Metropolis Algorithm

STAT 251, Unit 10

Overview

Review

Metropolis Algorithm

More about Tuning Parameter(s), (e.g., σ^2_{prop})

Comparison of Metropolis-estimated Density to Target Density

Additional Considerations for Metropolis Algorithm: Log scale and Supports

Why is it okay to only identify $g(x)$?

Your Turn to Practice

Review

Posterior Inference

- We assume that data (not just the observed sample data but the population of all data) comes from a parametric family of distributions (e.g., Poisson, or Normal, or Binomial, etc.) The joint distribution of the data is the **likelihood**.
- We have prior beliefs about the parameters in the likelihood.
- We use observed data to update the prior beliefs and form posterior beliefs about the unknown parameters.

With conjugate priors, it is easy to determine the posterior distribution.

What if the priors are not conjugate?

Non-conjugate Priors, part 1

One possibility is that in a multi-parameter distribution, the complete conditionals are easy to identify and to sample from. Then, Monte Carlo inference is possible through Gibbs sampling.

Example: Normal-Normal Inverse Gamma Setting of Unit 8. Gibbs sampling is used to sample from posterior distribution of (μ, σ^2) by repeatedly iterating through the following process:

- Sample a value of μ from its complete conditional distribution (with σ^2 equal to the current value of σ^2)
- Sample a value of σ^2 from its complete conditional distribution (with μ equal to the current value of μ)

Key: Even though full posterior was trickier to determine because in this setting the prior we used is nonconjugate, the complete conditionals may be easy to work with.

Non-conjugate Priors, part 2

If only one unknown parameter, complete conditional is the same as posterior, implying that if posterior is unknown, no advantage whatsoever in working with the complete conditionals instead. Gibbs can't help us here!

More generally, with multiple parameters it may not be easy to determine the joint posterior distribution nor the complete conditional disributions of the individual parameters. Without the complete conditionals, Gibbs alone doesn't help much.

Metropolis Algorithm

Simple Example of Nonstandard Distribution

Suppose the target distribution is

$$f(x) = c \exp[-x^6](1 + |2x|)^3$$

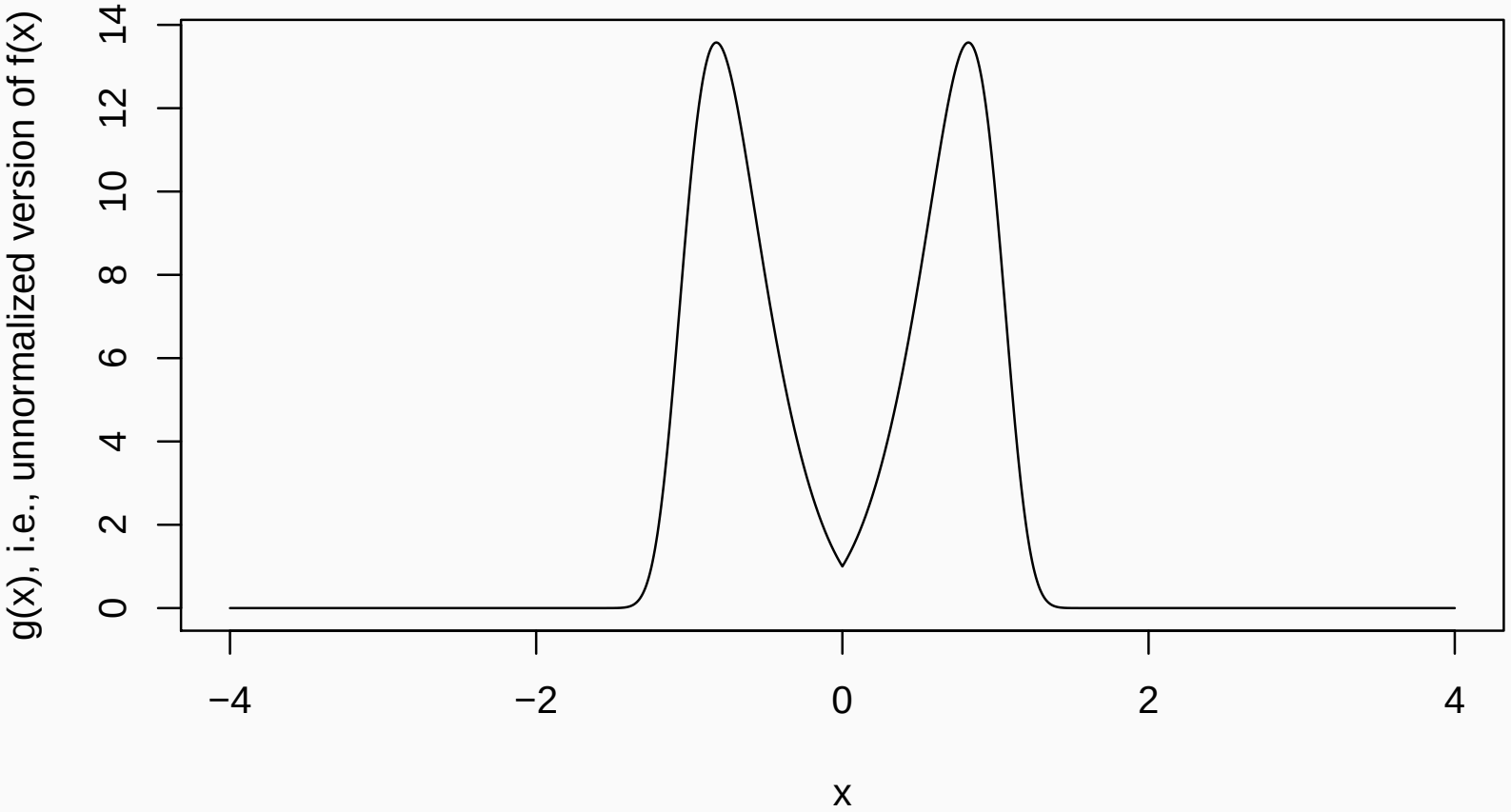
The c is an unknown *normalizing constant*, but for the algorithm to work we don't need to know c. The unnormalized density (i.e., a function which is only *proportional* to a density) is

$$g(x) = \exp[-x^6](1 + |2x|)^3,$$

which is depicted on the next slide.

Unnormalized Density, $g(x)$

```
plot(x <- seq(-4, 4, length.out=1001),
     exp(-x^6)*(1+abs(2*x))^3,
     type="l", xlab="x",
     ylab="g(x), i.e., unnormalized version of f(x)")
```



Goal: Obtain a sample of simulated draws from an arbitrary statistical distribution

We'll call the target function $f(\cdot)$. $f(\cdot)$ is a pdf or pmf, depending on whether the random variable is continuous or discrete.

It can be helpful to sample from $f(\cdot)$, such as to allow Monte Carlo- estimation of the distribution's mean.

- If the distribution is for a discrete random variable, sampling is often easy enough.
- If the distribution is for a continuous random variable, there are several possibilities. We will focus on the use of the Metropolis algorithm.

Metropolis Algorithm–A bedrock of Bayesian analysis

The Metropolis algorithm is an iterative algorithm for sampling from a target distribution.

- Like Gibbs sampling, it produces a (correlated) set of draws from the target distribution after a sufficiently long burn-in period.
- After initializing the sequence, new (proposal) values are sampled from a known proposal distribution that depends on the current value. The next value of the sequence is either the proposed value or a repeat of the current value.

Note: Metropolis is the surname of a physicist who came up with the algorithm (1953 publication). Hastings generalized the algorithm (1970 publication). The Metropolis-Hastings algorithm was popularized among statisticians for Bayesian inferences by Alan Gelfand and Adrian Smith (1990 publication).

Preliminaries for the Metropolis Algorithm

Let $f(x)$ be the distribution from which we want to sample. We will refer to this as the *target distribution*.

Let $g(x)$ be a function that is non-negative valued and that is proportional to $f(x)$. That is, $f(x) = cg(x)$ for some constant $c > 0$. **Importantly, we need not know the numerical value of c!**

We want to utilize a proposal distribution $q(x_p|x_c)$ that has the properties listed on the next slide.

