Lab Week 11 : Markov Chain Monte Carlo methods

STAT5003

Dr. Justin Wishart

Semester 2, 2022

Contents

Preparation and assumed knowledge

- Viewed the Makrov Chain Monte Carlo methods content in Module 10.

Aims

- Implement Markov Chain Monte Carlo techniques to
  - Compute a posterior distribution
  - Generate samples from a non-trivial density

Exploring the Markov Chain Monte Carlo method. There isn't any dataset in today's tutorial, only simulated examples.

1 Compute chance of a coin flip

Estimate the P(Head) of a biased coin. In particular, estimate the posterior probability P(Head|Observed Data). Start by simulating some coin flips, where we know p =P(Head). Then, use MCMC to see if p can be estimated from the data of the coin flips.

1.1 Simulate some data

Below is a way to simulate coin flips with chance of success $\theta = p = 0.7$ but you can use other simulated data if you wish. Denote these simulated values as the random variables $X_1, X_2, \ldots, X_n$ where $n = 200$

```
set.seed(5003)
theta = 0.7 # P(Head)
N = 200
```

```r
# simulate some coin flips
coin.flips <- rbinom(n = N, size = 1, prob = theta)
```

1.2 Determine how to compute the Likelihood of the parameter

Recall the Likelihood approach covered in a earlier module. In this case, each observation is assumed to be drawn from a Bernoulli (Binomial with $n = 1$) distribution. The formula is,

$$P(X = x|\theta) = \begin{cases} \theta^x(1-\theta)^{(1-x)}, & \text{if } x = 0 \text{ or } 1; \\ 0, & \text{otherwise.} \end{cases}$$

and it can be computed with `dbinom`. Write a function that computes the Likelihood of the parameter θ given all of the observed data.

$$L(\theta|X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X = x_i|\theta)$$

```r
likelihood <- function(theta, x)
  prod(dbinom(x, size = 1, prob = theta))
```

