Bayesian Statistics XI

Simple Examples of Metropolis–Hastings Algorithm

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 log scale):

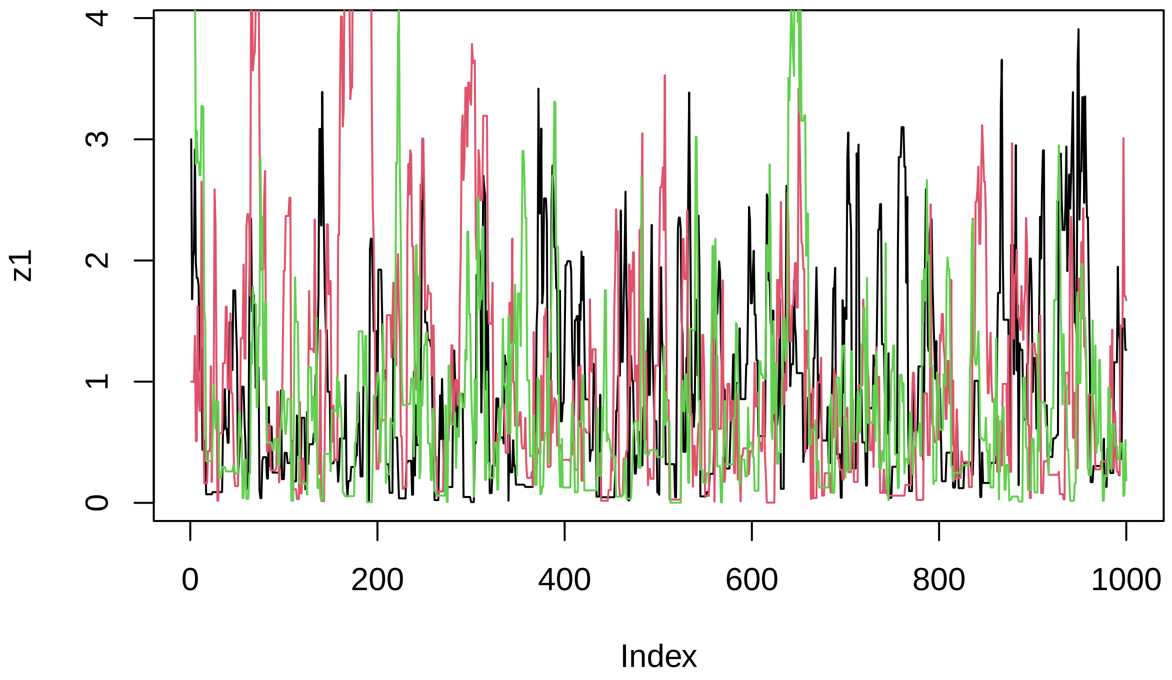
```
log_exp_target <- function(x){
  return(dexp(x,rate=1, log=TRUE))
}
```

The following code implements a simple MH 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. . .):

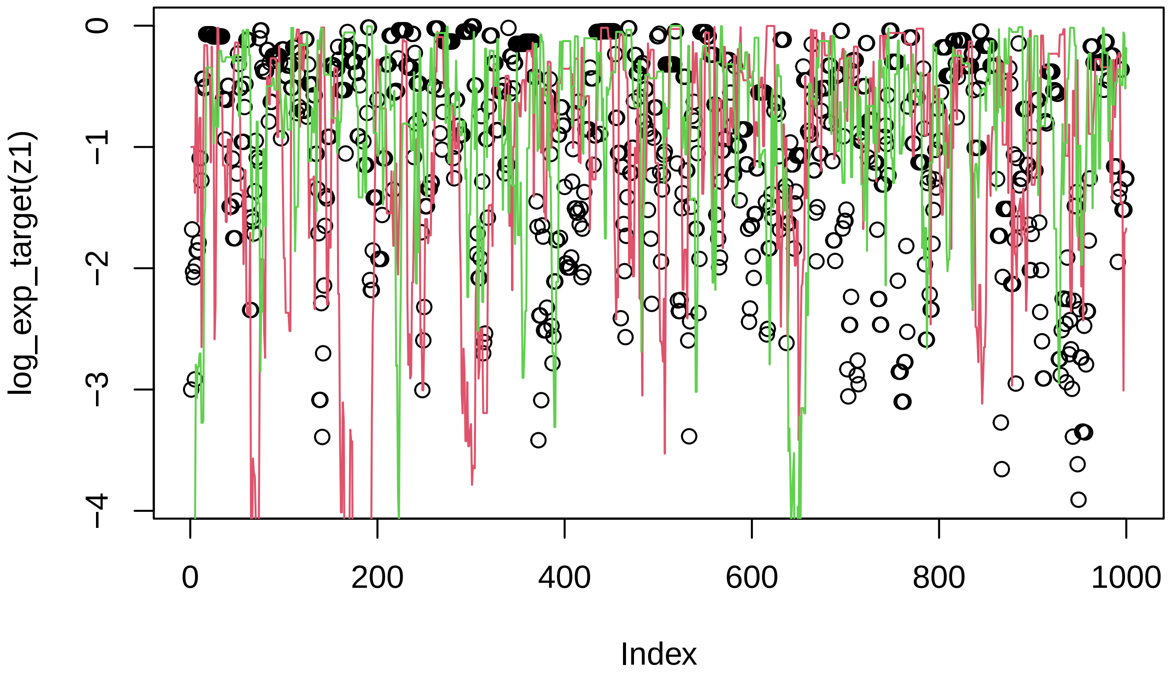
```
easyMCMC <- function(log_target, niter, startval, proposalsd){
  x <- rep(0,niter)
  x[1] <- startval
  for(i in 2:niter){
    currentx <- x[i-1]
    proposedx <- rnorm(1, mean=currentx, sd=proposalsd)
    A <- exp(log_target(proposedx) - log_target(currentx))
    if(runif(1)<A){
      x[i] <- proposedx       # accept move with probabily min(1,A)
    } else {
      x[i] <- currentx        # otherwise "reject" move, and stay where we are
    }
  }
  return(x)
}
```

