's somewhat artificial here, but to keep life simple, let's assume that the standard deviations σ_1 and σ_2 are known. Given priors on μ_1, μ_2, and p and the data X, we'd like to be able to sample from the posterior

$$f(\mu_1, \mu_2, p|X) \propto f(X|\mu_1, \mu_2, p)\, f(\mu_1)\, f(\mu_2)\, f(p).$$

Computing the likelihood $f(X|\mu_1, \mu_2, p)$ is hard to do in this case because for each rabbit, we have to consider both possibilities for its viral infection status Z. While this leads to only two terms in the likelihood $f(X_k|\mu_1, \mu_2, p)$, the overall likelihood is the product of all these partial likelihoods and involves

2^n terms. On the other hand, if we knew Z_k for every rabbit, writing the likelihood $f(X, Z|\mu_1, \mu_2, p)$ is easy. So let's demarginalize in this way and try to figure out how to sample from $f(Z, \mu_1, \mu_2, p|X)$. If we use Gibbs sampling, then we can break the problem into pieces.

First let's figure out how to sample from $f(Z|\mu_1, \mu_2, p, X)$. From Bayes' theorem we have that

$$f(Z|\mu_1, \mu_2, p, X) \;\propto\; f(X|\mu_1, \mu_2, Z)\, f(Z|p)$$

This just says that, if we are given p, μ_1, and μ_2, we should choose $Z_k = 1$ with a probability proportional to $p\,\text{normal}(X_k, \mu_1, \sigma_1)$ and $Z_k = 2$ with probability proportional to $(1-p)\,\text{normal}(X_k, \mu_2, \sigma_2)$.

In the second step of the Gibbs sampler, we'll assume that Z is given and sample from

$$f(p|Z, \mu_1, \mu_2, X) = f(p|Z) \;\propto\; f(Z|p)\, f(p).$$

This will be easy if we choose the prior $f(p)$ to be conjugate to the likelihood $f(Z|p)$. Since each $Z_k \sim \text{binomial}(1, p)$, we need to suppose that $p \sim \text{beta}(a, b)$. With this assumption, we sample p using

$$p \;\sim\; \text{beta}(a + n_1, b + n_2), \quad \text{where} \quad n_1 = \#\{Z_k = 1\} \quad \text{and} \quad n_2 = \#\{Z_k = 2\}.$$

In the third and last step of the Gibbs sampler, we sample from μ_1 and μ_2 using

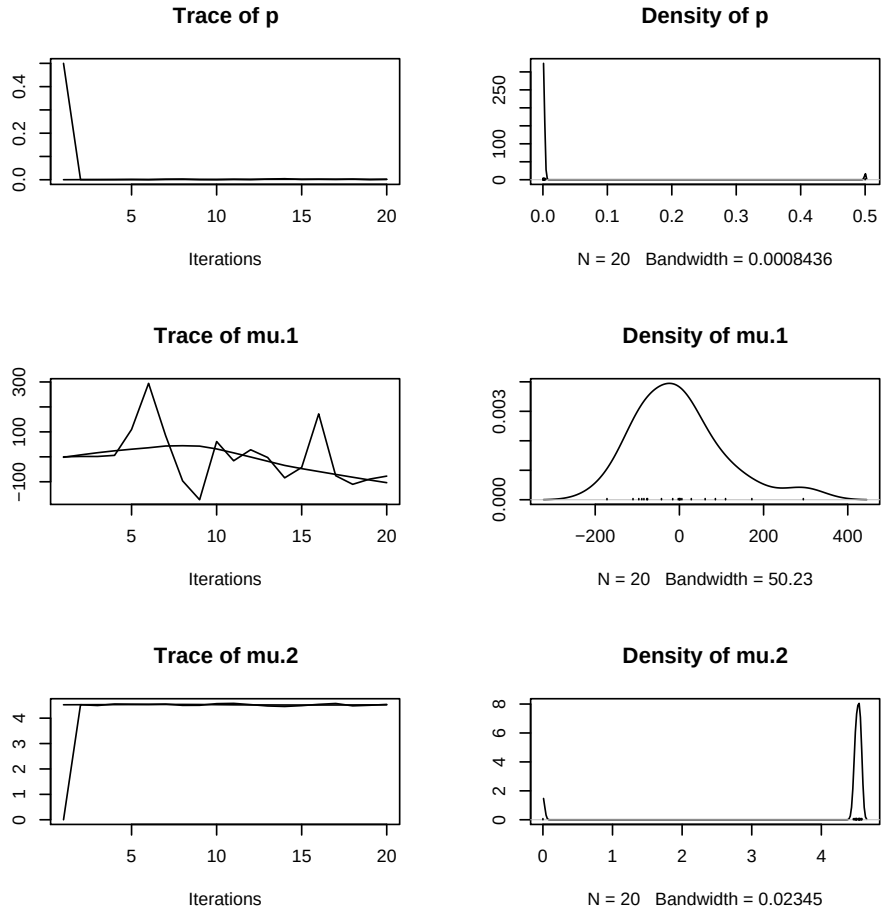
$$f(\mu_1, \mu_2|p, Z, X) \;\propto\; f(X|\mu_1, \mu_2, Z)\, f(\mu_1)\, f(\mu_2)$$