Notation: x_c denotes the *current* value of x in the algorithm. x_p denotes the *proposed* value of x in the algorithm.

Desired properties of the proposal distribution, $q(x_p|x_c)$.

- $q(\cdot|\cdot)$ is easy to sample from
- $q(x_p|x_c) = q(x_c|x_p)$ for all x_c and for all x_p (Such a q function is said to be *symmetric* because the value of the *pdf* is unchanged by swapping the order of its inputs.)
- The support of $q(\cdot|\cdot)$ is at least as large as the support of $f(\cdot)$.

If sampling a continuous random variable, $q(x_p|x_c)$ is often chosen to be the normal distribution with mean x_c and fixed variance σ^2_{prop}. That is, $X_p|X_c \sim N(X_c, \sigma^2_{prop})$.

- Do not confuse the fixed variance for the proposal distribution, σ^2_{prop}, with any variance(s) that appear in the prior, likelihood, or posterior. σ^2_{prop} is strictly a *tuning parameter*. More on this later.

Metropolis Algorithm

S0 Choose an initial (current) value of x, denoted by $x_{c,0}$; $x_{c,0}$ must be in the support of f(x).

S1 For $j = 1, 2, \ldots, J$,

S1a Sample a value x_p from the proposal distribution $q(x_p, |x_{c,j-1})$.

S1b Compute the probability of accepting the proposed value, α, which equals

$$\alpha = \min\left(1, \frac{g(x_p)}{g(x_{c,j-1})}\right)$$

S1c With probability α, accept the proposed value (i.e., set $x_{c,j} = x_p$). Otherwise, do not accept the proposed value and reuse the current value (i.e., set $x_{c,j} = x_{c,j-1}$).

S2 Determine the burn-in period and assess how well the chain mixes.

Eventually the series of $\{x_{c,j}\}$ values in the chain behave as though they are a (autocorrelated) random sample from the target distribution, $f(x)$.

Continued Example of sampling from nonstandard distribution

Recall that in our example, the target distribution was

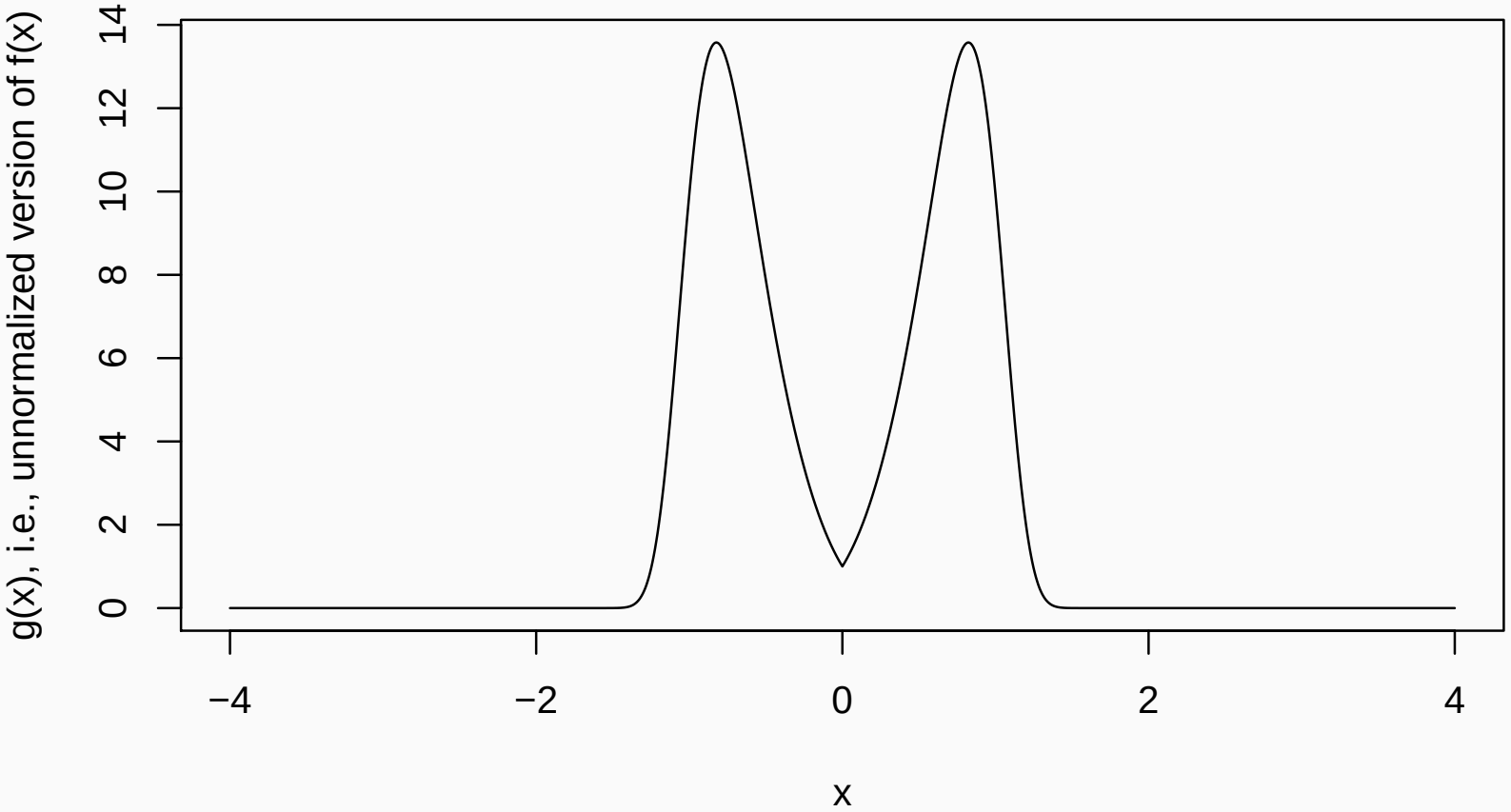
$$f(x) = c \exp[-x^6](1 + |2x|)^3$$

for some (unknown) c, and that the target density is proportional to

$$g(x) = \exp[-x^6](1 + |2x|)^3.$$

Unnormalized Target Density, $g(x)$

```
plot(x <- seq(-4, 4, length.out=1001),
     exp(-x^6)*(1+abs(2*x))^3,
     type="l", xlab="x",
     ylab="g(x), i.e., unnormalized version of f(x)")
```



Pseudocode for the Metropolis Algorithm Example, part 1

- Initialize the *chain* of X values by choosing an $x_{c,0}$ from the support of $f(x)$.
  - Let's use $x_{c,0} = 1$.
- Choose an easy-to-sample, symmetric proposal distribution $q(x_p|x_c)$ whose support contains the support of the target distribution, $f(x)$.
  - Note in this example, the support of $f(x)$ is $(-\infty, \infty)$. If $x_p|x_c \sim N(x_c, 0.5^2)$ then $q(x_p|x_c)$ is symmetric**, easy to sample from, and has the same support as $f(x)$.

**Easy to see the symmetry of this density function by writing out the pdf and plugging in x_p and x_c: Note that

$$q(x_p|x_c) = \frac{1}{\sqrt{2\pi(0.5^2)}} \exp\{-\frac{1}{2(0.5^2)}(x_p - x_c)^2\}$$
$$= \frac{1}{\sqrt{2\pi(0.5^2)}} \exp\{-\frac{1}{2(0.5^2)}(x_c - x_p)^2\} = q(x_c|x_p)$$

Pseudocode for the Metropolis Algorithm Example, part 2

- For j in 1 to 10,000:
  - sample a value x_p from $q(x_p|x_{c,j-1})$, that is, $x_p|x_{c,j-1} \sim N(x_{c,j-1}, 0.5^2)$.
  - Compute the **acceptance ratio**

$$\alpha(x_p, x_{c,j-1}) = \min\left(1, \frac{g(x_p)}{g(x_{c,j-1})}\right)$$

  - Generate one random variable, U, from the *uniform*$(0, 1)$ distribution. If $U \leq \alpha(x_p, x_{c,j-1})$ then assign $x_{c,j} = x_p$; otherwise set $x_{c,j} = x_{c,j-1}$.
- Check for convergence by examining a trace plot of the draws.