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

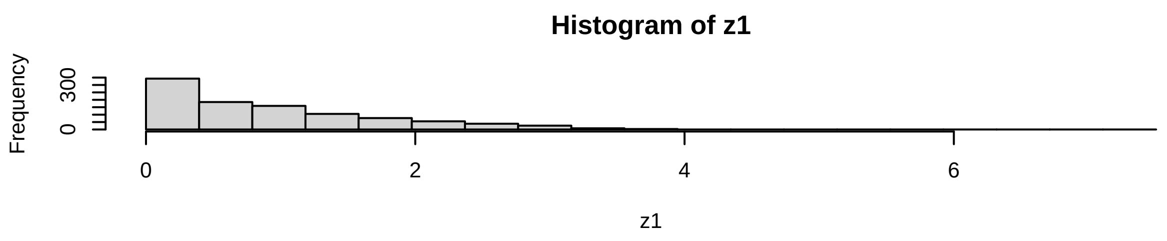
```
z1 <- easyMCMC(log_exp_target, 1000,3,1)
z2 <- easyMCMC(log_exp_target, 1000,1,1)
z3 <- easyMCMC(log_exp_target, 1000,5,1)

plot(z1,type="l")
lines(z2,col=2)
lines(z3,col=3)
```

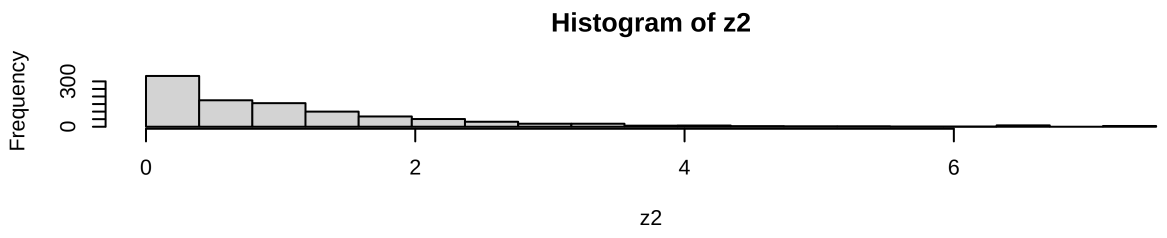


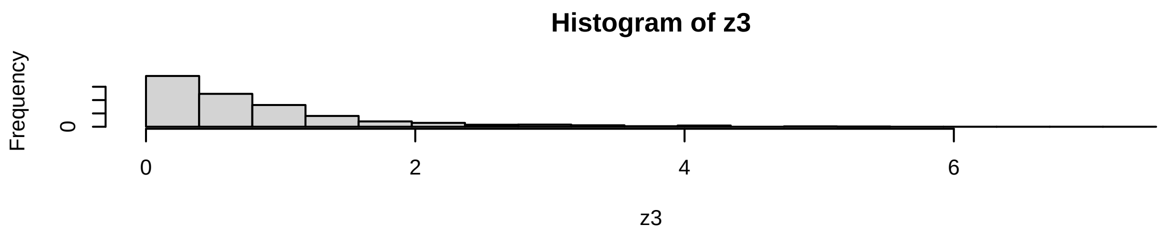
```r
plot(log_exp_target(z1))
lines(log_exp_target(z2),col=2)
lines(log_exp_target(z3),col=3)
```



```r
par(mfcol=c(3,1)) #rather odd command tells R to put 3 graphs on a single page
maxz <- max(c(z1,z2,z3))
hist(z1,breaks=seq(0,maxz,length=20))
hist(z2,breaks=seq(0,maxz,length=20))
hist(z3,breaks=seq(0,maxz,length=20))
```







