02 Sampling algorithms

Shravan Vasishth

SMLP

MCMC sampling

The inversion method for sampling

This method works when we know the closed form of the pdf we want to simulate from and can derive the inverse of that function.
Steps:

1. Sample one number u from $Unif(0, 1)$. Let $u = F(z) = \int_L^z f(x)\,dx$ (here, L is the lower bound of the pdf f).
2. Then $z = F^{-1}(u)$ is a draw from $f(x)$.

Example 1: Samples from Standard Normal

Take a sample from the Uniform(0,1):

```
u<-runif(1,min=0,max=1)
```

Let f(x) be a Normal density—we want to sample from this density. The inverse of the CDF in R is qnorm. It takes as input a probability and returns a quantile.

```
qnorm(u)
```

```
## [1] -2.117653
```

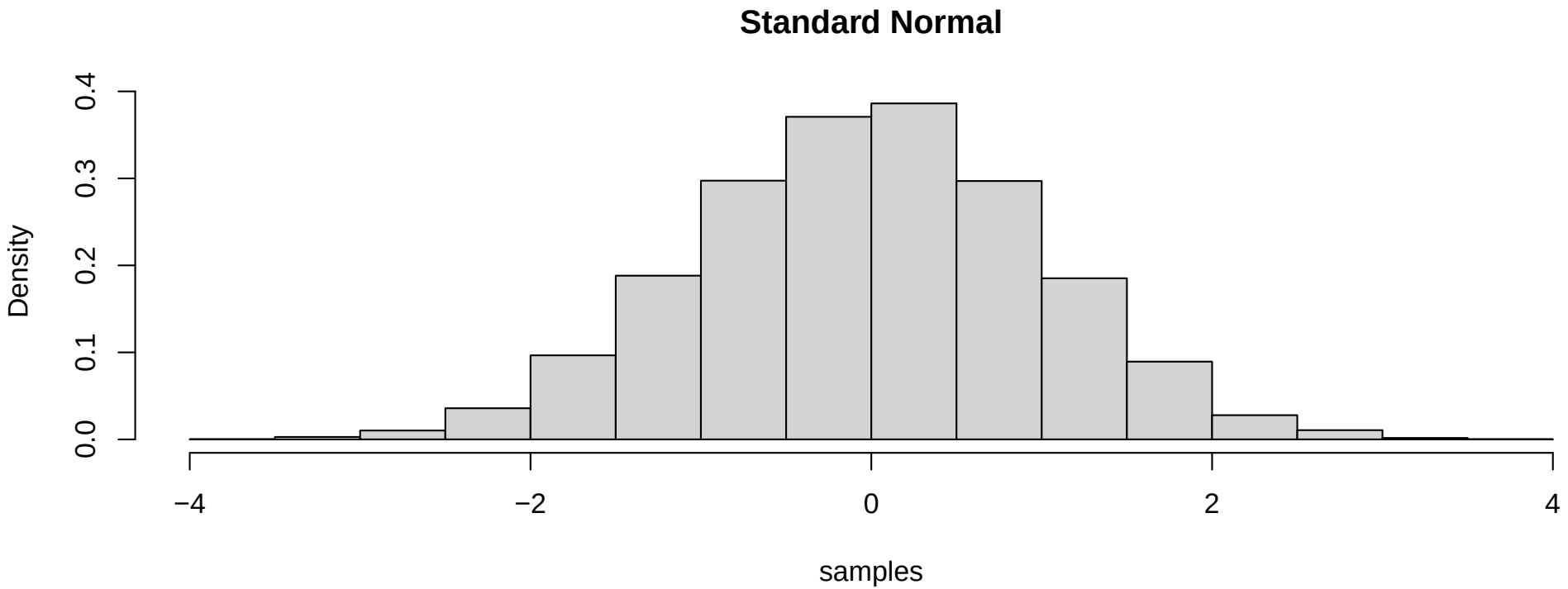
Example 1: Samples from Standard Normal

If we do this repeatedly, we will get samples from the Normal distribution (here, the standard normal).

```
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qnorm(u)
}
```

Example 1: Samples from Standard Normal

```
hist(samples,freq=FALSE,
     main="Standard Normal")
```



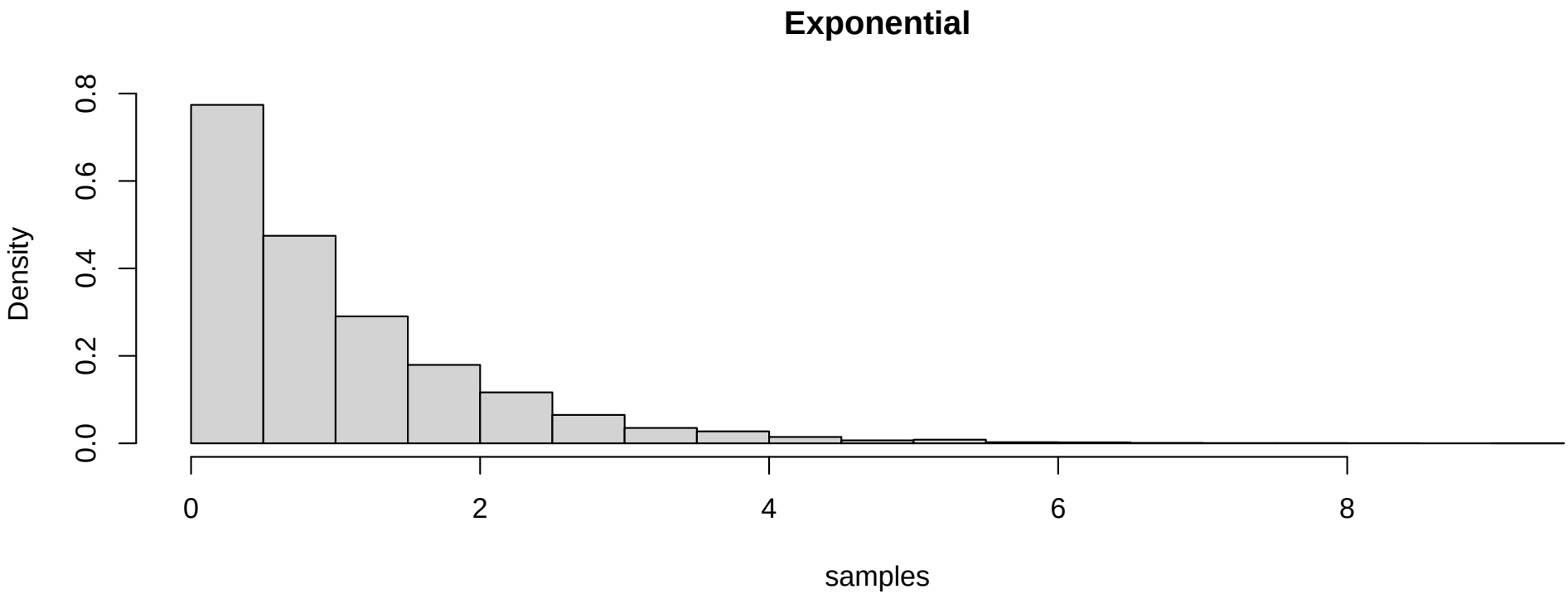
Example 2: Samples from Exponential or Gamma