Again, we'll need to assume conjugate priors to make this work. We'll assume that the priors for μ_1 and μ_2 are normal with means m_1, m_2 and precisions q_1, q_2, respectively. [NB: the precision q is just the reciprocal of the variance, i.e., $\sigma^2 = 1/q$.] With this assumption about the shape of the priors, we sample μ_1 and μ_2 using

$$\mu_1 \;\sim\; \text{normal}\left(\mu = \frac{m_1\, q_1 + s_1\, Q_1}{q_1 + n_1\, Q_1}, \text{prec} = Q_1\right) \qquad \mu_2 \;\sim\; \text{normal}\left(\mu = \frac{m_2\, q_2 + s_2\, Q_2}{q_2 + n_2\, Q_2}, \text{prec} = Q_2\right)$$
$$\text{where} \quad s_1 = \sum_{k:Z_k=1} X_k, \quad s_2 = \sum_{k:Z_k=2} X_k, \quad Q_1 = q_1 + \frac{n_1}{\sigma_1^2}, \quad Q_2 = q_2 + \frac{n_2}{\sigma_2^2}.$$

The code in Box 2 implements a Gibbs sampler for this model.

```
theta <- gibbs.mixture(theta=c(p=0.5,mu=c(0,0)),X=worm.data,niter=20)

plot(theta)
```




```
 theta <- gibbs.mixture(theta=c(p=0.5,mu=c(7,4)),X=worm.data,niter=4000)

 effectiveSize(theta)

       p     mu.1     mu.2
3020.526 2266.234 2964.898

 raftery.diag(theta)

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

      Burn-in  Total Lower bound  Dependence
      (M)      (N)   (Nmin)       factor (I)
 p    2        3866  3746         1.03
 mu.1 3        4374  3746         1.17
 mu.2 2        3866  3746         1.03

 theta <- gibbs.mixture(theta=theta[nrow(theta),],X=worm.data,niter=5000,thin=4)
```

```
 heidel.diag(theta)
```

```
     Stationarity start     p-value
     test         iteration
p    passed       1         0.666
mu.1 passed       1         0.543
mu.2 passed       1         0.892

     Halfwidth Mean  Halfwidth
     test
p    passed    0.201 0.00037
mu.1 passed    6.914 0.00142
mu.2 passed    3.924 0.00107
```

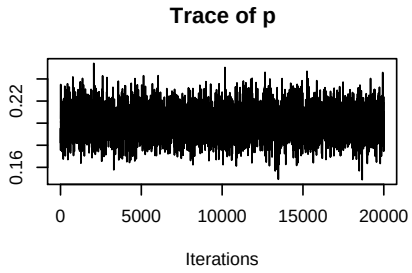
```
 geweke.diag(theta)
```

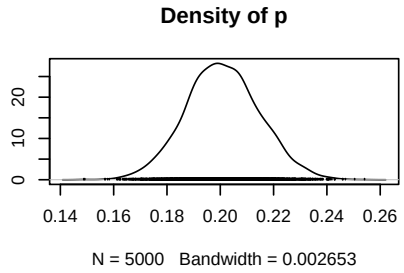
```
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

      p    mu.1    mu.2
 0.2144 -0.7851 -0.6544
```

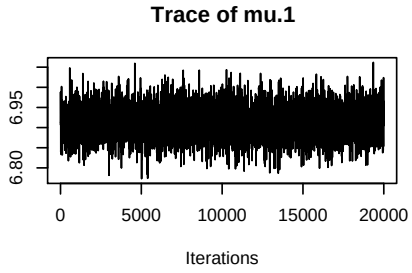
```
 pnorm(q=-abs(geweke.diag(theta)$z))
```