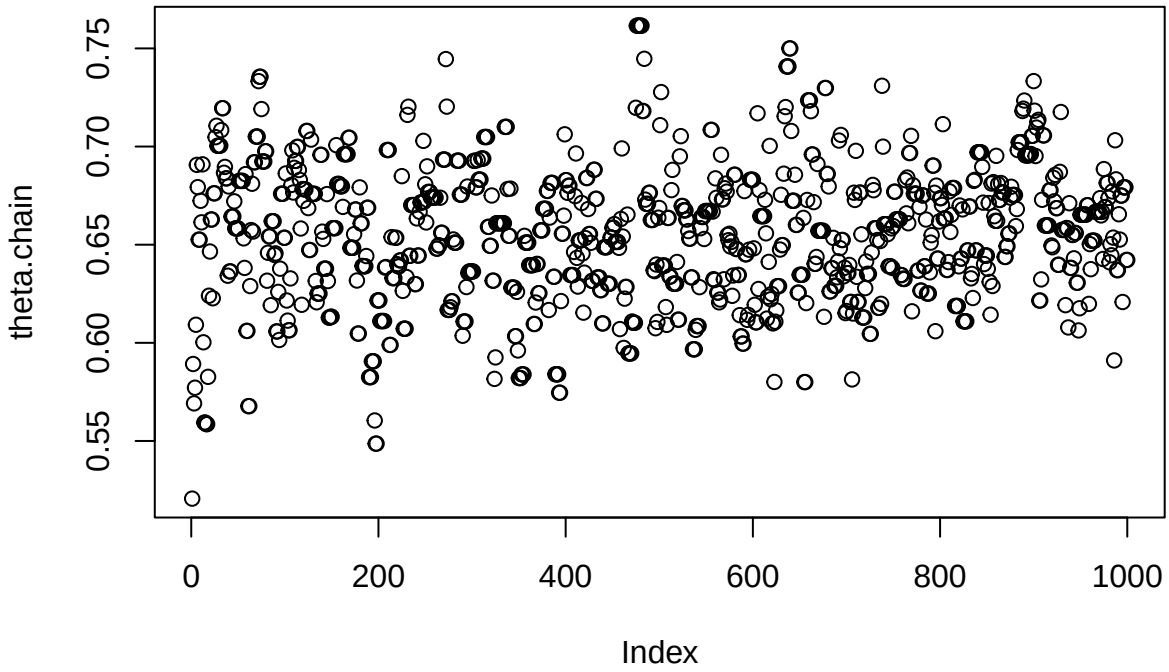
1.3 Naive Metropolis Hastings

Write a Metropolis Hastings algorithm that takes a specified initial value for the parameter θ (or you can make it random if you wish) and then proposes a new value with a random walk sampler. For simplicity it is recommended to use the Gaussian density as the proposal density q. Psuedo-code of the Metropolis Hasting algorithm is given below. It needs some work that is obvious, other work required are the proposed theta needs to be between 0 and 1, it will need some code to ensure an impossible theta isn't generated.

```r
# In this example, we set the prior of P(Head) to be a uniform distribution
niters <- 1000
curr.theta <- 0.5 # This is the parameter we want to find; initialise it

theta.chain <- numeric(niters) # Initialise the vector to be created
for(i in 0:(niters - 1)) {
  proposal.theta <- ... # A random draw from the proposal density that is conditioned on the previous i

  curr.likeli <- ... # computed likelihood from the current theta
  proposal.likeli <- ... # computed likelihood from the proposed theta

  alpha <- min(..., 1)

   # Randomly decide whether to accept our new proposal
   if (runif(1) < alpha)
        curr.theta <- ...
   theta.chain[i + 1] <- ...
}
```

```r
niters <- 1000
curr.theta <- 0.5 # This is the parameter we want to find; initialise it

theta.chain <- numeric(niters) # Initialise the vector to be created
for(i in 0:(niters - 1)) {
  proposal.theta <- curr.theta + rnorm(1, mean = 0, sd = 0.05) # A random draw from the proposal densit
```

```
  if (proposal.theta < 0 || proposal.theta > 1)
  {
      theta.chain[i + 1] <- curr.theta
      next
  }

  curr.likeli <- likelihood(curr.theta, coin.flips) # computed likelihood from the current theta
  proposal.likeli <- likelihood(proposal.theta, coin.flips) # computed likelihood from the proposed the

  alpha <- min(proposal.likeli/curr.likeli, 1)

   # Randomly decide whether to accept our new proposal
   if (runif(1) < alpha)
        curr.theta <- proposal.theta
   theta.chain[i + 1] <- curr.theta
}
plot(theta.chain)
```



There was a typo in the original version of this that made the chain above not converge. It now works quite well and the remaining analysis of this question not required. However, if hypothetically, the computed likelihoods are too small for the machine to handle accurately (division by a small value can be unstable). Then, an alternative it to compute the product and division of the likelihood in the log domain (where it is a sum!)

1.4 Log Likelihood computation

The log likelihood can be computed to stabilize the computation above where the computed likelihood values fall below the `.Machine$double.eps`, roughly speaking the smallest possible value that the computer can handle with precision. (see `? .Machine`).

Write a function that computes the log-Likelihood of the parameter θ given all of the observed data.

$$\mathcal{L}(\theta|X_1, X_2, \ldots, X_n) = \log L(\theta|X_1, X_2, \ldots, X_n) = \log(\prod_{i=1}^{n} P(X = x_i|\theta)) = \sum_{i=1}^{n} \log P(X = x_i|\theta)$$

Hint: The argument of `log` is useful in the `dbinom` function. (See `? dbinom`).

```
log.likelihood <- function(theta, x)
  sum(dbinom(x, size = 1, prob = theta, log = TRUE))
```

1.5 Log likelihood computed approach to the Metropolis-Hasting algorithm

Amend the Metropolis-Hastings algorithm above to use the log likelihood instead. Both the likelihood computations will now need to compute the log likelihood and the acceptance probability α will need to be modified.