Now try this with the exponential with rate 1:

```
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qexp(u)
}
```

Example 2: Samples from Exponential or Gamma

```
hist(samples,freq=FALSE,main="Exponential")
```



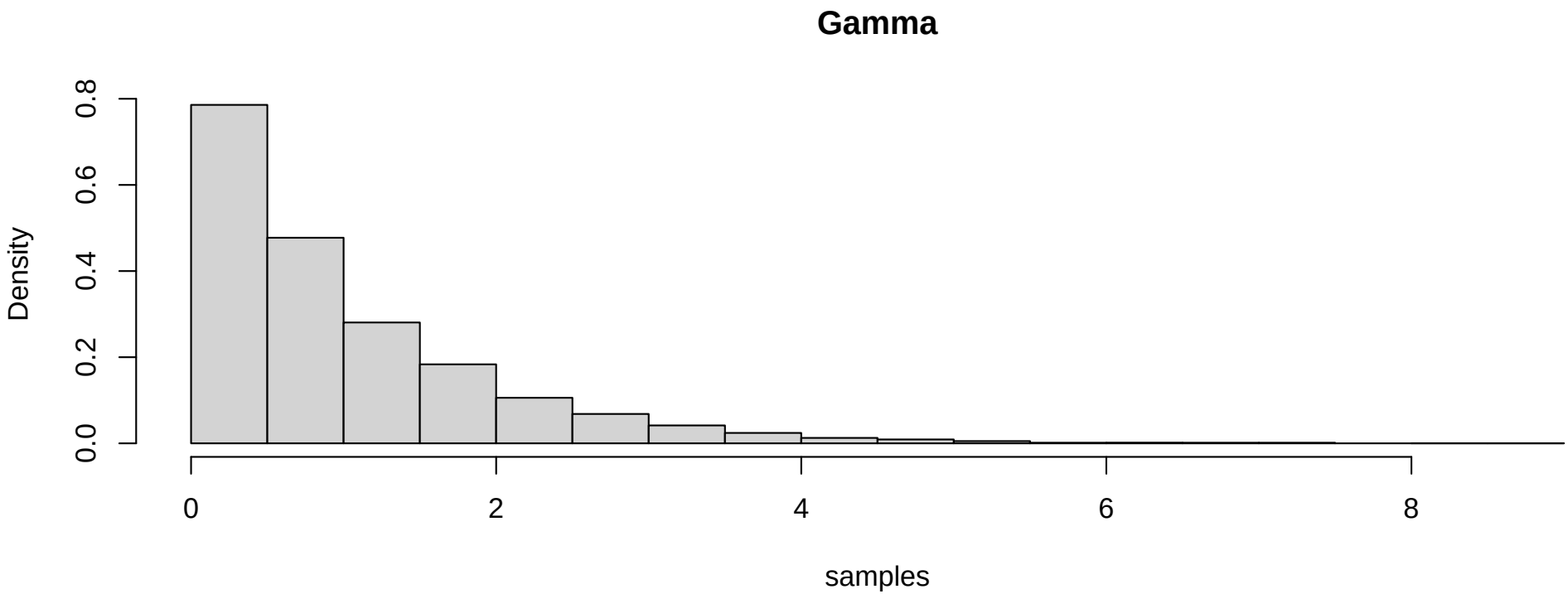
Example 2: Samples from Exponential or Gamma

Or the Gamma with rate and shape 1:

```
nsim<-10000
samples<-rep(NA,nsim)
for(i in 1:nsim){
  u <- runif(1,min=0,max=1)
  samples[i]<-qgamma(u,rate=1,shape=1)
}
```

Example 2: Samples from Exponential or Gamma

```
hist(samples,freq=FALSE,main="Gamma")
```



Example 3

Let $f(x) = \frac{1}{40}(2x + 3)$, with $0 < x < 5$. Now, we can't just use the family of q functions in R, because this density is not defined in R.

We have to draw a number from the uniform distribution and then solve for z, which amounts to finding the inverse function:

$$u = \int_0^z \frac{1}{40}(2x + 3) \tag{1}$$

```
u<-runif(1000,min=0,max=1)

z<-(1/2) * (-3 + sqrt(160*u +9))
```

This method can't be used if we can't find the inverse, and it can't be used with multivariate distributions.

Gibbs sampling

Gibbs sampling is a very commonly used method in Bayesian statistics. Here is how it works.

Let Θ be a vector of parameter values, let length of Θ be k. Let j index the j-th iteration.

1. Assign some starting values to Θ:

   $\Theta^{j=0} \leftarrow S$

2. Set $j \leftarrow j + 1$
3. **1.** Sample $\theta_1^j \mid \theta_2^{j-1} \ldots \theta_k^{j-1}$.

   2. Sample $\theta_2^j \mid \theta_1^j \theta_3^{j-1} \ldots \theta_k^{j-1}$.

$\vdots$

   k. Sample $\theta_k^j \mid \theta_1^j \ldots \theta_{k-1}^j$.

4. Return to step 1.

Example: A simple bivariate distribution

Assume that our bivariate (joint) density is:

$$f(x, y) = \frac{1}{28}(2x + 3y + 2) \tag{2}$$

Using the methods discussed in the Foundations chapter, it is possible to analytically work out the conditional distributions from the joint distribution:

$$f(x \mid y) = \frac{f(x, y)}{f(y)} = \frac{(2x + 3y + 2)}{6y + 8} \tag{3}$$

$$f(y \mid x) = \frac{f(x, y)}{f(x)} = \frac{(2x + 3y + 2)}{4y + 10} \tag{4}$$

Example: A simple bivariate distribution

The Gibbs sampler algorithm is:

1. Set starting values for the two parameters $x = -5, y = -5$. Set j=0.
2. Sample x^{j+1} from $f(x \mid y)$ using inversion sampling. You need to work out the inverse of $f(x \mid y)$ and $f(y \mid x)$ first. To do this, for $f(x \mid u)$, we have find z_1:

$$u = \int_0^{z_1} \frac{(2x + 3y + 2)}{6y + 8} \, dx \tag{5}$$

And for $f(y \mid x)$, we have to find z_2:

$$u = \int_0^{z_2} \frac{(2x + 3y + 2)}{4y + 10} \, dy \tag{6}$$

Example: A simple bivariate distribution

```
x<-rep(NA,2000)
y<-rep(NA,2000)
x[1]<- -5 ## initial values
y[1]<- -5
for(i in 2:2000)
{ #sample from x | y
  u<-runif(1,min=0, max=1)
  x[i]<-sqrt(u*(6*y[i-1]+8)+(1.5*y[i-1]+1)*(1.5*y[i-1]+1))-
    (1.5*y[i-1]+1)
  #sample from y | x
u<-runif(1,min=0,max=1)
y[i]<-sqrt((2*u*(4*x[i]+10))/3 +((2*x[i]+2)/3)*((2*x[i]+2)/3))
    ((2*x[i]+2)/3)
}
```

Example: A simple bivariate distribution

You can run this code (hidden) to visualize the simulated posterior distribution. See Figure 1.