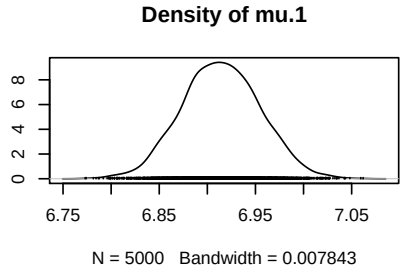
```
        p      mu.1      mu.2
0.4151111 0.2161924 0.2564301
```

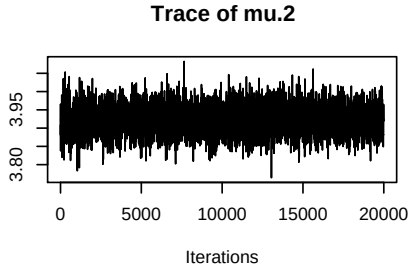
```
 plot(theta)
```

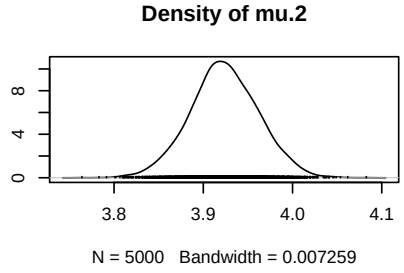








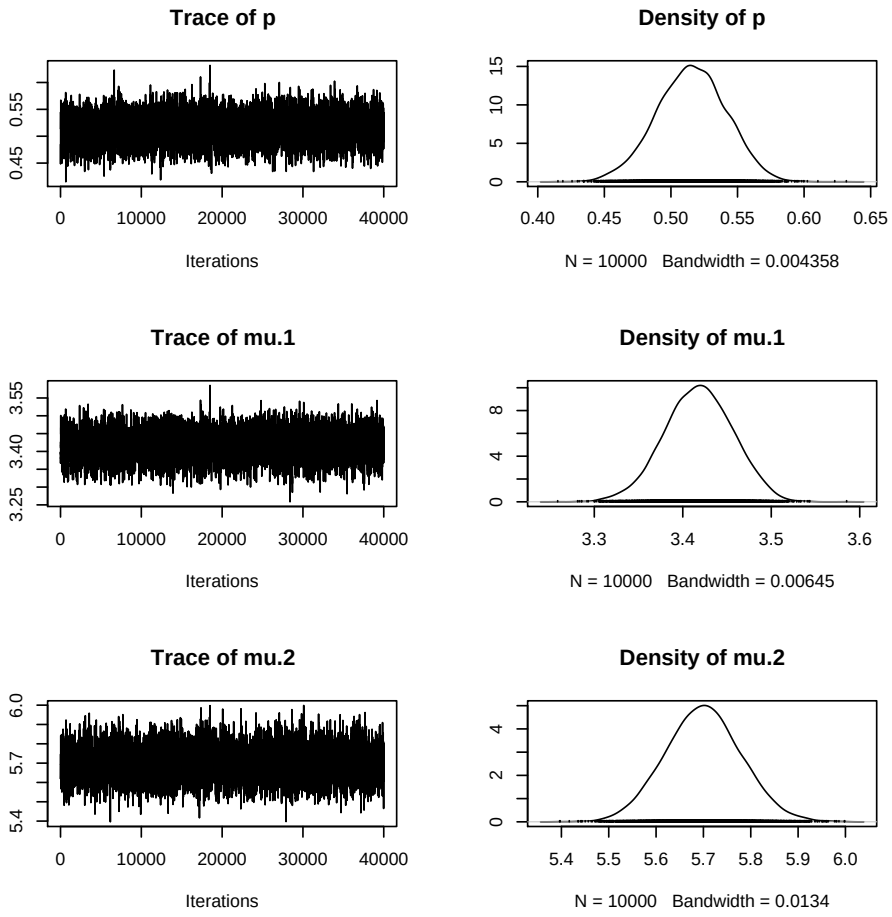





```
 summary(theta)

Iterations = 1:19997
Thinning interval = 4
Number of chains = 1
Sample size per chain = 5000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean      SD  Naive SE Time-series SE
p    0.2009 0.01387 0.0001962      0.0001886
mu.1 6.9143 0.04064 0.0005748      0.0007225
mu.2 3.9236 0.03820 0.0005403      0.0005463

2. Quantiles for each variable:

       2.5%    25%    50%    75%  97.5%
p    0.1743 0.1915 0.2006 0.2099 0.2288
mu.1 6.8370 6.8866 6.9138 6.9418 6.9937
mu.2 3.8486 3.8988 3.9230 3.9492 3.9990
```