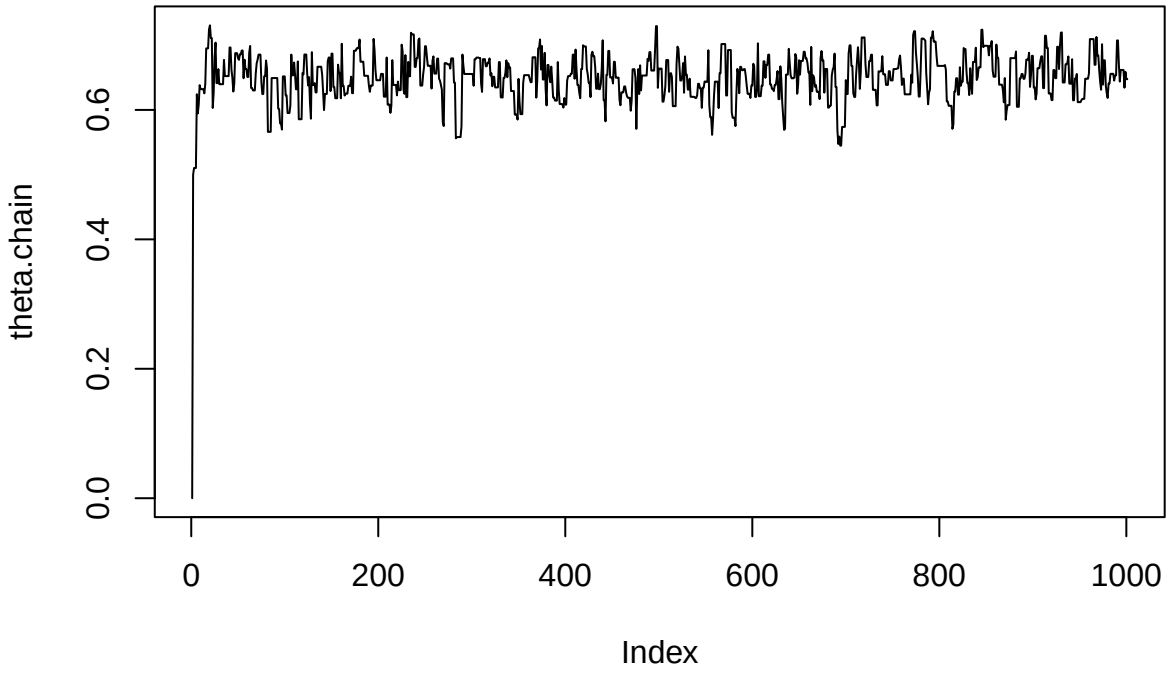
```
niters <- 1000
curr.theta <- 0.5 # This is the parameter we want to find; initialise it

theta.chain <- numeric(niters) # Initialise the vector to be created
for(i in 0:(niters - 1)) {
  proposal.theta <- ... # A random draw from the proposal density that is conditioned on the previous i

  curr.log.likeli <- ... # computed log likelihood from the current theta
  proposal.log.likeli <- ... # computed log likelihood from the proposed theta

  alpha <- min(..., 1)  # This needs to be updated (hint, exp cancels out log)

   # Randomly decide whether to accept our new proposal
   if (runif(1) < alpha)
        curr.theta <- ...
   theta.chain[i + 1]
}
```

```
# In this example, we set the prior of P(Head) to be a uniform distribution
niters <- 1000
curr.theta <- 0.5 # This is the parameter we want to find; initialise it

theta.chain <- numeric(niters)
for(i in 1:niters) {
  proposal.theta <- curr.theta + rnorm(1, mean = 0, sd = 0.05)

  # If proposed theta leaves the feasible range of values
  if(proposal.theta > 1 || proposal.theta < 0) {
     theta.chain[i + 1] <- curr.theta
     next
  }

  # It is easier to work with log of probabilities
  curr.log.likeli <- log.likelihood(curr.theta, coin.flips)
  proposal.log.likeli <- log.likelihood(proposal.theta, coin.flips)

  # Because we have taken log(prob), now we need a slightly different
  # formula to get the acceptance probability. Follows from exp(log(a/b)) = exp(log(a) - log(b))
  alpha <- min(exp(proposal.log.likeli - curr.log.likeli), 1)
```

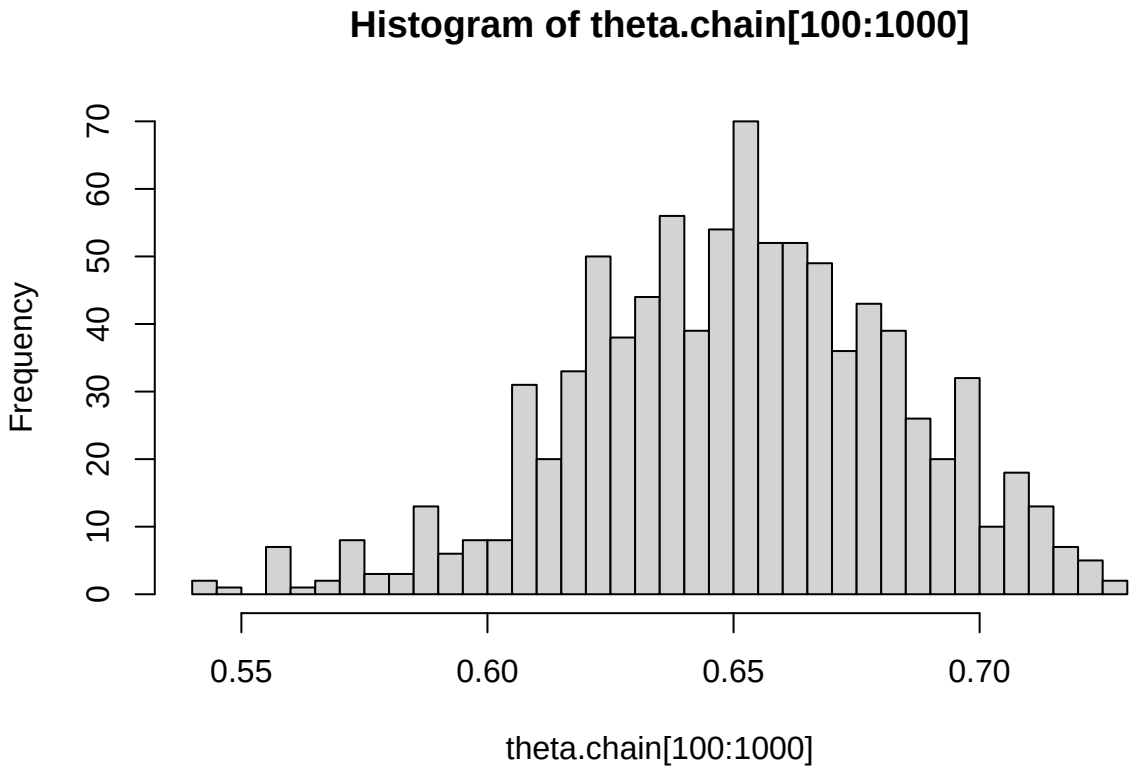
```
    # Randomly decide whether to accept our new proposal
    if (runif(1) < alpha)
         curr.theta <- proposal.theta
    theta.chain[i + 1] <- curr.theta
}
```