Simulated bivariate density using Gibbs sampling

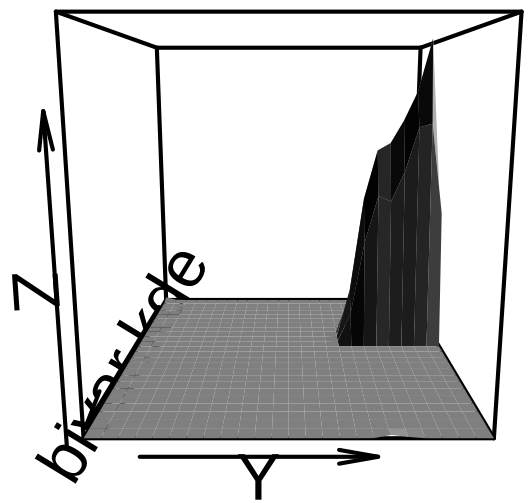


Figure 1: Example of posterior distribution of a bivariate distribution.

Example: A simple bivariate distribution

A central insight here is that knowledge of the conditional distributions is enough to simulate from the joint distribution, provided such a joint distribution exists.

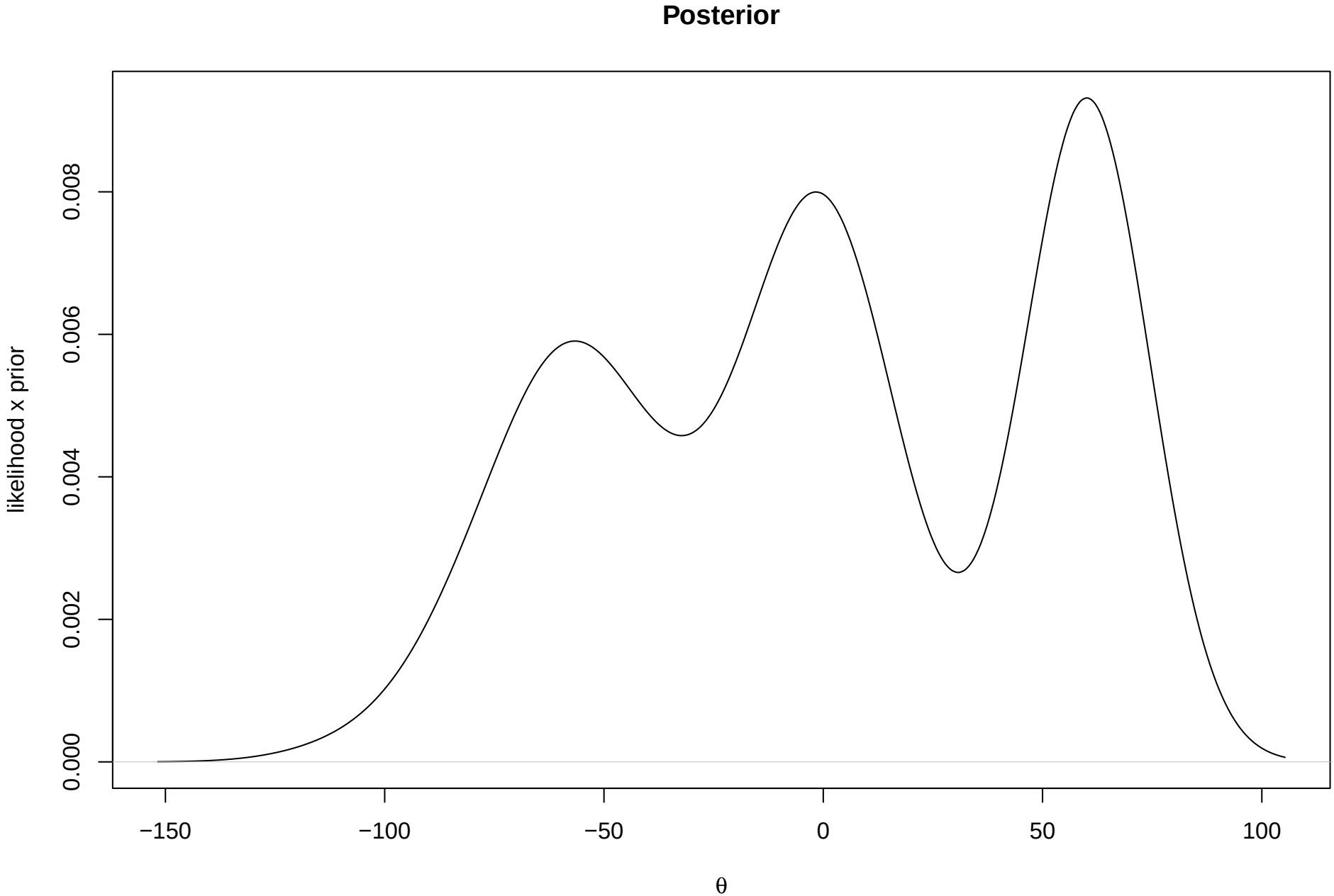
Random walk Metropolis

- Start at random location $\theta_0 \in \Theta$
- For step $i = 1, \ldots, I$
  - Propose new location using a "symmetric jumping distribution"
  - Calculate