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/smaller proposal standard deviation? (again, try some extreme values)
c) try changing the (log-)target function to the following

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

What does this target distribution look like? What happens if the proposal sd is too small here? (try e.g. 1 and 0.1)

Example 2: Estimating an allele frequency

A standard assumption when modelling genotypes of bi-allelic loci (e.g. loci with alleles A and a) is that the population is "randomly mating". 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 allele A then the genotypes AA, Aa and aa will 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} with genotype AA, n_{Aa} with genotype Aa and n_{aa} with genotype aa.

The following R code gives a short MCMC routine to sample from the posterior distribution of p. Try to go through the code to see how it works.

```
log_prior <- function(p){
  if((p<0) || (p>1)){  # || here means "or"
```

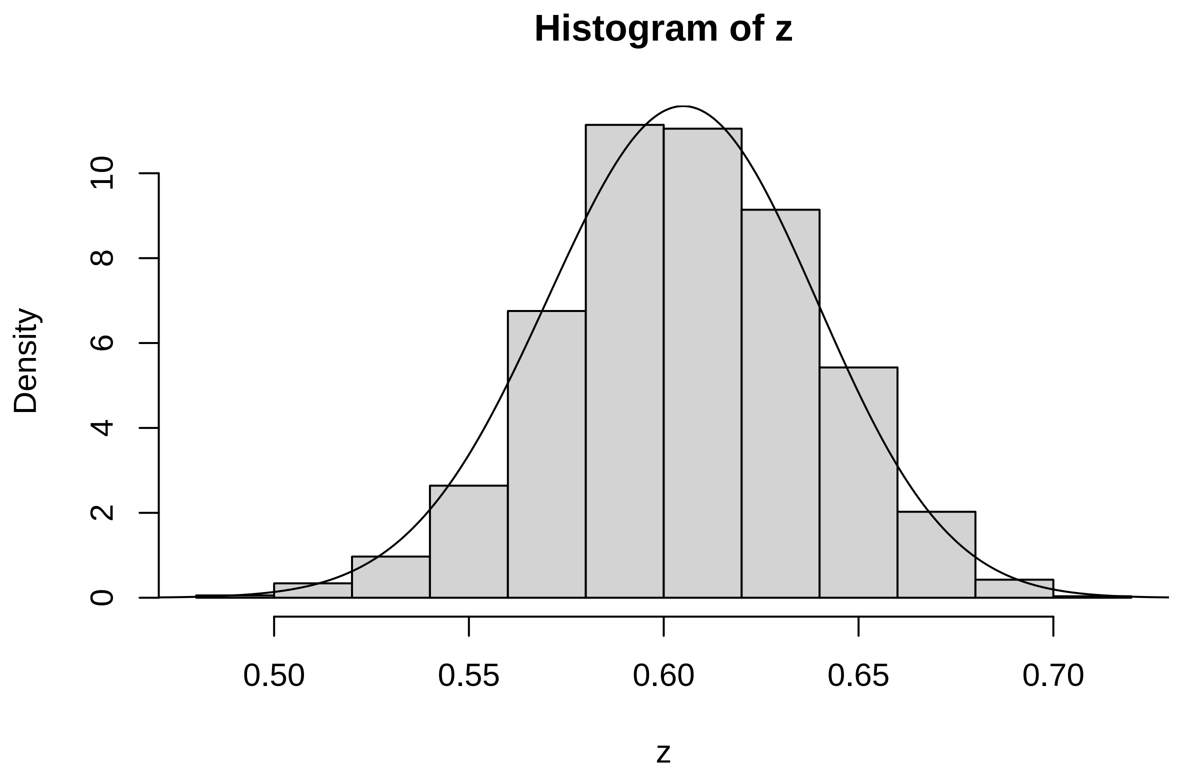
```
    return(-Inf)}
  else{
    return(0)}
}

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

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_likelihood_hwe(newp,nAA,nAa,naa) - log_prior(currentp) - log_likelih
    if(runif(1)<A){
      p[i] <- newp       # accept move with probabily min(1,A)
    } else {
      p[i] <- currentp        # otherwise "reject" move, and stay where we are
    }
  }
  return(p)
}
```

Running this sample for $n_{AA} = 50$, $n_{Aa} = 21$, n_{aa}=29.