You can see from the plot that the parameter of theta converges to around 0.7. The first part of the plot is the burn-in. We should discard it and use the rest of the chain to estimate theta. This plot looks like a hairy caterpillar which means it has converged properly.

```
plot(theta.chain, type = "l")
```




```
hist(theta.chain[100:1000], n = 30)
```



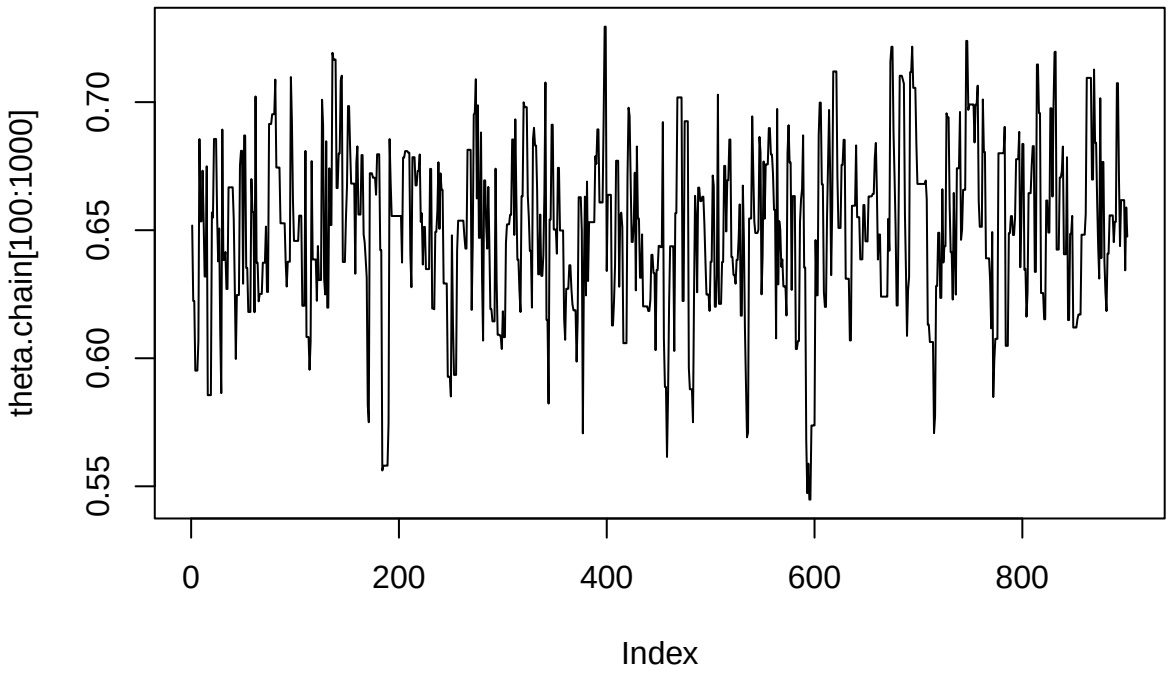

```
# MCMC estimates of theta after 1000 iterations
print(mean(theta.chain[100:1000]))
```

```
## [1] 0.6506242
```

```
print(sd(theta.chain[100:1000]))
```

```
## [1] 0.03286099
```

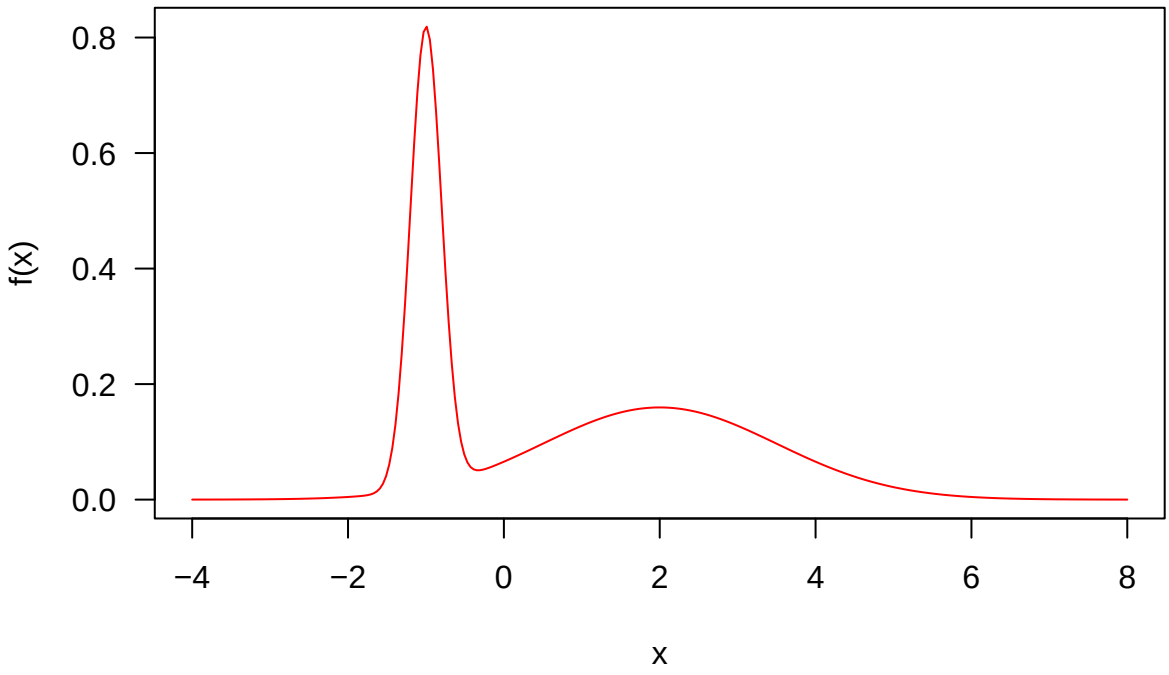
```
plot(theta.chain[100:1000], type = 'l')
```