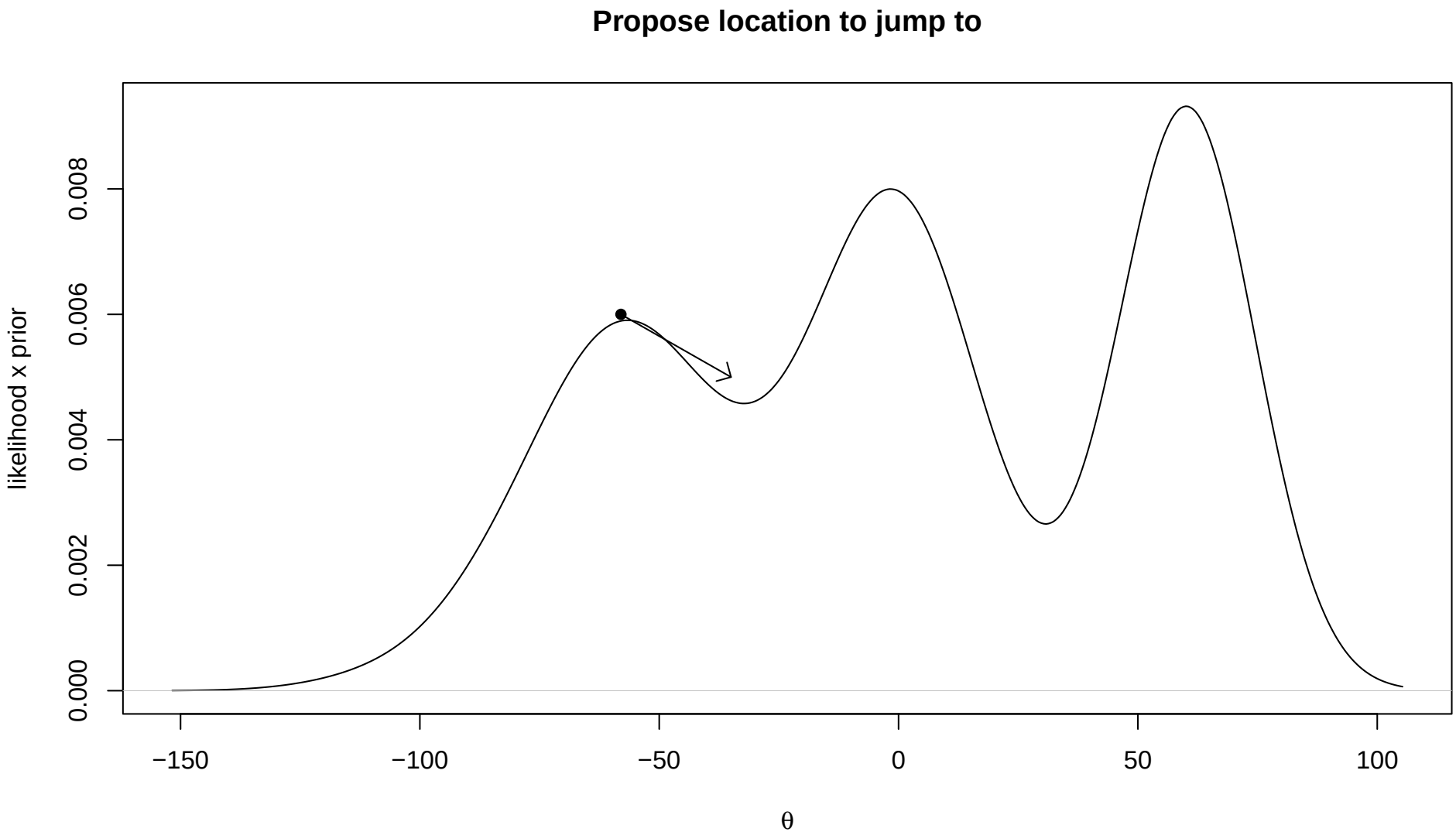
$$\text{ratio} = \frac{lik(\theta_{i+1}) \times prior(\theta i+1)}{lik(\theta_i) \times prior(\theta i)}$$

  - Generate $u \sim Uniform(0, 1)$
  - r>u, move from θ_i to θ_{i+1}, else stay at θ_i

