Simple examples of the Metropolis-Hastings algorithm

Matthew Stephens
University of Chicago
January 23, 2026

See here for a PDF version of this vignette.

Prerequisites

You should be familiar with the Metropolis-Hastings algorithm, introduced here, and elaborated on here.

Caveat on code

Note that the code is designed to be readable by a beginner, rather than "efficient". The idea is that you can use this code to learn about the basics of MCMC, but not as a model for how to program well in R!

Example 1: Sampling from an exponential distribution using MCMC

Any MCMC scheme aims to produce (dependent) samples from a "target" distribution. In this case we are going to use the exponential distribution with mean 1 as our target distribution. Here we define this function (on the log scale):

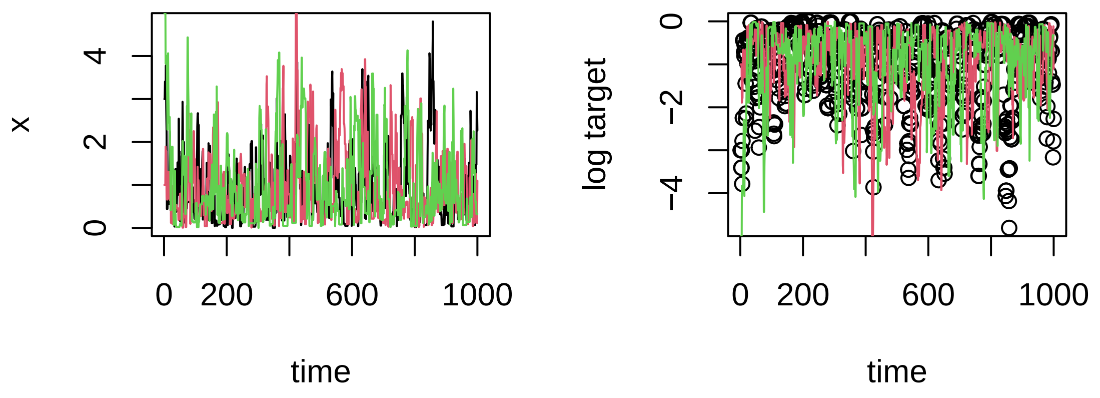
```
log_exp_target <- function (x)
  dexp(x,rate = 1,log = TRUE)
```

The following code implements a simple M-H algorithm. (Note that the parameter "log_target" is a function which computes the log of the target distribution; you may be unfamiliar with the idea of passing a function as a parameter, but it works just like any other type of parameter)

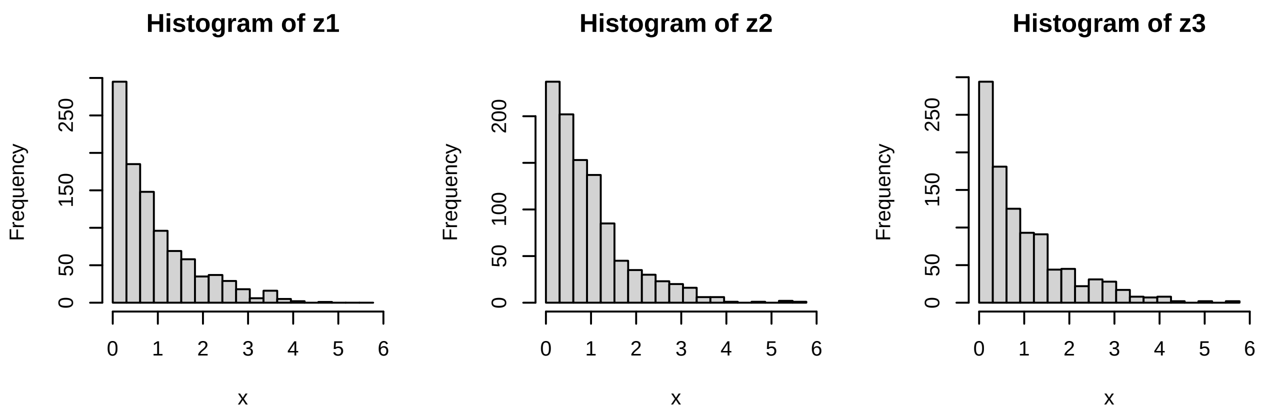
```
easyMCMC <- function (log_target, niter, startval, proposalsd) {
  x <- rep(0,niter)
  x[1] <- startval
  for (i in 2:niter) {
    currentx <- x[i-1]
    proposedx <- rnorm(1,currentx,proposalsd)
    A <- exp(log_target(proposedx) - log_target(currentx))
    u <- runif(1)
    if(u < A)
      x[i] <- proposedx
    else
      x[i] <- currentx
  }
  return(x)
}
```