2 Generate Samples from a non-trivial distribution

Use the MCMC procedure to generate random samples from the following probability distribution function. This function is simply the sum of two normal distributions. Explore the use of the proposal distribution to explore the target space in the MCMC algorithm.

```
p <- 0.4
mn <- c(-1, 2)
std <- c(0.2, 1.5)
f <- function(x, mu = mn, sd = std)  p * dnorm(x, mu[1], sd[1]) + (1 - p) * dnorm(x, mu[2], sd[2])

curve(f(x), col = "red", from = -4, to = 8, n = 300, las = 1)
```



2.1 Use Metropolis-Hastings algorithm to draw samples from the normal mixture above

Use the Gaussian density centered on the current point as the proposal transition. Consider the random walk Metropolis-Hastings algorithm again (symmetric proposal distribution). Define a function q in `R` to be a sample from a Gaussian distribution but with two parameters, x to denote the current value of the markov chain and σ to be the sd of the Gaussian distribution.

```
q <- function(x, sigma) ...(1, mean = ..., sd = ...)
```

```
q <- function(x, sigma) rnorm(1, mean = x, sd = sigma)
```

2.2 Function to do a random walk Metropolis-Hasting algorithm

Write a function that uses a random walk Metropolis-Hastings sampling of the target distribution `f` that uses your proposal function `q` but has argument, `sigma` to control the sd parameter of the proposal distribution. i.e. your function should have signature.

```
#' x_0 : Initial value of the Markov chain
#' n : number of iterations to compute
#' sigma : size of the sd in the proposal distribution
function(x_0, n, sigma)
```

```
# Function that takes a starting value and a number of iterations to compute
mc.algorithm <- function(x_0, n, sigma)
{
    x <- numeric(n)
    x[1] <- x_0
    for (i in 1:(n - 1))
    {
        prop.x <- q(x[i], sigma = sigma)
        alpha <- min(f(prop.x)/f(x[i]), 1)
        if (runif(1) < alpha)
            x[i + 1] <- prop.x
```

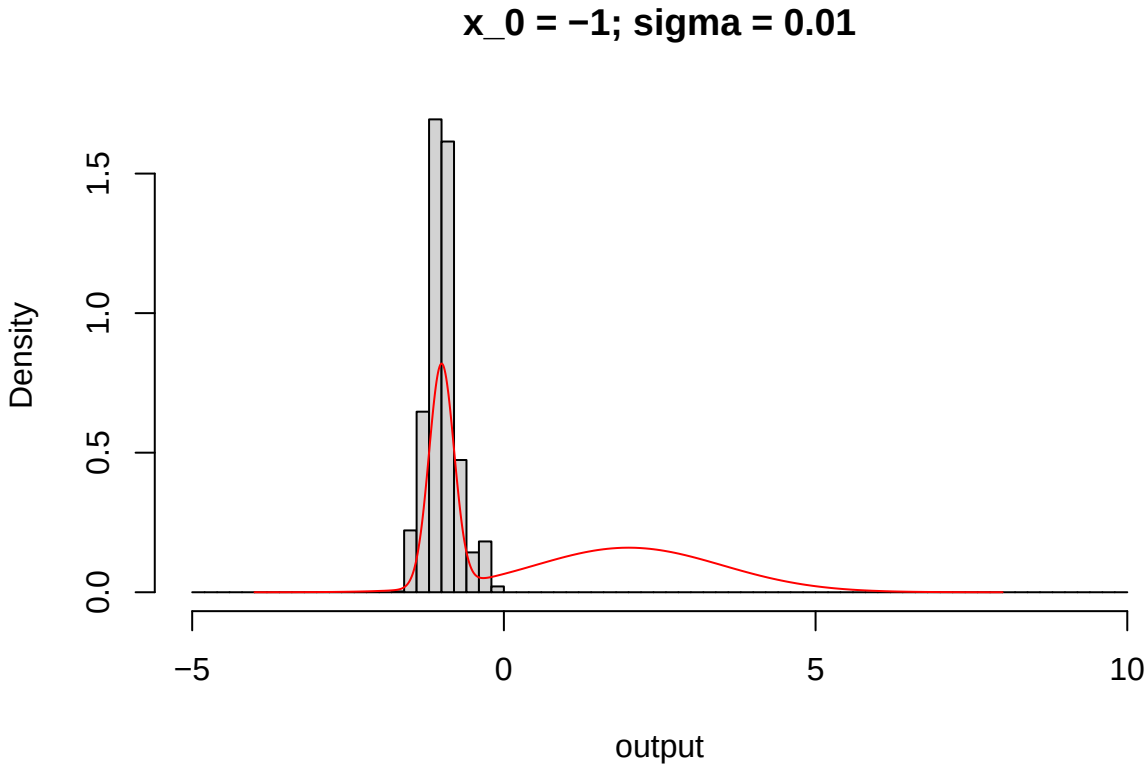
```
        else
            x[i + 1] <- x[i]
    }
    x
}
```