Metropolis Algorithm Implementation

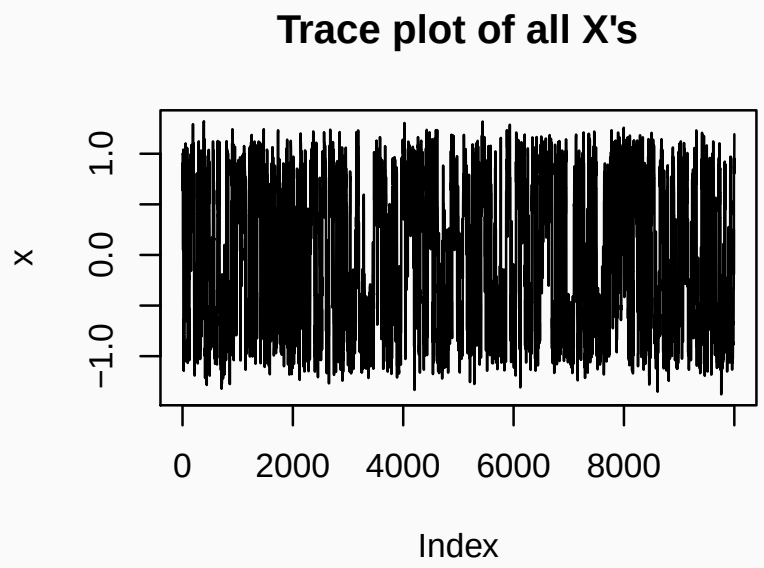
```r
### g(x) is an UNNORMALIZED version of f(x)
g <- function(x){ exp(-x^6) * (1 + abs(2 * x))^3 }
## Initialize the sequence:
x <- rep(NA, 10001)
x[1] <- 1
set.seed(363)
for (j in 2:length(x)){
  xcurr <- x[j-1]
  xprop <- rnorm(1,xcurr,0.5)
  alpha <- min(1, g(xprop)/g(xcurr))
  x[j] <- ifelse(runif(1) <= alpha, xprop, xcurr)
}
```

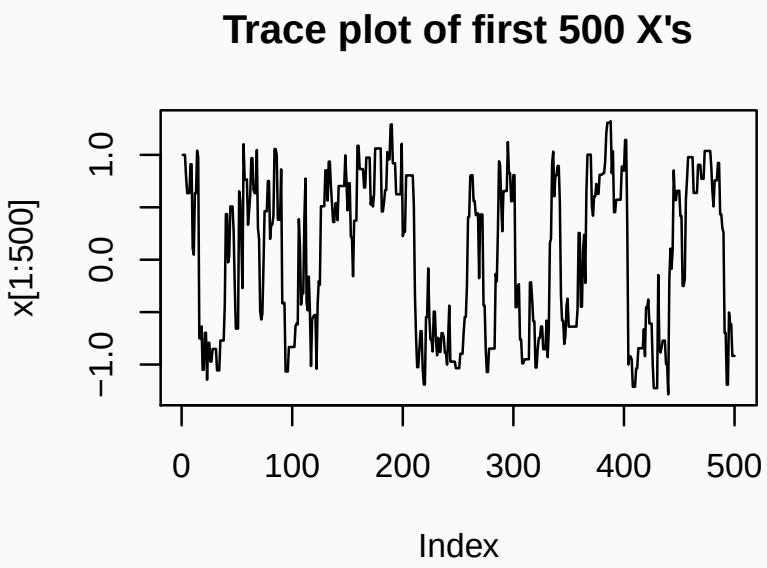
Assessing convergence and mixing

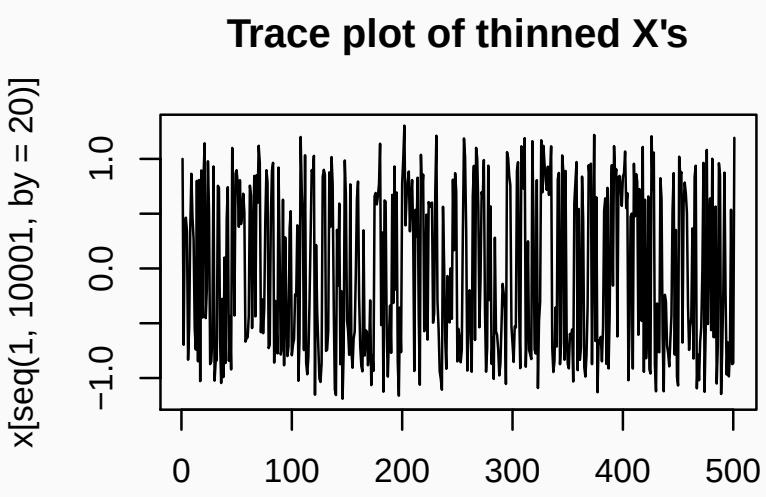
As with Gibbs sampling, we use trace plots to assess convergence and mixing, and we also rely on autocorrelation function (acf) plots to assess mixing.

Trace plots with many iterations are too compressed to get much out of, so two alternatives are to consider blocks of, say, 1000 consecutive draws or to consider a thinned sequence of the draws.

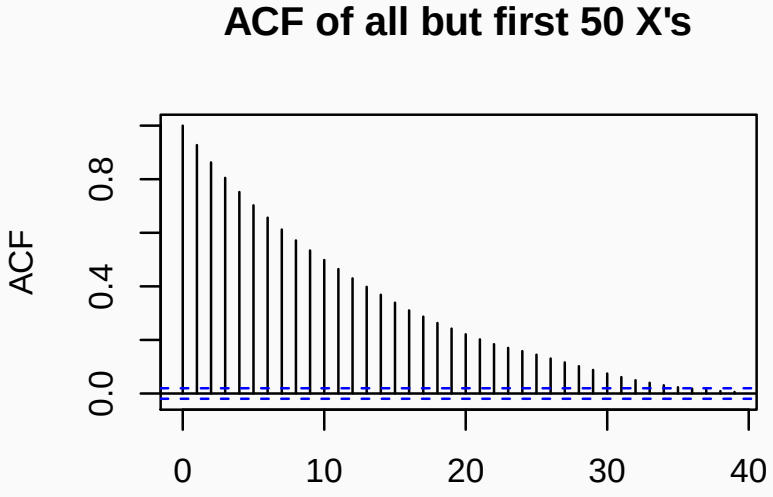
> *Thinning* refers to saving every kth draw, for some integer $k > 1$. Thinned sequences should exhibit less autocorrelation.