Now we run the MCMC three times from different starting points and compare the results:

```
par(mfrow = c(1,2))
z1 <- easyMCMC(log_exp_target,1000,3,1)
z2 <- easyMCMC(log_exp_target,1000,1,1)
z3 <- easyMCMC(log_exp_target,1000,5,1)
plot(z1,type = "l",xlab = "time",ylab = "x")
lines(z2,col = 2)
lines(z3,col = 3)
plot(log_exp_target(z1),xlab = "time",ylab = "log target")
lines(log_exp_target(z2),col = 2)
lines(log_exp_target(z3),col = 3)
```



```
par(mfrow = c(1,3))
maxz <- max(c(z1,z2,z3))
hist(z1,breaks = seq(0,maxz,length.out = 20),xlab = "x")
hist(z2,breaks = seq(0,maxz,length.out = 20),xlab = "x")
hist(z3,breaks = seq(0,maxz,length.out = 20),xlab = "x")
```



Exercise

Use the function “easyMCMC” to explore the following:

a. How do different starting values affect the MCMC scheme? (Try some extreme starting points.)
b. What is the effect of having a bigger (or smaller) proposal standard deviation? (Again, try some extreme values.)
c. Try changing the (log) target function to “log_target_bimodal” below. What does this target distribution look like? What happens if the proposal standard deviation is too small here? (e.g., try 1 and 0.1 and see what happens)

```r
log_target_bimodal <- function (x)
  log(0.8*dnorm(x,-4,1) + 0.2*dnorm(x,4,1))
```

Example 2: Estimating an allele frequency

A standard assumption when modelling genotypes of biallelic loci (e.g., loci with alleles A and a) is that the population is “mating at random”. From this assumption, it follows that the population will be in “Hardy-Weinberg equilibrium” (HWE), which means that if p is the frequency of the A allele, then the genotypes AA, Aa and aa have frequencies p^2, $2p(1-p)$ and $(1-p)^2$, respectively.

A simple prior for p is to assume it is uniform on $[0,1]$. Suppose that we sample n individuals, and observe n_{AA} individuals with genotype AA, n_{Aa} individuals with genotype Aa, and n_{aa} individuals with genotype aa.

The following R code implements a M-H algorithm to sample from the posterior distribution of p. Try to go through the code to see how it works.

The first function returns the (log) prior density:

```r
log_prior <- function (p) {
  if((p < 0) || (p > 1))
    return(-Inf)
  else
    return(0)
}
```

The second function returns the (log) likelihood:

```r
log_likelihood_hwe <- function (p, nAA, nAa, naa)
  return(2*nAA*log(p) +
              nAa*log(2*p*(1-p)) +
              2*naa*log(1-p))
```

And now this function implements the M-H algorithm:

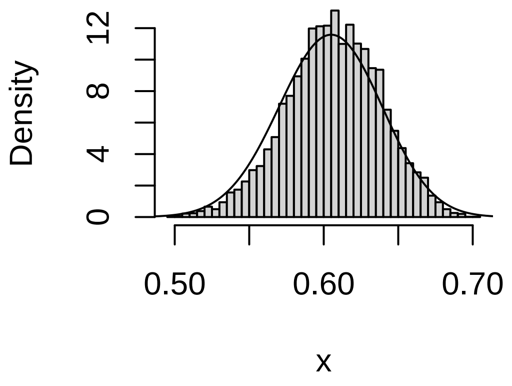
```
psampler <- function (nAA, nAa, naa, niter, pstartval, pproposalsd) {
  p <- rep(0,niter)
  p[1] <- pstartval
  for (i in 2:niter) {
    currentp <- p[i-1]
    newp <- currentp + rnorm(1,0,pproposalsd)
    A <- exp(log_prior(newp)
             - log_prior(currentp)
             + log_likelihood_hwe(newp,nAA,nAa,naa)
             - log_likelihood_hwe(currentp,nAA,nAa,naa))
    if (runif(1) < A)
      p[i] <- newp
    else
      p[i] <- currentp
  }
  return(p)
}
```

Try running this algorithm for $n_{AA} = 50$, $n_{Aa} = 21$, $n_{aa} = 29$:

```
z <- psampler(50,21,29,10000,0.5,0.01)
```