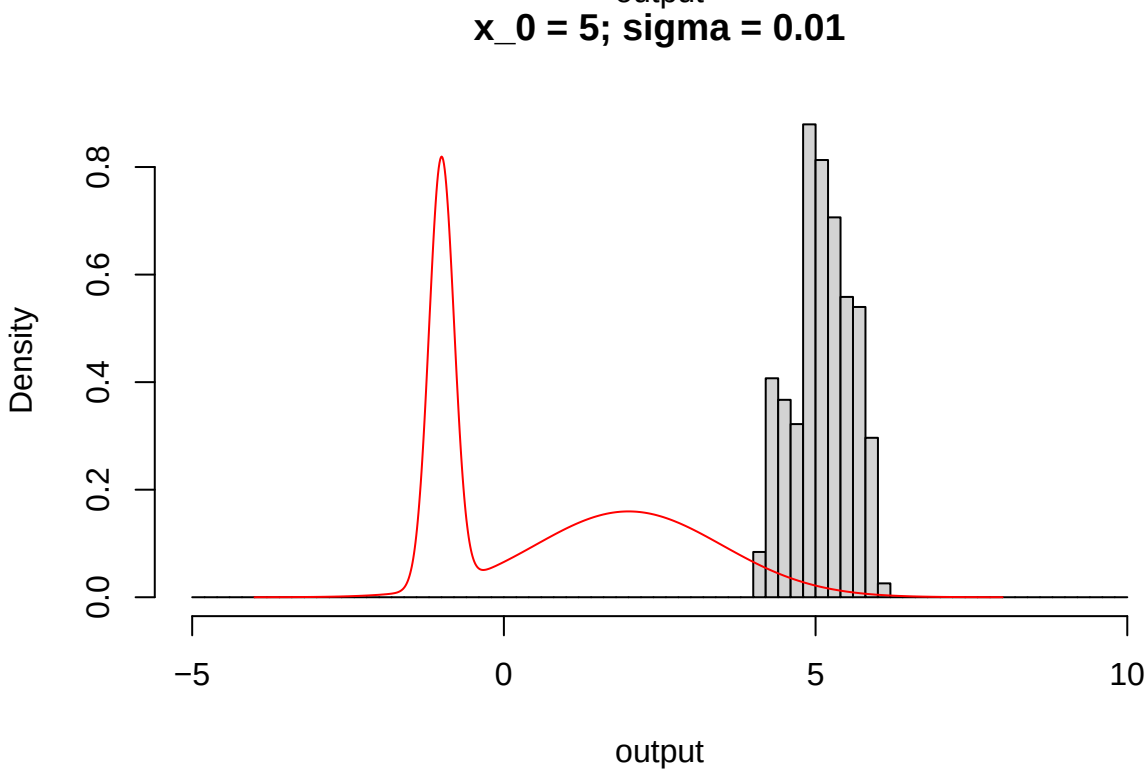
2.3 Explore the impact of choices of `x_0` and `sigma`