We shouldn't be too glib, however. Suppose that we'd started from a different place:

```
theta <- gibbs.mixture(theta=c(p=0.5,mu=c(4,7)),X=worm.data,niter=4000)
theta <- gibbs.mixture(theta=theta[nrow(theta),],X=worm.data,niter=10000,thin=4)

plot(theta)
```




```
summary(theta)

Iterations = 1:39997
Thinning interval = 4
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean      SD  Naive SE Time-series SE
p    0.5154 0.02594 0.0002594      0.0004382
```

```
mu.1 3.4179 0.03840 0.0003840      0.0006090
mu.2 5.6998 0.07997 0.0007997      0.0012957

2. Quantiles for each variable:

       2.5%    25%    50%   75%  97.5%
p    0.4634 0.4978 0.5156 0.533 0.5644
mu.1 3.3407 3.3921 3.4182 3.444 3.4912
mu.2 5.5448 5.6457 5.6995 5.753 5.8575
```

In fact, this mixture example has a multimodal likelihood. In general, one has to be very careful assessing not only that a particular chain has converged, but that the global parameter space has been sufficiently well explored.

Exercise 3. Estimate the parameters of a normally distributed random variable on the basis of $n = 100$ independent and identically distributed samples using Gibbs. Parameterize the normal in terms of mean μ and precision q (= 1/variance). Take a normal prior for μ and a gamma prior for q. Modify the code in Box 2 to implement a Gibbs sampler for this model. Use the facts that, with this choice of priors

1. if $f(\mu)$ has mean m_0 and precision q_0, then $f(\mu|q, X)$ has mean $(q_0\, m_0 + q\, s)/(q_0 + n\, q)$ and precision $q_0 + n\, q$, where $s = \sum X_k$.
2. if $f(q)$ has shape parameter a_0 and rate parameter b_0, then $f(q|\mu, X)$ has shape $a_0 + \frac{n}{2}$ and rate $b + \frac{1}{2} \sum (X_k - \mu)^2$.

Simulate some data and run the sampler to estimate its mean and variance. You can view one solution in the file `http://www.umich.edu/~kingaa/EEID/Ecology/gibbs_mvnorm.R`.

Remark 10. In the examples here, we've assumed conjugate priors so that sampling from the conditional distribution is as easy as sampling from a standard distribution. Is it necessary to go this far? No. In the example Matt Ferrari will present, the parameters are updated one at a time, but the updates are not acheived by sampling from a standard distribution. Instead, he uses MH to sample from these conditional distributions. This is one variant of many within the family of MCMC algorithms.

4 Block and hybrid MCMC schemes

I've presented MH and Gibbs as two distict algorithms, Gibbs a very special case of MH, but they're actually the two extremes of a continuuum: between them lie a range of hybrid algorithms, more or less tailored to specific problems. Many of these involve grouping parameters into *blocks*. The key to sampling from a distribution $f(\theta_1, \ldots, \theta_n)$ using MH is the acceptance probability

$$\alpha = \frac{f(\theta^*)\, q(\theta^* \to \theta)}{f(\theta)\, q(\theta \to \theta^*)}$$

There is no need to propose changes to all of the θ_i at once, though, nor to make proposals one at a time, as in Gibbs sampling. Sometimes it makes a whole lot more sense to propose changing the parameters in blocks. For example, let's let θ_A stand for one block of variables and θ_B for the rest. Then

$$\frac{f(\theta_A^*, \theta_B)}{f(\theta_A, \theta_B)} = \frac{f(\theta_A^*|\theta_B)\, f(\theta_B)}{f(\theta_A|\theta_B)\, f(\theta_B)} = \frac{f(\theta_A^*|\theta_B)}{f(\theta_A|\theta_B)}$$

So we can take Metropolis-Hastings steps or Gibbs steps on block A, leaving block B fixed, then turn to block B, leaving block A fixed. In fact, there's no reason why we couldn't take many steps on one block

before turning to the other, nor why we couldn't divide the system up into more blocks, or even use different samplers on different blocks. In practice, it's rarely the case that the best-performing algorithm will be one that makes proposals to all parameters at once, as in the simplest version of MH. Nor is it likely that you will always be content to assume conjugate priors in order to get the speed of full Gibbs sampling. Rather, you're likely to want to use Gibbs where you can, coupled with some form of blocked Metropolis-Hastings, the details tailored to the problem at hand.