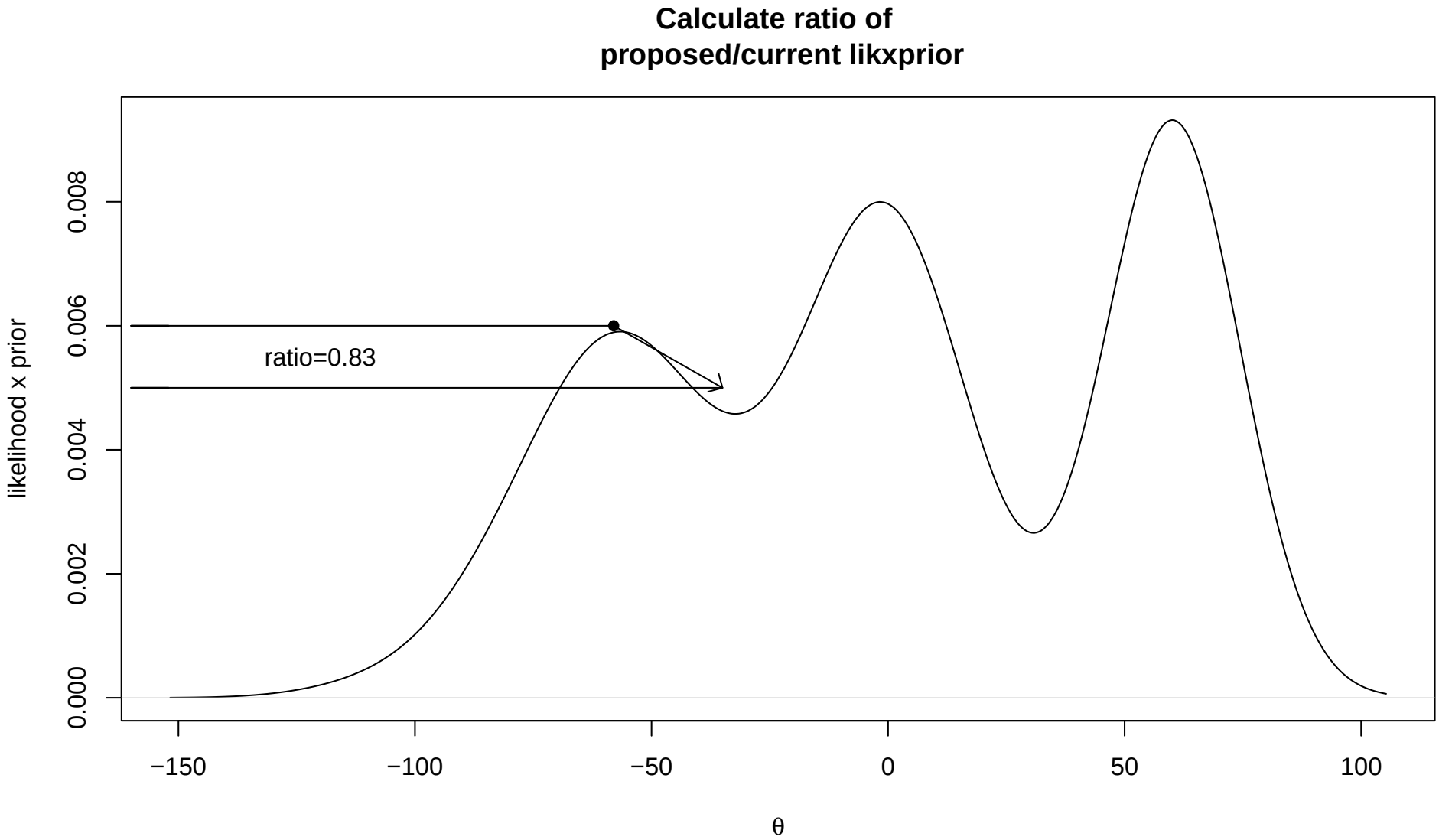
Random Walk Metropolis



Random Walk Metropolis

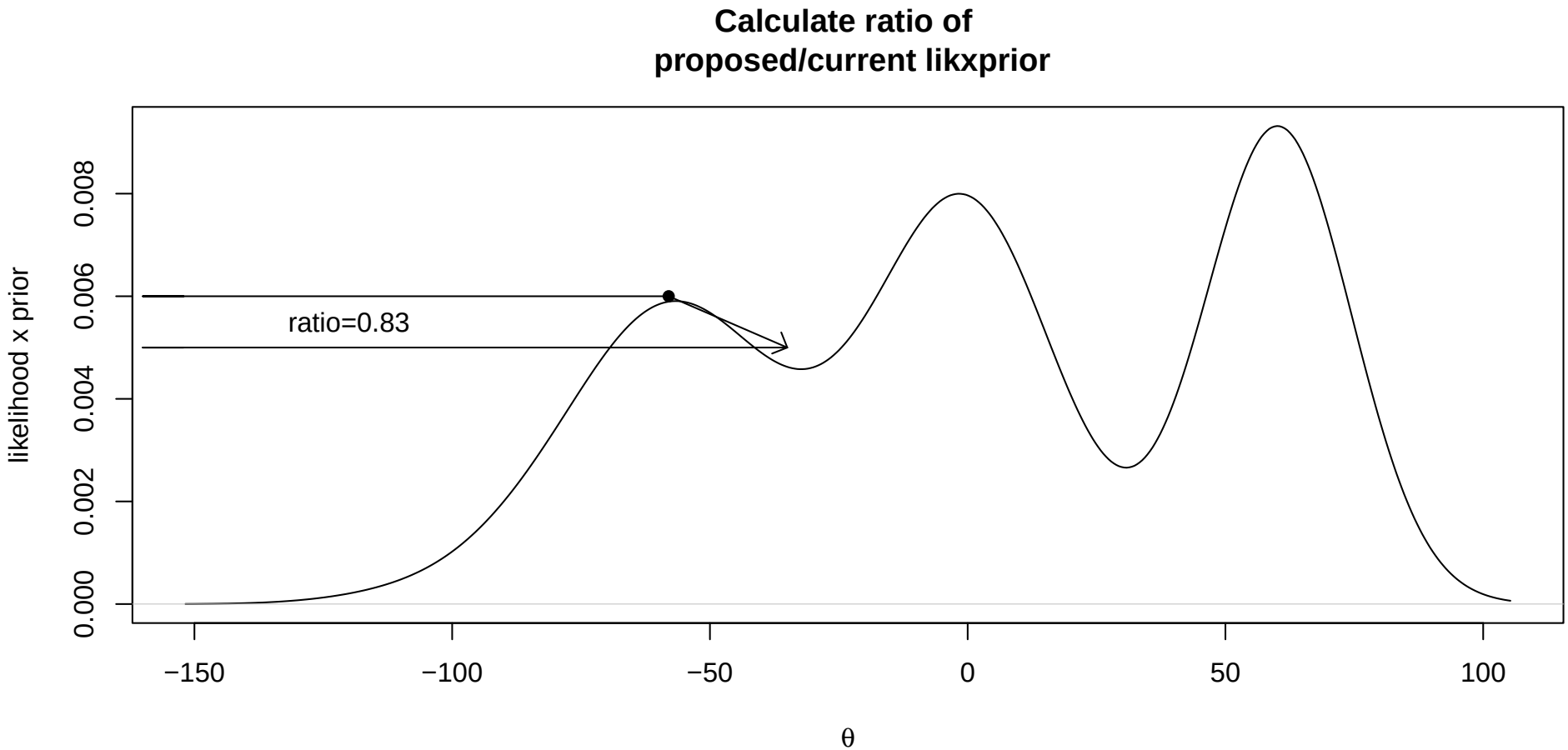


Random Walk Metropolis

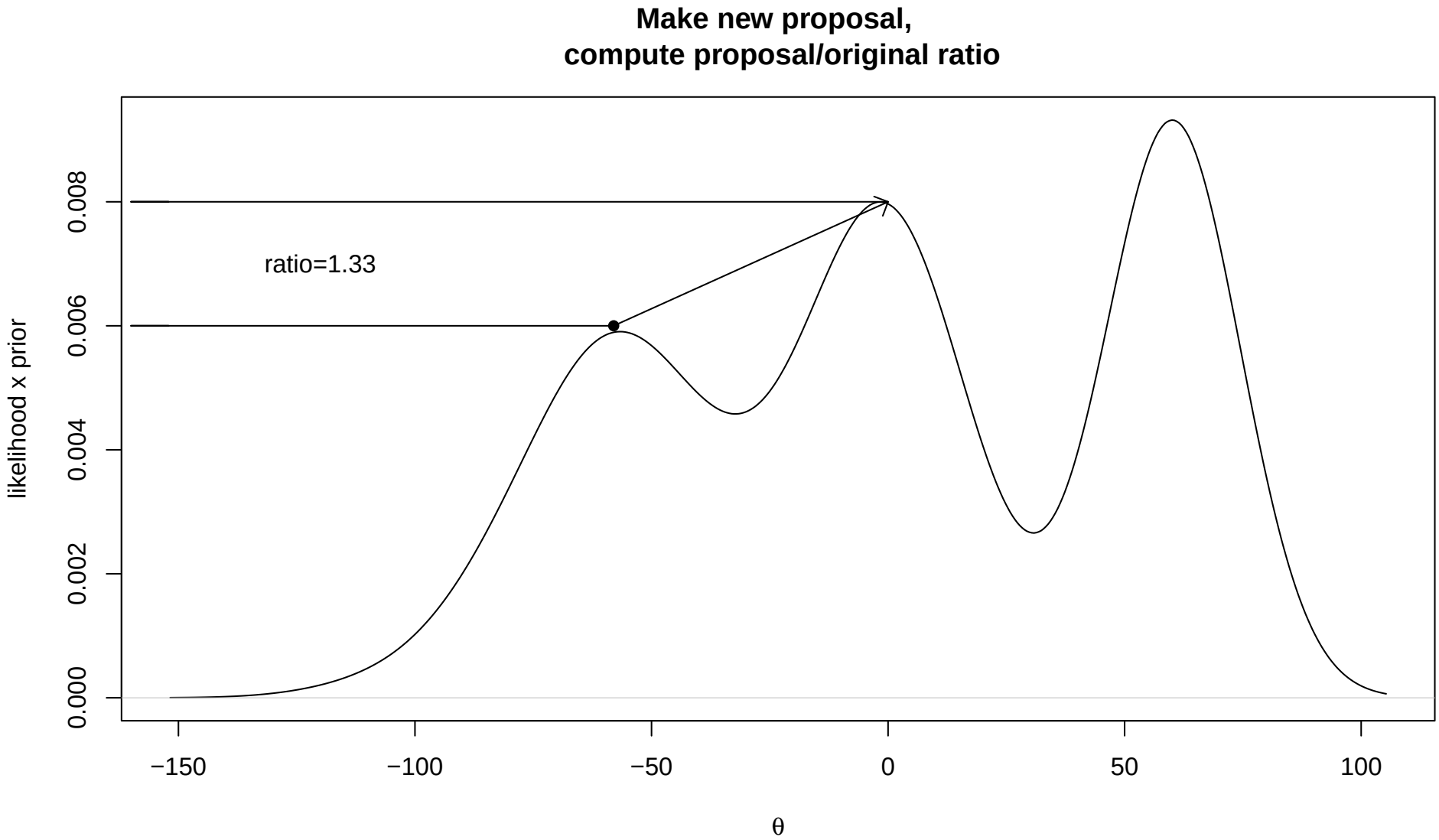


Random Walk Metropolis

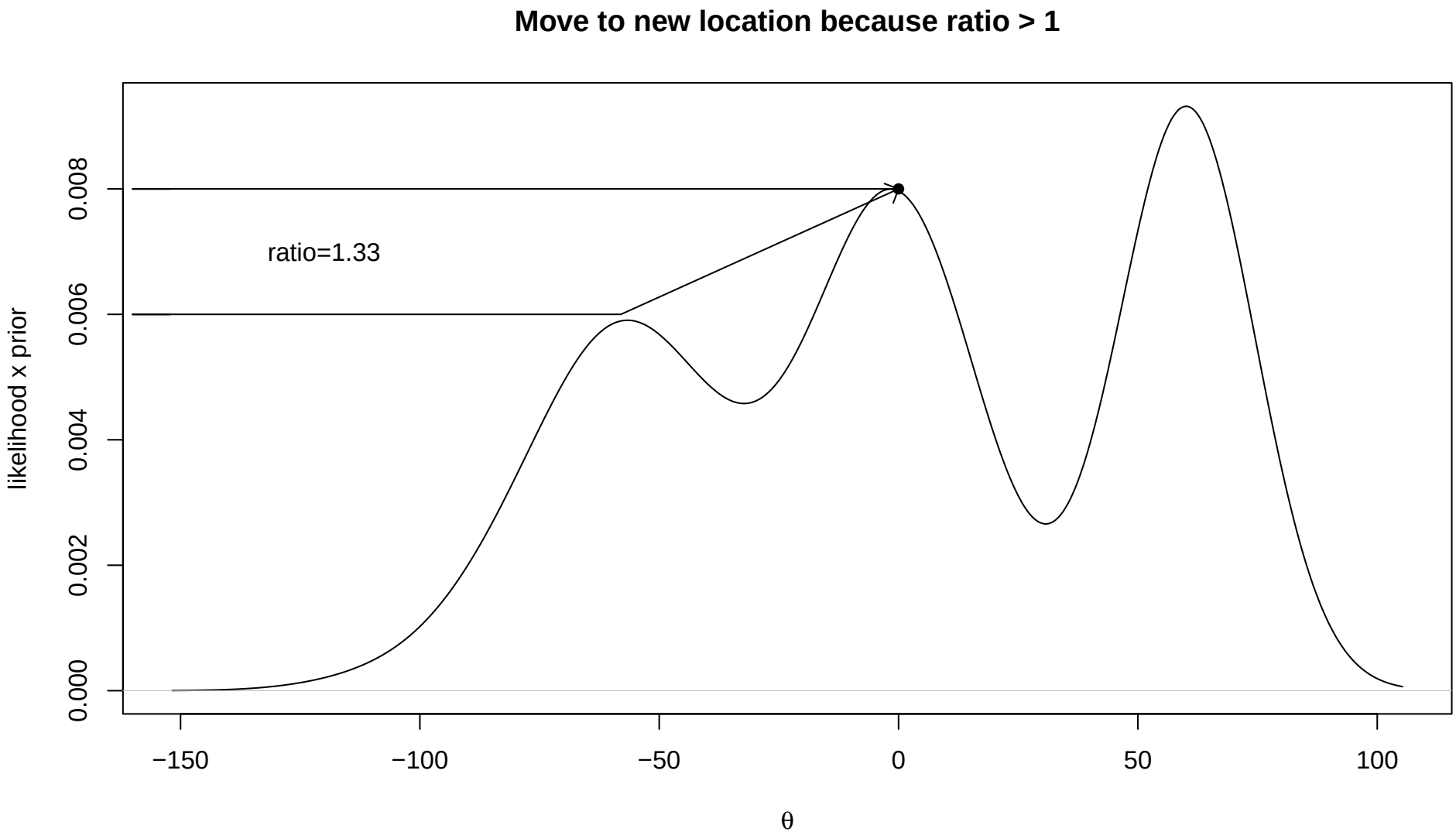
Take a sample $u \sim Uniform(0, 1)$. Suppose u $= 0.90$. Since $ratio < u$, remain at current position (reject proposal).



Random Walk Metropolis



Random Walk Metropolis



Hamiltonian Monte Carlo

- Instead of Gibbs sampling or Metropolis etc., Stan uses this more efficient sampling approach.
- HMC works well for the high-dimensional models we will fit (hierarchical models).
- Gibbs sampling faces difficulties with some of the complex hierarchical models we will be fitting later.
- HMC will always succeed for these complex models.

Hamiltonian Monte Carlo

- One limitation of HMC (which Gibbs sampling does not have) is that HMC only works with continuous parameters (not discrete parameters).
- For our purposes, it is enough to know what sampling using MCMC is, and that HMC gives us posterior samples efficiently.
- A good reference explaining HMC is Neal 2011. However, this paper is technically very demanding.
- More intuitively accessible introductions are available via Michael Betancourt's home page: https://betanalpha.github.io/. In particular, this video is helpful: https://youtu.be/jUSZboSq1zg.

Background: Hamiltonian dynamics

Imagine an ice puck moving over a frictionless surface of varying heights.

- The puck moves at constant velocity (momentum) k on flat surface
- When the puck moves up an incline, it's kinetic energy goes down, and its potential energy goes up
- When the puck slows down and comes to a halt, kinetic energy becomes 0.
- When the puck slides back, kinetic energy goes up, potential energy goes down.

See animation.