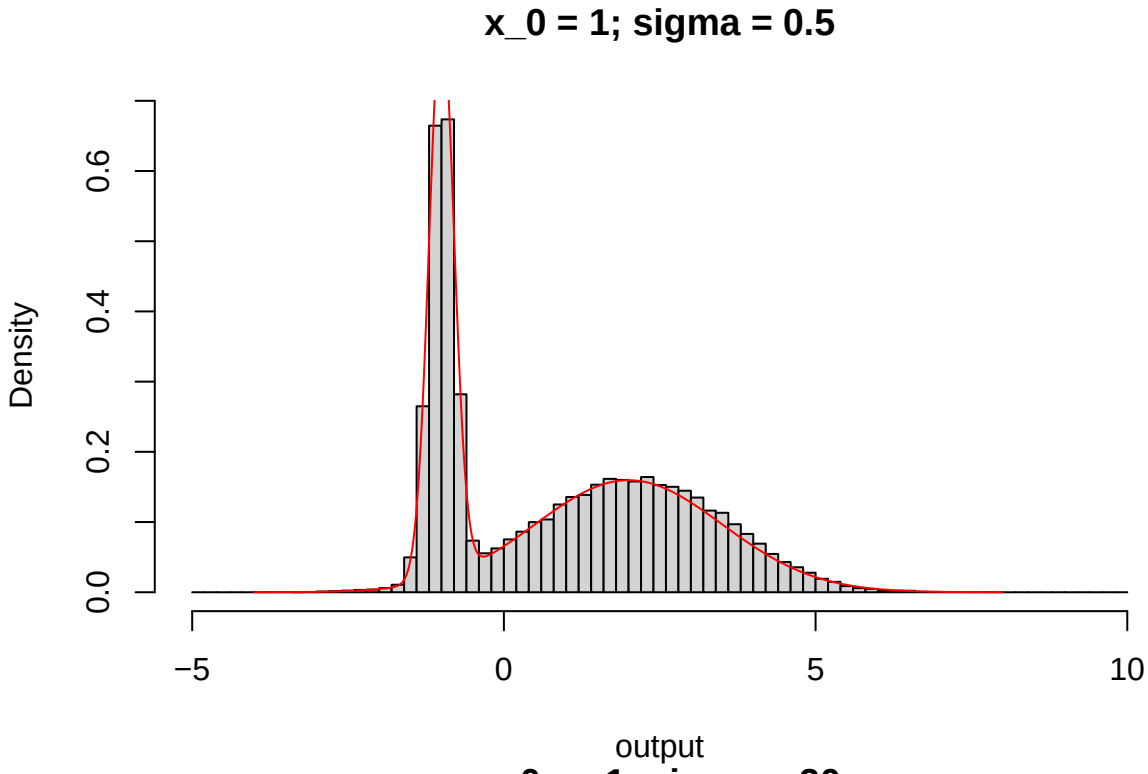
Sample the target distribution using the random walk Metropolis-Hastings sampling with different choices of `x_0` and `sigma` and view and interpret the output. Consider how the different choices of the starting point `x_0` and the ability for the proposal to explore the space with different values of `sigma` impact the results of the algorithm. You can choose your own different parameters or use these ones provided `x_0.vals <- c(-1, 1, 5)` and `sigma.vals <- c(0.01, 0.5, 20)`. To visualize your results, consider plotting the probability histogram of the outputs and drawing the true target density on top.

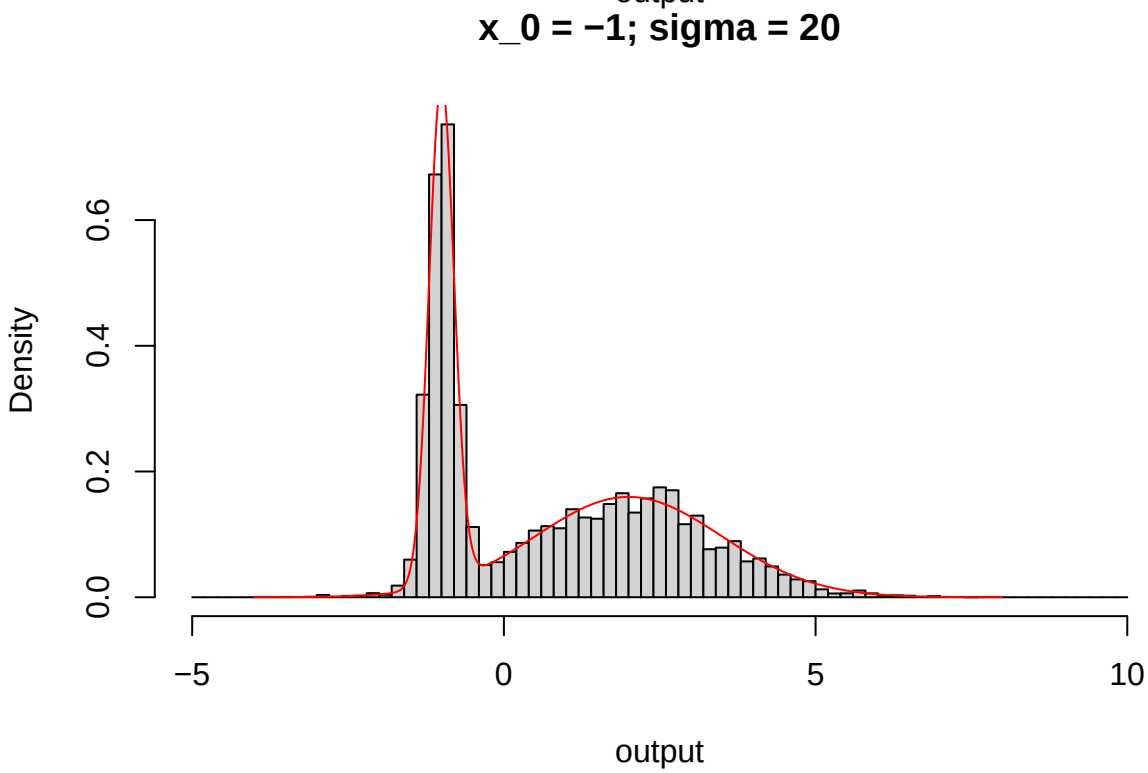
Hint: `expand.grid` might be a useful function to look at many combinations of choices of `x_0` and `sigma`