Further Reading

W. Gilks, S. Richardson, & D. Spiegelhalter (eds.) (1995). *Markov Chain Monte Carlo in Practice.* Chapman & Hall/CRC.

M. A. McCarthy (2007). *Bayesian Methods for Ecology.* Cambridge University Press.

C. P. Robert & G. Casella (2004). *Monte Carlo Statistical Methods.* Springer-Verlag, 2nd edn.

Box 1 Implementation of Metropolis-Hastings for a target distribution `fun` with starting value `theta`. It returns a Markov chain of length `niter` and, optionally, thins by a factor `thin` (with default value 1). Note that since the proposal distribution is symmetric, i.e., $q(\theta \to \theta^*) = q(\theta^* \to \theta)$, the Metropolis-Hastings acceptance probability is just $f(\theta^*)/f(\theta)$. You can download a somewhat enhanced version of this code from `http://www.umich.edu/~kingaa/EEID/Ecology/mh.R`.

```
require(coda)
mcmc.mh <- function (theta, fun, niter, thin = 1) {
  chain <- matrix(
                nrow=as.integer(niter),
                ncol=length(theta),
                dimnames=list(NULL,names(theta))
                )
  f <- fun(theta)
  chain[1,] <- theta
  k <- 1                                # step in the chain
  nsave <- 1                            # number of values saved so far
  nprop <- 0                            # number of proposals made
  nacpt <- 0                            # number of proposals accepted
  while (nsave < niter) {
    theta.star <- runif(n=1,min=theta-10,max=theta+10) # propose a new parameter
    f.star <- fun(theta.star)
    nprop <- nprop+1
    alpha <- f.star/f  # Metropolis' rule (since proposal is symmetric)
    if ((alpha>=1)||(runif(1)<alpha)) {
      theta <- theta.star
      f <- f.star
      nacpt <- nacpt+1
    }
    k <- k+1
    if (k%%thin==0) {
      nsave <- nsave+1
      chain[nsave,] <- theta
    }
  }
  cat("acceptance fraction = ",nacpt/nprop,"\n")
  mcmc(data=chain,thin=thin)            # store the result in a CODA object
}
```

Box 2 Implementation of a Gibbs sampler for a Bernoulli mixture of two normals. You can download it from `http://www.umich.edu/~kingaa/EEID/Ecology/gibbs_mixture.R`.

```
gibbs.mixture <- function (theta, X, niter, thin = 1) {
  ## use diffuse priors:
  ## mu ~ normal(m,1/sqrt(q)), m = 0, q = 0.0001
  ## p ~ uniform(0,1) = beta(a,b), a = b = 1
  prior.m <- c(mu.1=0,mu.2=0)           # mean of normal priors
  prior.q <- c(q1=0.0001,q2=0.0001)     # precision of normal priors
  prior.ab <- c(a=1,b=1)                # beta dist. shape parameters
  true.sigma <- c(sigma1=0.5,sigma2=1)  # normal SDs assumed known
  true.q <- 1/true.sigma^2              # precisions
  chain <- matrix(
                nrow=as.integer(niter),
                ncol=length(theta),
                dimnames=list(NULL,c("p","mu.1","mu.2"))
                )
  p <- theta[1]
  mu <- theta[-1]
  chain[1,] <- c(p,mu)
  k <- 1
  nsave <- 1
  while (nsave < niter) {
    ## first we choose Z | p,mu
    f1 <- dnorm(x=X,mean=mu[1],sd=true.sigma[1]) # Prob[X | Z==1,mu]
    f2 <- dnorm(x=X,mean=mu[2],sd=true.sigma[2]) # Prob[X | Z==2,mu]
    prob1 <- p*f1/(p*f1+(1-p)*f2)               # Prob[Z=1 | mu,p,X]
    Z <- ifelse(runif(n=length(X))<prob1,1,2)
    ## now choose mu | p,Z
    n <- c(sum(Z==1),sum(Z==2))
    s <- c(sum(X[Z==1]),sum(X[Z==2]))
    post.q <- prior.q+n*true.q
    post.m <- (prior.q*prior.m+true.q*s)/post.q
    mu <- rnorm(n=2,mean=post.m,sd=1/sqrt(post.q))
    ## now choose p | Z
    post.ab <- prior.ab+n
    p <- rbeta(n=1,shape1=post.ab[1],shape2=post.ab[2])
    k <- k+1
    if (k%%thin==0) {
      nsave <- nsave+1
      chain[nsave,] <- c(p,mu)
    }
  }
  mcmc(data=chain,thin=thin)            # store the result in a CODA object
}
```