Thinned by 20 (i.e., every 20th value in sequence)



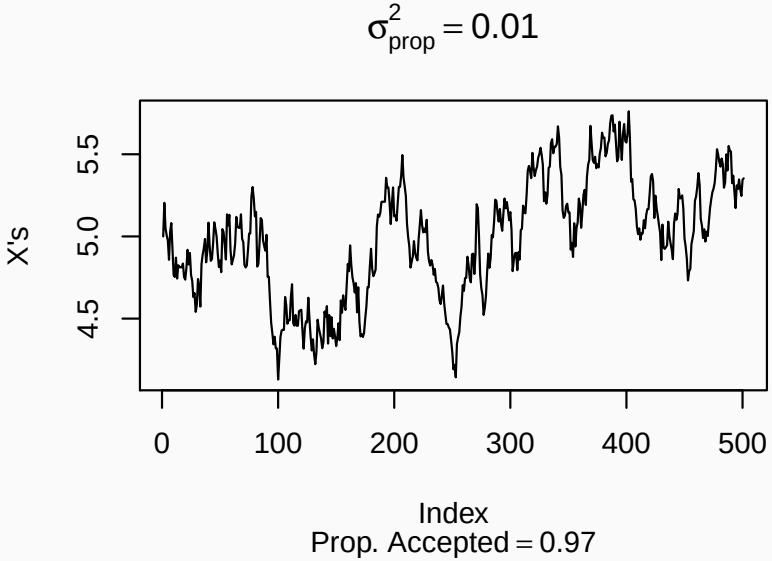
First 50 X's were omitted as burn−in

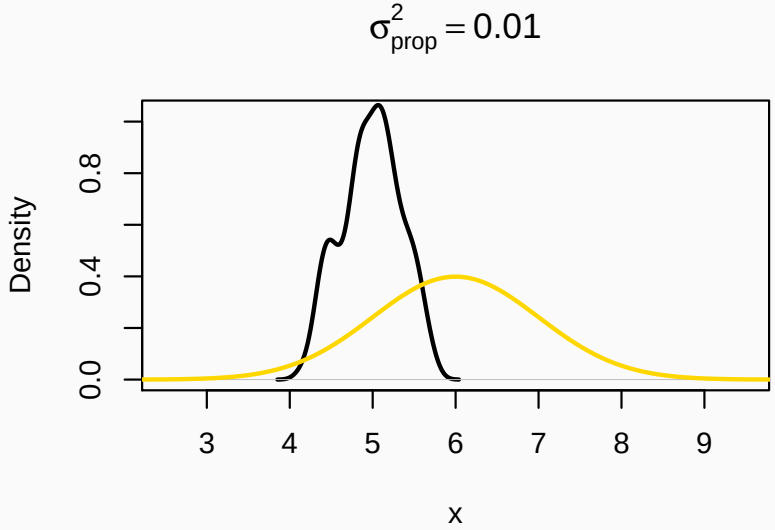
More about Tuning Parameter(s), (e.g., σ^2_{prop})

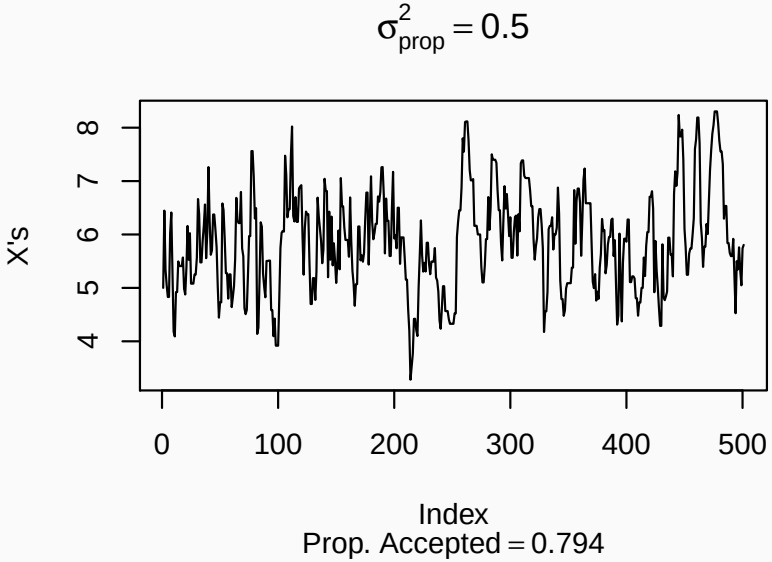
Convergence of the Metropolis Algorithm

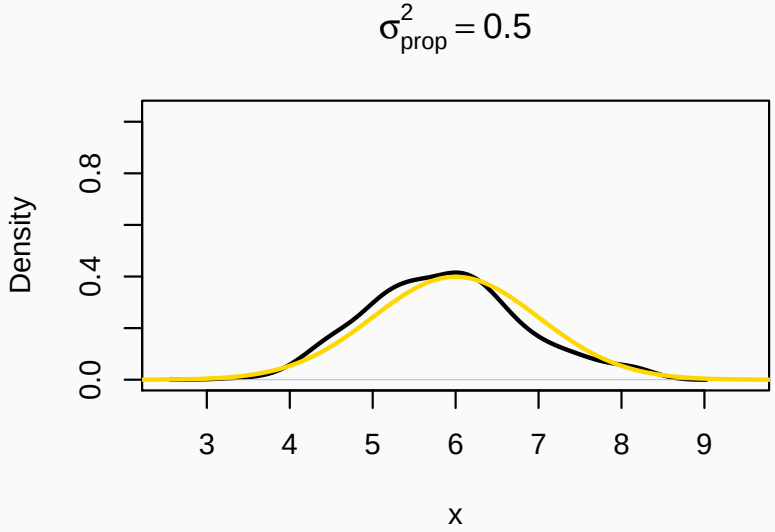
- The convergence depends on many factors, one of which is the size of the proposed changes.
- You don't want to accept too often, nor too infrequently.
- If the size of the proposed changes is too small, you will accept often, but the draws don't explore the distribution's support well enough.
- If the size of the proposed changes is too large, you will not accept often enough and the sequence will not mix well.

When using various values of the tuning parameter σ^2_{prop} with $N(6, 1^2)$ Target Distribution and starting at $X_0 = 5$