Background: Hamiltonian dynamics

The ice puck has

- location θ
- momentum k

We can describe the dynamics of puck movement in terms of this **total energy** equation

$$\underbrace{Energy(\theta, k)}_{} = \underbrace{U(\theta)}_{\text{Potential energy}} + \underbrace{KE(k)}_{\text{Kinetic energy}}$$

In classical mechanics, this total energy is called a Hamiltonian, so we can write:

$$H(\theta, k) = U(\theta) + KE(k)$$

Background: Hamiltonian dynamics

Potential energy

Define the potential energy of the puck as
$U(\theta) = -\log(p(X|\theta)p(\theta))$
Thus:

- $U(\theta)$ is defined to be the negative log posterior density
- It is defined to be the inverse of the posterior space

Background: Hamiltonian dynamics

Kinetic energy

Kinetic energy is $\frac{1}{2}mv^2$
m=mass, v=velocity
Assuming q dimensions, and m=1
$KE(k) = \sum_{i=1}^{q} \frac{k_i^2}{2}$

Background: Hamiltonian dynamics

The evolution of a puck: The equations of motion

Let there be $i = 1, \ldots, d$ parameters.

Given the equation:

$H(\theta, k) = U(\theta) + KE(k)$

Classical mechanics defines these equations of motion:

- position: $\frac{d\theta_i}{dt} = \frac{\delta H}{\delta k_i}$
- momentum: $\frac{dk_i}{dt} = -\frac{\delta H}{\delta \theta_i}$

These equations define the mapping from state of the puck at time t to time $t + s$.

Simplified algorithm

- Choose initial **momentum** $k \sim N(0, \Sigma)$.
- Record puck's current **position** (value of θ)
- Record puck's **momentum**, the current value of k
- The puck's **position** and **momentum** lead to an accept/reject rule that yields samples from the posterior with a high probability of acceptance.
- The approximate solution to the equations of motion is done using a modification of Euler's method.

HMC demonstration

The HMC algorithm takes as input the log density and the gradient of the log density. In Stan, these will be computed internally; the user doesn't need to do any computations.

For example, suppose the log density is $f(\theta) = -\frac{\theta^2}{2}$. Its gradient is $f'(\theta) = -\theta$. Setting this gradient to 0, and solving for θ, we know that the maximum is at 0. We know it's a maximum because the second derivative, $f''(\theta) = -1$, is negative. See Figure 2.

This is the machinery we learnt in the foundations chapter (recall how we found MLEs in particular).

HMC demonstration

```
theta<-seq(-4,4,by=0.001)
plot(theta,-theta^2/2,type="l",main="Log density")
```

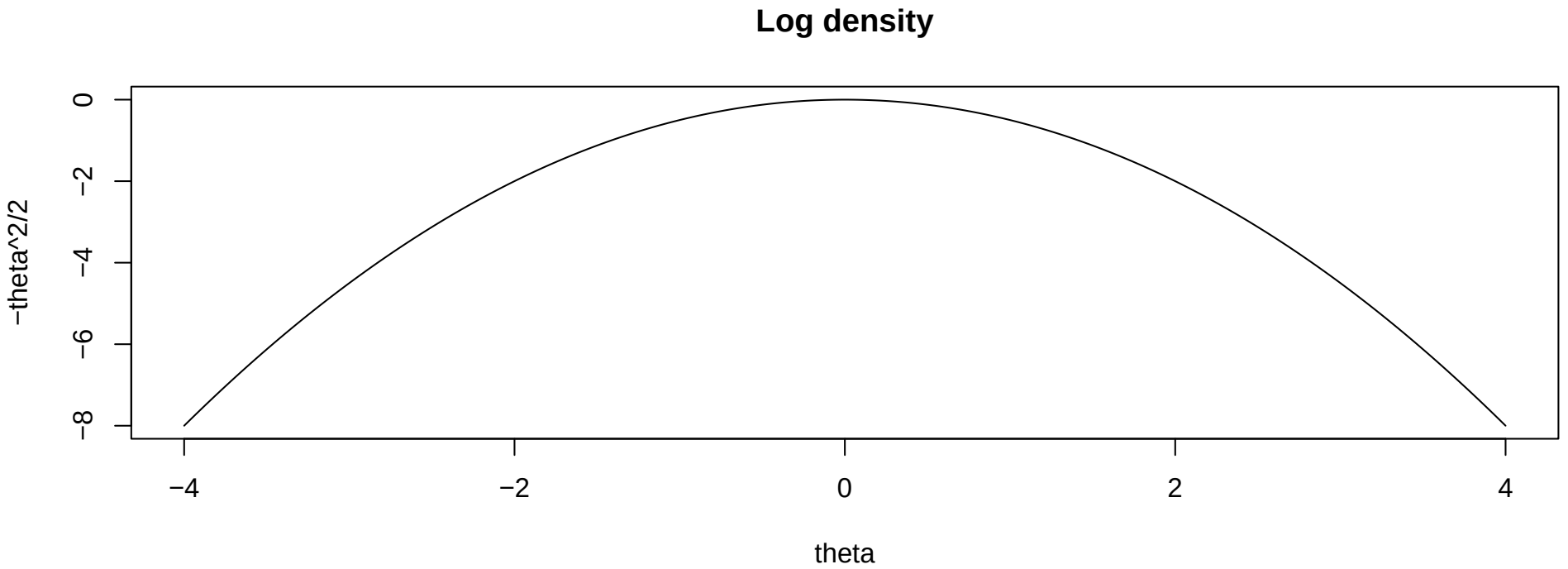


Figure 2: Example log density.

HMC demonstration

The Radford Neal algorithm for HMC.

Source: Jarad Niemi's github repository.

HMC demonstration

```
## Radford Neal algorithm:
HMC_neal <- function(U, grad_U, e, L, current_theta) {
theta = current_theta
omega = rnorm(length(theta),0,1)
current_omega = omega
omega = omega - e * grad_U(theta) / 2
for (i in 1:L) {
theta = theta + e * omega
if (i!=L) omega = omega - e * grad_U(theta)
}
omega = omega - e * grad_U(theta) / 2
omega = -omega
current_U = U(current_theta)
current_K = sum(current_omega^2)/2
proposed_U = U(theta)
proposed_K = sum(omega^2)/2
if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K))
{
return(theta)
}
else {
return(current_theta)
}}
HMC <- function(n_reps, log_density, grad_log_density, tuning, initial) {
theta = rep(0, n_reps)
theta[1] = initial$theta
for (i in 2:n_reps) theta[i] = HMC_neal(U = function(x) -log_density(x),
grad_U = function(x) -grad_log_density(x),
e = tuning$e,
L = tuning$L,
theta[i-1])
theta
}
```

HMC demonstration

Then, we use the HMC function above to take 2000 samples from the posterior.