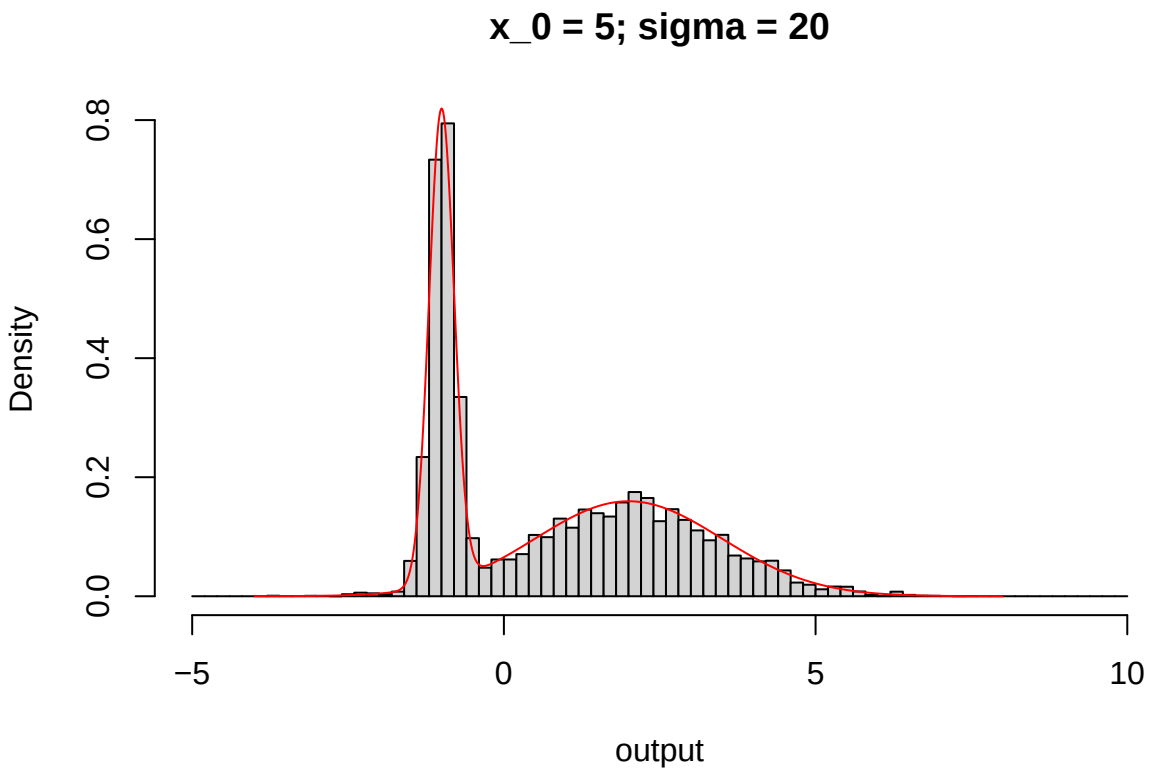
```
set.seed(5003)
sigma <- c(0.01, 0.5, 20)
x_0 <- c(-1, 1, 5)
params <- expand.grid(x_0, sigma)
x <- seq(from = -4, to = 8, length.out = 1000)
invisible(lapply(1:nrow(params), function(i) {
  x_0 <- params[i, 1]
  sigma <- params[i, 2]
  output <- mc.algorithm(x_0, n = 50000, sigma = sigma)
  hist(output, probability = TRUE,
       main = paste0("x_0 = ", x_0, "; sigma = ", sigma),
       breaks = seq(from = -5, to = 10, by = 0.2))
  lines(x, f(x), col = 'red')
}))
```


x_0 = −1; sigma = 0.01
Density
0.0
0.5
1.0
1.5
−5
0
5
10
output


x_0 = 5; sigma = 0.01
Density
0.0
0.2
0.4
0.6
0.8
−5
0
5
10
output


x_0 = 1; sigma = 0.5
Density
0.0
0.2
0.4
0.6
−5
0
5
10
output


x_0 = −1; sigma = 20
Density
0.0
0.2
0.4
0.6
−5
0
5
10
output



Looking at the outputs, we can see that sigma too small means the algorithm gets stuck and doesn't explore the space sufficiently. Conversely sigma too large, meaning some proposals will jump beyond the reasonable possibilities of the target distribution and be rejected and the Markov chain wont move enough.