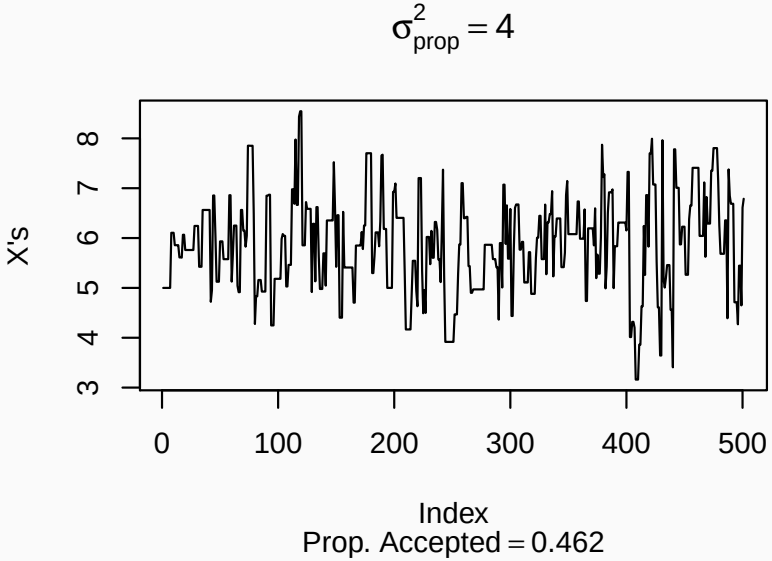


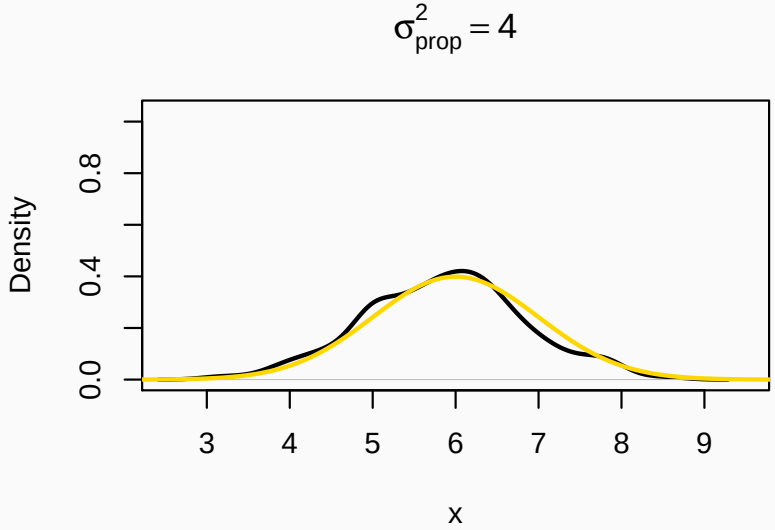


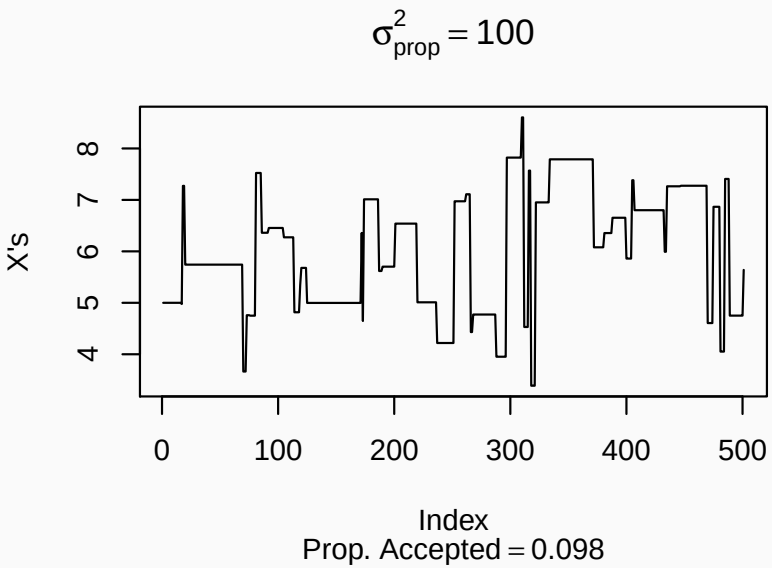


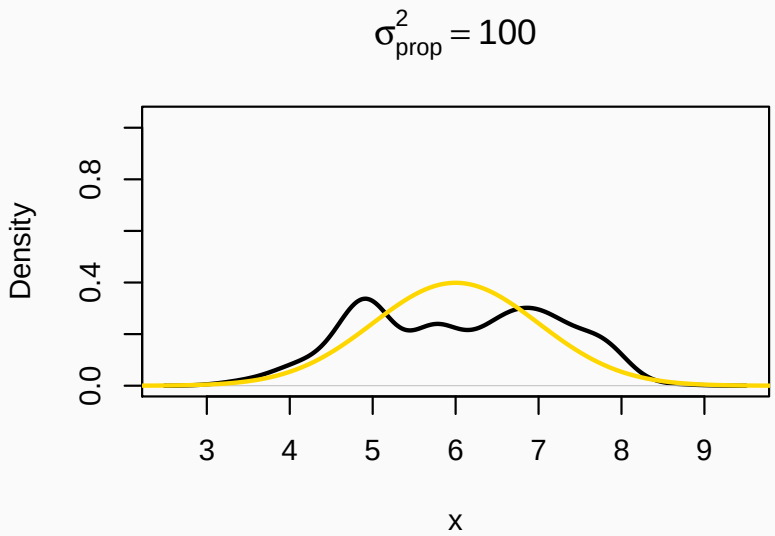


When using various values of the tuning parameter σ^2_{prop} with $N(6, 1^2)$ Target Distribution and starting at $X_0 = 5$









How to choose the tuning parameter?

In the provided example, the variance of the proposal distribution was $0.5^2 = 0.25$. This proposal distribution variance is referred to as a tuning parameter because

- it has no inherent connection with any model quantities of interest, but
- nonetheless, its value must be chosen in such a way that the algorithm can mix well. This often requires trial and error.

What is a good acceptance rate to target with the MH-algorithm?

> With a normal proposal distribution and an approximately normal target density, then about 44% of proposed draws should be accepted (Gelman, Carlin, Stern, & Rubin, 2004, p. 306). However, anywhere from 35% - 55% is reasonable.

How to check what proportion of proposed draws were accepted?

One method is to simply compare how often consecutive values in the chain differ. E.g., use the diff() function and see how often it is nonzero

```
mean( diff(x) != 0) ## Proportion of accepted draws
```

```
## [1] 0.5611
```

Further detail is on the following slide

Pedagogical Demonstration (only first 5 x's so we can see all)

```
x[1:5]
```

```
## [1] 1.0000000 1.0000000 1.0000000 0.7936021 0.6347361
```

```
diff(x[1:5]) # e.g., x[2]-x[1], x[3]-x[2], etc.
```

```
## [1]  0.0000000  0.0000000 -0.2063979 -0.1588661
```

```
diff(x[1:5]) != 0 ## jth element TRUE IFF x[j+1] != x[j]
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

```
mean(diff(x[1:5]) != 0)
```

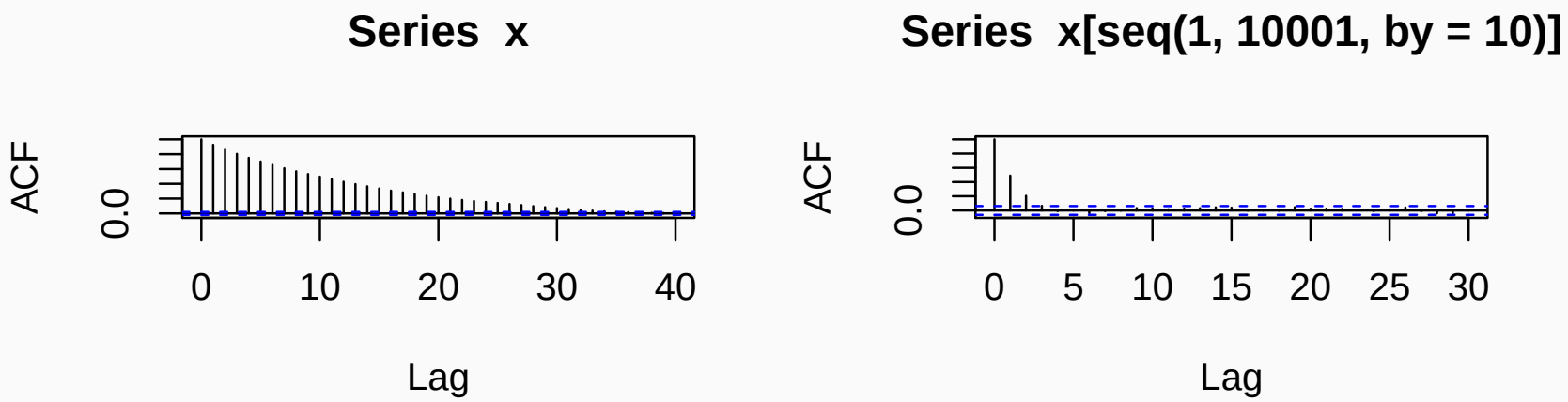
```
## [1] 0.5
```

Metropolis Algorithm Mixing

With poor mixing, need to run the algorithm for **many** more iterations. Might therefore be impractical to save all posterior draws. If so, thin the sequence (i.e., save only every k^{th} draw, for integer $k > 1$).

- Remember, thinned sequences expected to have reduced autocorrelation.

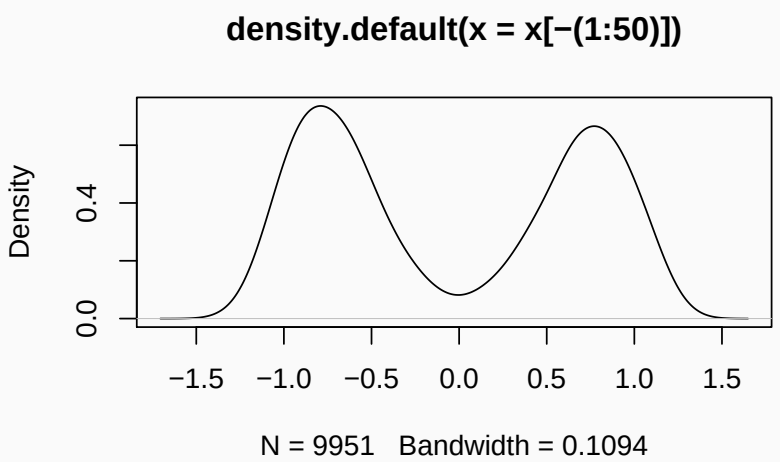
```
par(mfrow=c(1,2))
acf(x)
acf(x[seq(1, 10001, by=10)]) ## every 10th draw
```

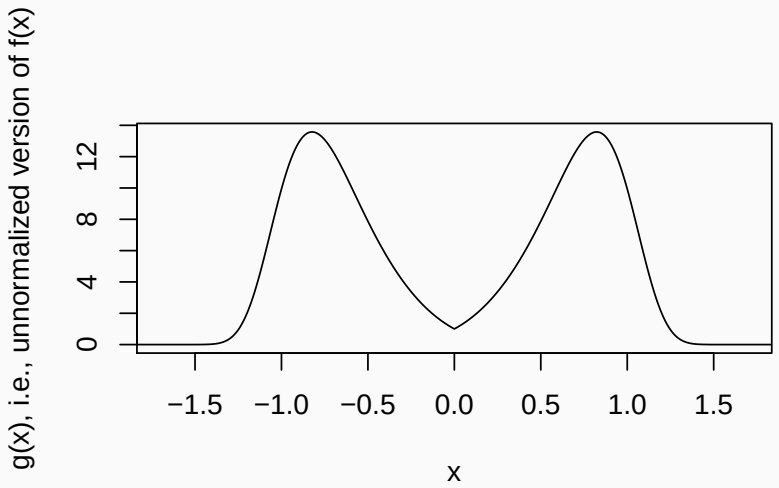


Comparison of Metropolis-estimated Density to Target Density

Metropolis-estimated Density vs. $g(x)$ (?!?)