We drop the first few (typically, the first half) samples, which are called warm-ups. The reason we drop them is that the initial samples may not yet be sampling from the posterior.

HMC demonstration

```
theta <- HMC(n_reps=2000,
             log_density=function(x) -x^2/2,
             grad_log_density=function(x) -x,
             tuning=list(e=1,L=1),
             initial=list(theta=0))
```

HMC demonstration

Figure 3 shows a **trace plot**, the trajectory of the samples over 2000 iterations.

This is called a **chain**. When the sampler is correctly sampling from the posterior, we see a characteristic "fat hairy caterpillar' ' shape, and we say that the sampler has **converged**. You will see later what a failed convergence looks like.

HMC demonstration

```
plot(theta,type="l",main="Trace plot of posterior samples")
```

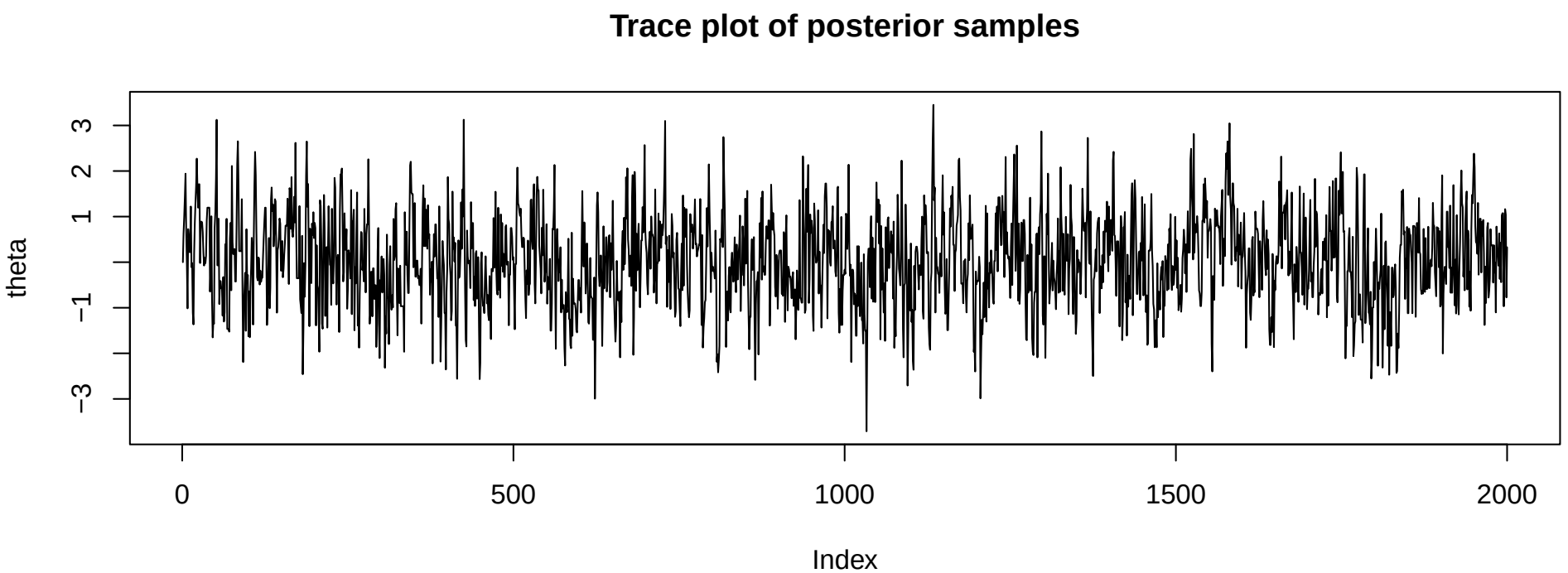


Figure 3: An example of a trace plot.

HMC demonstration

When we fit Bayesian models, we will always run four parallel chains.

This is to make sure that even if we start with four different initial values chosen randomly, the chains all end up sampling from the same distribution.

When this happens, we see that the chains overlap visually, and we say that the chains are **mixing**.

HMC demonstration

Figure 4 shows the posterior distribution of θ.

We are not discarding samples here because the sampler converges quickly in this simple example.

HMC demonstration

```
hist(theta, freq=F, 100,

     main="Posterior distribution of the parameter.",
     xlab=expression(theta))
curve(dnorm, add=TRUE, col='red', lwd=2)
```

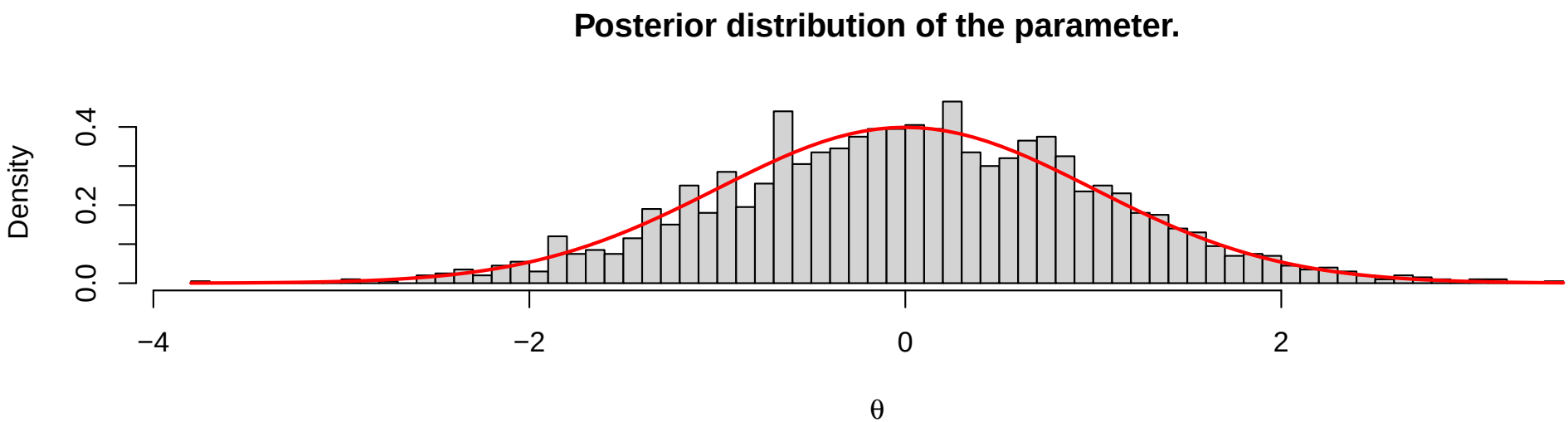


Figure 4: Sampling from the posterior using HMC. The red curve shows the distribution Normal(0,1).

HMC demonstration

In the modeling we do in the next part of the course, the Stan software will do the sampling for us.