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

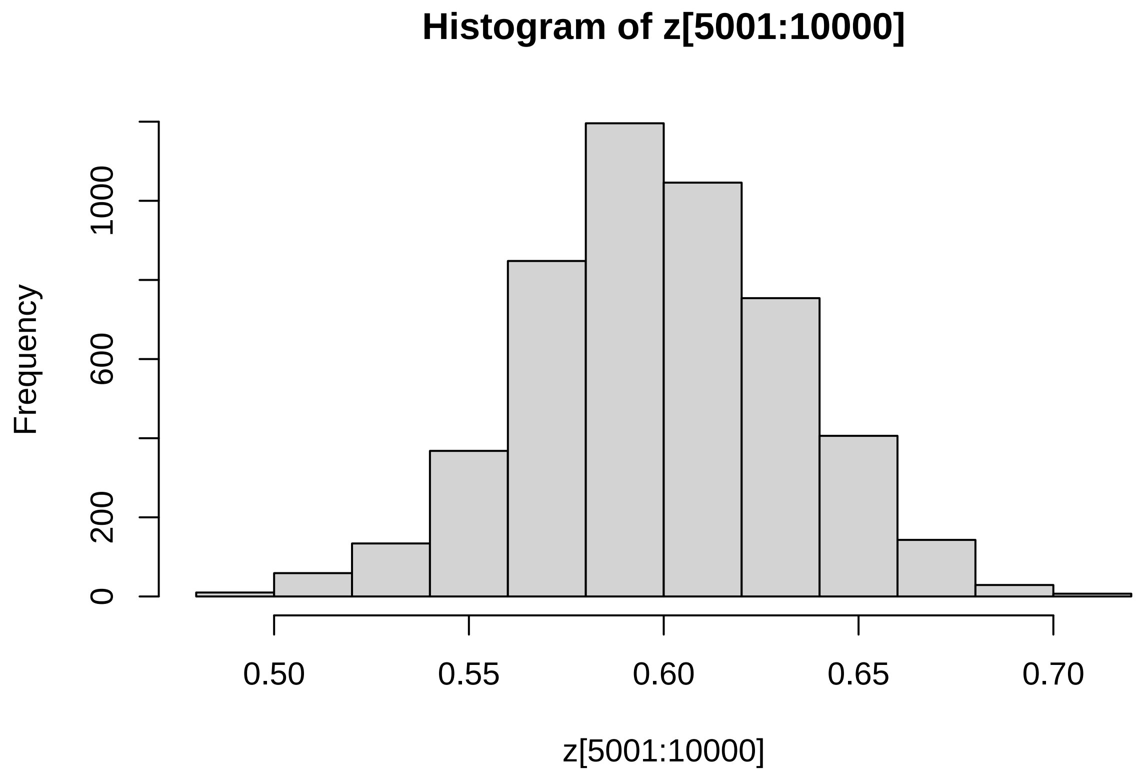
Now some R code to compare the sample from the posterior with the theoretical posterior (which in this case is available analytically; since we observed 121 As, and 79 as, out of 200, the posterior for p is Beta(121+1,79+1).

```
x <- seq(0,1,length=1000)
hist(z,prob=T)
lines(x,dbeta(x,122, 80))  # overlays beta density on histogram
```



You might also like to discard the first 5000 z's as "burnin". Here's one way in R to select only the last 5000 z's

```
hist(z[5001:10000])
```



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 closely-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 $fp + (1-f)p*p, (1-f)2p(1-p)$, and $f(1-p) + (1-f)(1-p)(1-p)$.

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, uniform on $[0, 1]$. Suppose that we sample n individuals, and observe n_{AA} with genotype AA, n_{Aa} with genotype Aa and n_{aa} with genotype aa.

Exercise:

- Write a short MCMC routine to sample from the joint distribution of f and p.

Hint: here is a start; you'll need to fill in the ...

```
# The first step is probably to code a log-likelihood function for the inbreeding model....
log_likelihood_inbreeding <- function(...){
  ...
}

# then use the log-likelihood within your MCMC scheme
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)) # return a "list" with two elements named f and p
}
```

- Use this sample to obtain point estimates for f and p (e.g. using posterior means) and interval estimates for both 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.

To do so you will want to use the following "latent variable" representation of the model:

$$z_i \sim Bernoulli(f)$$

$$p(g_i = AA|z_i = 1) = p; p(g_i = AA|z_i = 0) = p^2$$

$$p(g_i = Aa|z_i = 1) = 0; p(g_i = Aa|z_i = 0) = 2p(1-p)$$
$$p(g_i = aa|z_i = 1) = (1-p); p(g_i = aa|z_i = 0) = (1-p)^2$$

Summing over z_i gives the same model as above:

$$p(g_i = AA) = fp + (1-f)p^2$$

Exercise:

Using the above, implement a Gibbs Sampler to sample from the joint distribution of z, f, and p given genotype data g.

Hint: this requires iterating the following steps

1) sample z from $p(z|g, f, p)$
2) sample f, p from $p(f, p|g, z)$