```
par(mfrow=c(1,2))
g <- function(x){ exp(-x^6) * (1 + abs(2 * x))^3 }
plot(density(x[-(1:50)]))  ## Remember to exclude burn-in
plot(tt <- seq(-4, 4, length.out=1001), g(tt),
     type="l", xlab="x", xlim=c(-1.7, 1.7),
     ylab="g(x), i.e., unnormalized version of f(x)")
```





Notice the y limits. **Target density is $f(x)$, not $g(x)$!**

Comparison of Metropolis-estimated Density to Target Density, part 1

Recall that although $f(x) = c \exp[-x^6](1 + |2x|)^3$, we didn't know the normalizing constant c. We only know $g(x) = \exp[-x^6](1 + |2x|)^3$.

Comparison of Metropolis-estimated Density to Target Density, part 1

Recall that although $f(x) = c \exp[-x^6](1 + |2x|)^3$, we didn't know the normalizing constant c. We only know $g(x) = \exp[-x^6](1 + |2x|)^3$.

Problem: We can't compare the Metropolis-estimated density to the true target density (i.e., to $f(x)$) without knowing c.

Comparison of Metropolis-estimated Density to Target Density, part 1

Recall that although $f(x) = c \exp[-x^6](1 + |2x|)^3$, we didn't know the normalizing constant c. We only know $g(x) = \exp[-x^6](1 + |2x|)^3$.

Problem: We can't compare the Metropolis-estimated density to the true target density (i.e., to $f(x)$) without knowing c.

Solution: First recognize that c is simply the reciprocal of what $g(x)$ integrates to.

That is, $c = \frac{1}{\int_{-\infty}^{\infty} g(x)dx}$ is the (unique) value of the normalizing constant that is required for $f(x) = cg(x)$ to be a proper density function.

Comparison of Metropolis-estimated Density to Target Density, part 2

Easy to verify the normalizing constant to convert $g(x)$ to $f(x)$ is $c = \frac{1}{\int_{-\infty}^{\infty} g(x)dx}$:

$$\begin{aligned}
\int_{-\infty}^{\infty} f(x)dx &= \int_{-\infty}^{\infty} cg(x)dx \\
&= c\int_{-\infty}^{\infty} g(x)dx \\
&= \underbrace{\left[\frac{1}{\int_{-\infty}^{\infty} g(x)dx}\right]}_{=c}\int_{-\infty}^{\infty} g(x)dx \\
&= 1
\end{aligned}$$

Comparison of Metropolis-estimated Density to Target Density, part 2

Easy to verify the normalizing constant to convert $g(x)$ to $f(x)$ is $c = \frac{1}{\int_{-\infty}^{\infty} g(x)dx}$:

$$\begin{aligned}
\int_{-\infty}^{\infty} f(x)dx &= \int_{-\infty}^{\infty} cg(x)dx \\
&= c\int_{-\infty}^{\infty} g(x)dx \\
&= \underbrace{\left[\frac{1}{\int_{-\infty}^{\infty} g(x)dx}\right]}_{=c}\int_{-\infty}^{\infty} g(x)dx \\
&= 1
\end{aligned}$$

Problem: How to evaluate the integral of $g(x)$ if it is complicated?

Comparison of Metropolis-estimated Density to Target Density, part 2

Easy to verify the normalizing constant to convert $g(x)$ to $f(x)$ is $c = \frac{1}{\int_{-\infty}^{\infty} g(x)dx}$:

$$\begin{aligned}
\int_{-\infty}^{\infty} f(x)dx &= \int_{-\infty}^{\infty} cg(x)dx \\
&= c\int_{-\infty}^{\infty} g(x)dx \\
&= \underbrace{\left[\frac{1}{\int_{-\infty}^{\infty} g(x)dx}\right]}_{=c}\int_{-\infty}^{\infty} g(x)dx \\
&= 1
\end{aligned}$$

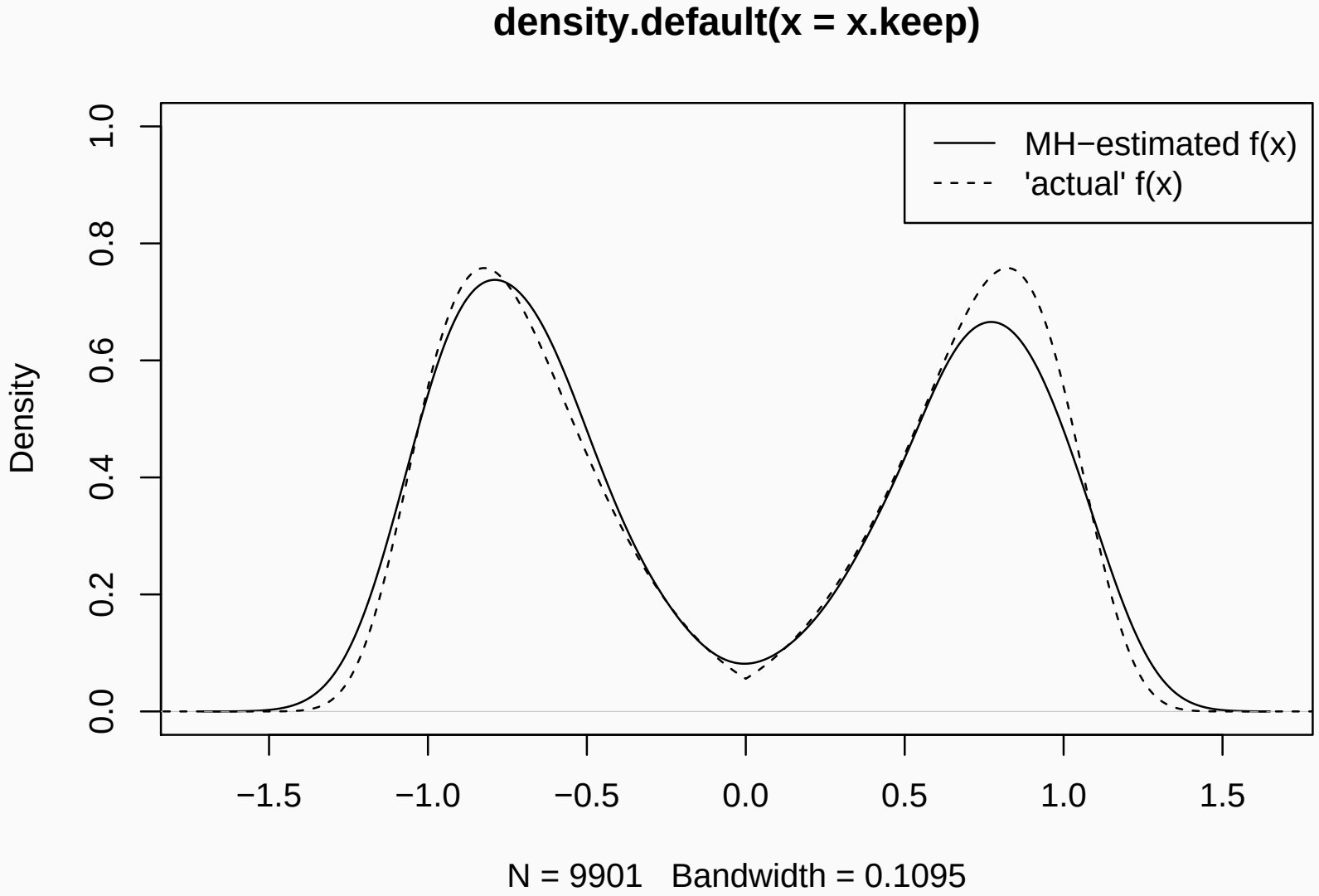
Problem: How to evaluate the integral of $g(x)$ if it is complicated? Workaround: A numerical integration procedure in R (the *integrate* function) yields a very accurate (nonstochastic) estimate of c.