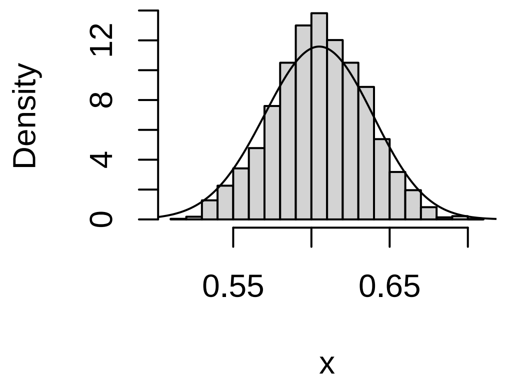
Now here's some R code to compare the samples generated by the M-H algorithm with the theoretical posterior. In this case, the theoretical posterior is available analytically; since we observed A 121 times and a 79 times, the posterior for p is $\text{Beta}(121+1, 79+1)$.

```
x <- seq(0,1,length.out = 1000)
hist(z,breaks = 32,probability = TRUE,xlab = "x",main = "")
lines(x,dbeta(x,122,80))
```



You might also like to discard the first 5,000 samples as "burnin":

```
hist(z[5001:10000],breaks = 16,probability = TRUE,xlab = "x",main = "")
lines(x,dbeta(x,122,80))
```



Exercise

Investigate how the starting point and proposal standard deviation affect the convergence of the algorithm.

Example 3: Estimating an allele frequency and inbreeding coefficient

A slightly more complex alternative than HWE is to assume that there is a tendency for people to mate with others who are slightly more related than "random" (as might happen in a geographically structured population, for example). This will result in an excess of homozygotes compared with HWE. A simple way to capture this is to introduce an extra parameter, the "inbreeding coefficient" f, and assume that the genotypes AA, Aa and aa have frequencies $fp + (1-f)p^2$, $2(1-f)p(1-p)$, and $f(1-p) + (1-f)(1-p)^2$, respectively.

In most cases, it would be natural to treat f as a feature of the population, and therefore assume f is constant across loci. For simplicity, we will consider just a single locus.

Note that both f and p are constrained to lie between 0 and 1 (inclusive). A simple prior for each of these two parameters is to assume that they are independent and uniform on $[0, 1]$.

Suppose that we sample n individuals, and observe n_{AA} individuals with genotype AA, n_{Aa} individuals with genotype Aa, and n_{aa} individuals with genotype aa.

Exercise

1. Write an M-H algorithm to sample from the joint distribution of f and p. *Hint:* Below is some code to get you started. (You'll need to fill in the "...") As a first step, I suggest writing the function to compute the log-likelihood for the inbreeding model.

```
log_likelihood_inbreeding <- function (...) {
  # ...
}


fpsampler <- function (nAA, nAa, naa, niter, fstartval, pstartval,
                       fproposalsd, pproposalsd) {
  f <- rep(0,niter)
  p <- rep(0,niter)
  f[1] <- fstartval
  p[1] <- pstartval
  for (i in 2:niter) {
    currentf <- f[i-1]
    currentp <- p[i-1]
    # newf <- currentf + ...
    # newp <- currentp + ...
    # ...
  }
  return(list(f = f,p = p))
}
```

2. Use the output of your M-H algorithm to compute point estimates for f and p (e.g., posterior means) as well as interval estimates for f and p (e.g., 90% posterior credible intervals) when the data are $n_{AA} = 50$, $n_{Aa} = 21$, $n_{aa} = 29$.

Addendum: Gibbs sampling

You could also tackle this problem with a Gibbs sampler (see also the gibbs1 and gibbs2 vignettes). To do so, you will want to use the following "latent variable" representation of the model:

$$\begin{aligned}
\Pr(g_i = AA \mid z_i = 0) &= p^2 \\
\Pr(g_i = Aa \mid z_i = 0) &= 2p(1-p) \\
\Pr(g_i = aa \mid z_i = 0) &= (1-p)^2,
\end{aligned}$$

$$\begin{aligned}
\Pr(g_i = AA \mid z_i = 1) &= p \\
\Pr(g_i = Aa \mid z_i = 1) &= 0 \\
\Pr(g_i = aa \mid z_i = 1) &= 1-p,
\end{aligned}$$

and

$$z_i \sim \text{Bernoulli}(f).$$

Note that summing over z_i gives the same model as above, e.g., $\Pr(g_i = AA) = fp + (1-f)p^2$. (As an exercise, show that the probabilities of $g_i = aa$ and $g_i = Aa$ from this "latent variable augmented" model are the same as above.)

Question: What does the z_i latent variable represent?

Exercise

Using the latent variable representation above, implement a Gibbs sampler to generate samples from the joint distribution of z, f and p given the genotype data g. *Hint:* this involves iterating the following steps:

1. Sample z from $p(z \mid g, f, p)$.
2. Sample f, p from $p(f, p \mid g, z)$.