```r
x.keep <- x[-(1:100)]  # discard burn-in part of sample

g <- function(x){ exp(-x^6) * (1 + abs(2 * x))^3 }
## Numerical integration of g from -Infinity to Infinity
(int.g <- integrate(g,-Inf,Inf))


## 17.91411 with absolute error < 0.00055

### Normalizing constant is approximately
### c=1/int.g$value
### Thus, f(x) approx g(x)/int.g$value
plot(density(x.keep), ylim=c(0, 1.0))
lines(tt <- seq(-4, 4, length=1001),
      (1/int.g$value) * g(tt), lty=2)
legend("topright",c("MH-estimated f(x)",
                    "'actual' f(x)"),
       lty=c(1,2))
```


density.default(x = x.keep)
MH−estimated f(x)
'actual' f(x)
Density
0.0
0.2
0.4
0.6
0.8
1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
N = 9901 Bandwidth = 0.1095

Additional Considerations for Metropolis Algorithm: Log scale and Supports

Working on (natural) logarithm scale

Particularly with the Metropolis algorithm, it is sometimes necessary to work on the log scale. Why? Sometimes the Metropolis algorithm acceptance ratio is the quotient of two very, very small (or two very, very large) numbers.

Why is the log scale preferable? Consider $\frac{301!}{300!}$. Easy to verify analytically that this equals 301. But in R?

```
factorial(301)/factorial(300)
```

```
## [1] NaN
```

What went wrong?

Each number in the quotient was very large, too large for R to reliably evaluate and store:

```
factorial(301)
```

```
## [1] Inf
```

```
factorial(300)
```

```
## [1] Inf
```

However, their true ratio is moderately sized ($301!/300! = 301$).

How to fix it?

For any positive values a and b, we can express a as $a = exp\{log(a)\}$ and b as $b = exp\{log(b)\}$. Thus,

$$\frac{a}{b} = \frac{exp\{log(a)\}}{exp\{log(b)\}} = exp\{log(a) - log(b)\}$$

In particular,

$$\frac{301!}{300!} = exp\{log(301!) - log(300!)\}$$

There is an R function dedicated to evaluating the (natural) logarithm of a factorial: *lfactorial*. Observe:

```
exp( lfactorial(301) - lfactorial(300) )
```

```
## [1] 301
```

> Throughout this class (and in nearly all statistical instances), by "log" we mean the natural logarithm, also known as *ln*().

More robust implementation of Metropolis Algorithm

```
### logg is the log of the function g(x)
#g <- function(x){ exp(-x^6) * (1+abs(2*x))^3 }
logg <- function(x){-x^6 +3*log(1+abs(2*x))}

## Initialize the sequence:
x<- rep(NA, 10001)
x[1] <- -15
set.seed(363)
for (j in 2:length(x)){
  xcurr <- x[j-1]
  xprop <- rnorm(1,xcurr,0.5)
  #alpha <- min(1, g(xprop)/g(xcurr))
  alpha <- min(1, exp(logg(xprop) -logg(xcurr)))
  x[j] <- ifelse(runif(1) <= alpha, xprop, xcurr)
}
```

Make sure to include supports when defining g.

I have made the support of distributions a point of emphasis. In writing $f(x)$ and $g(x)$, make sure the supports are included, or else havoc can ensue.

Suppose $x \sim Beta(2, 1)$. Then

$$f(x) = 2x\mathbb{1}_{(0<x<1)}$$

The normalizing constant here is known to equal 2, but recall $g(x)$ does not need the normalizing constant. So we could correctly use the Metropolis algorithm with

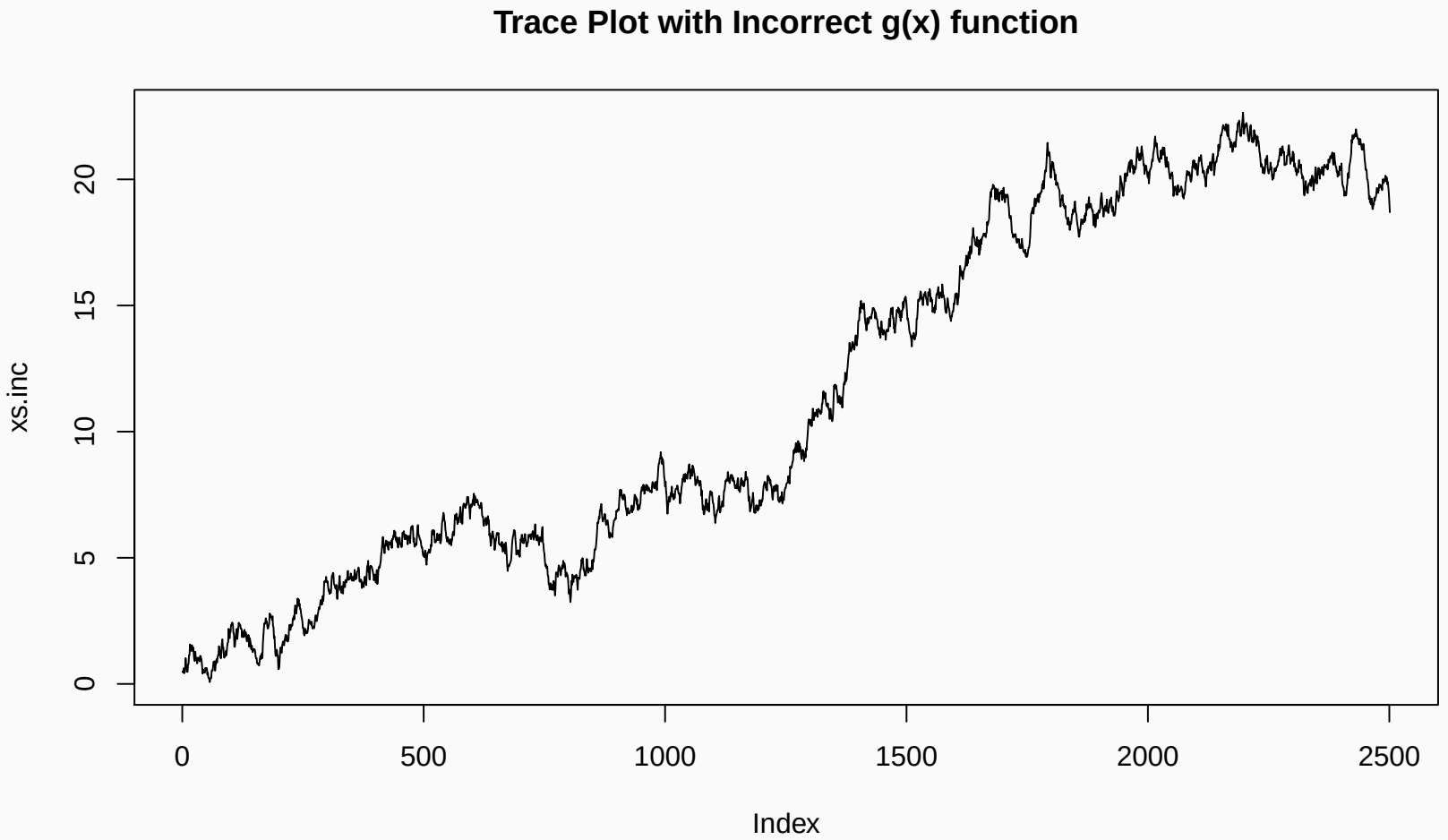
$$g_{correct}(x) = x\mathbb{1}_{(0<x<1)}$$

But if we were to incorrectly use $g_{Incorrect}(x) = x$, what would happen?

Havoc when supports ignored!

```
g.incorrect <- function(x) {x}
g.correct <- function(x){ x * as.numeric(x>0 & x< 1)}
xs.inc <- rep(NA, 2501)
xs.inc[1] <- 0.5
for (j in 2:2501){
  xcurr <- xs.inc[j-1]
  xprop <- rnorm(1, xcurr, 0.2)
  acc.ratio <- min(1,
                   g.incorrect(xprop)/g.incorrect(xcurr))
  xs.inc[j] <- ifelse( runif(1) <= acc.ratio,
                   xprop,
                   xcurr)
}
```

```
plot(xs.inc, type="l", main=
     "Trace Plot with Incorrect g(x) function")
```

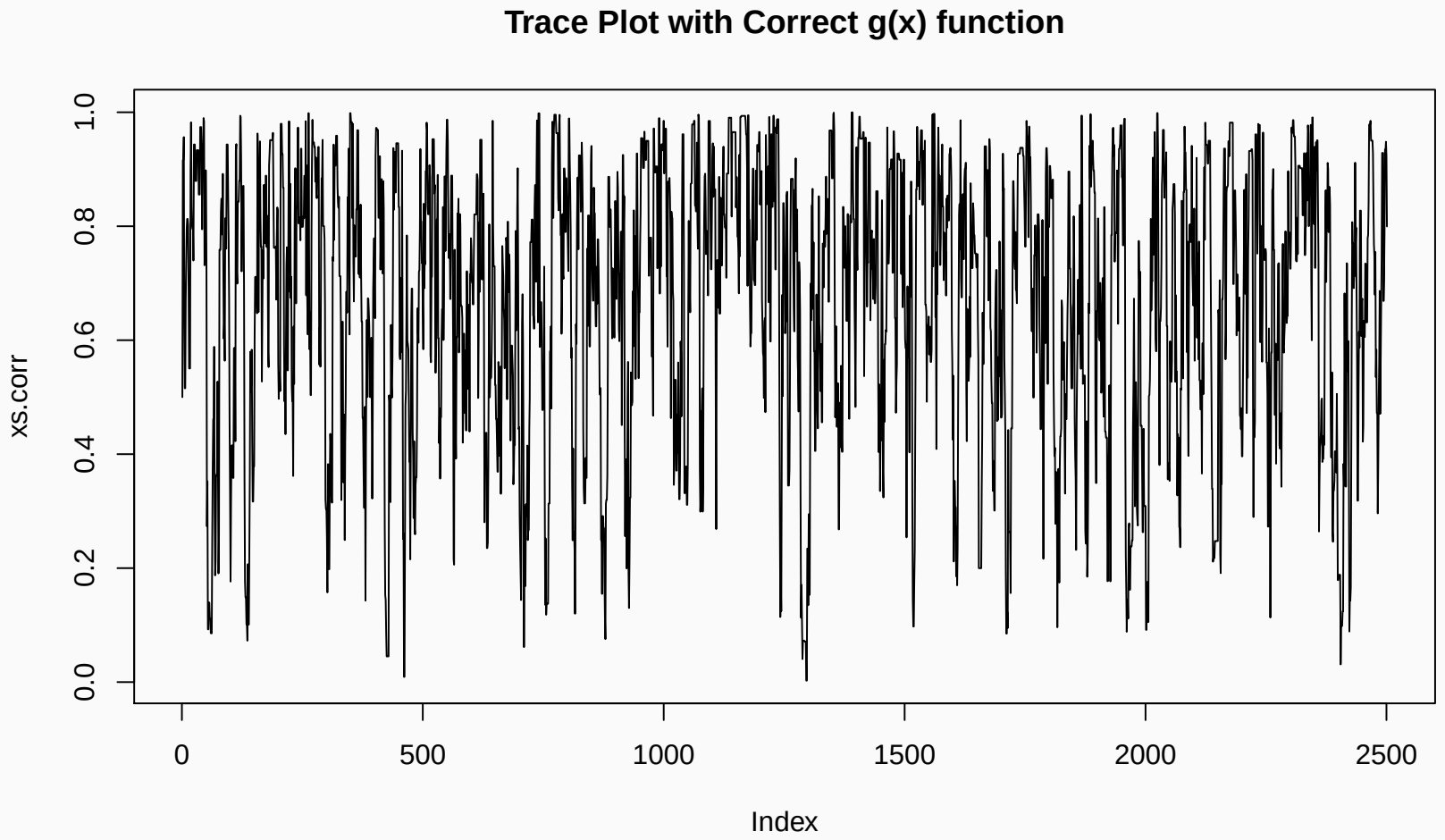


Beta(2,1) random variables must be between 0 and 1, but look at the xs in this sequence!

Beauty when supports included!

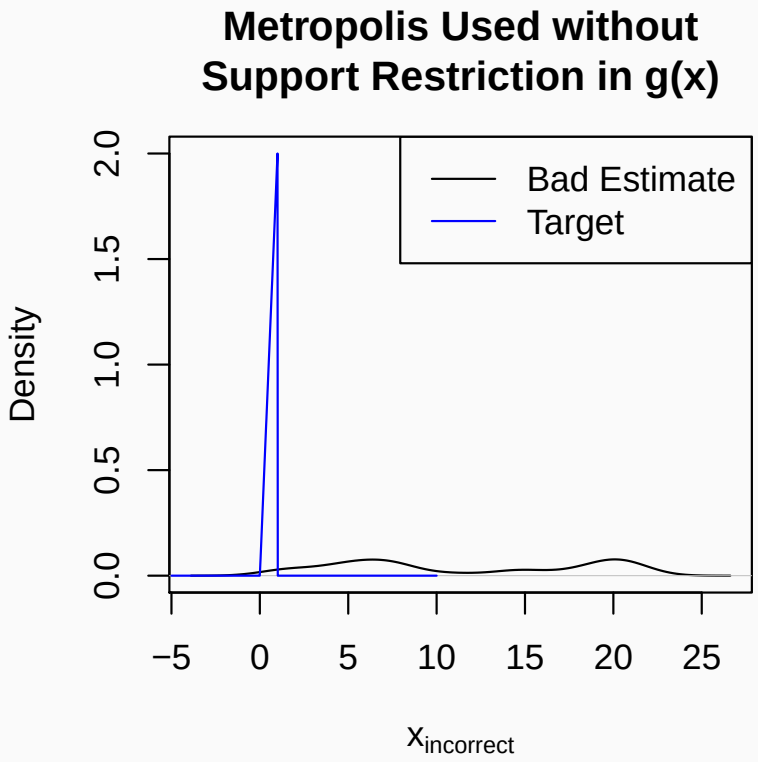
```
g.incorrect <- function(x) {x}
g.correct <- function(x){ x * as.numeric(x>0 & x< 1)}
xs.corr <- rep(NA, 2501)
xs.corr[1] <- 0.5
for (j in 2:2501){
  xcurr <- xs.corr[j-1]
  xprop <- rnorm(1, xcurr, 0.2)
  acc.ratio <- min(1,
                   g.correct(xprop)/g.correct(xcurr))
  xs.corr[j] <- ifelse( runif(1) <= acc.ratio,
                   xprop,
                   xcurr)
}
```

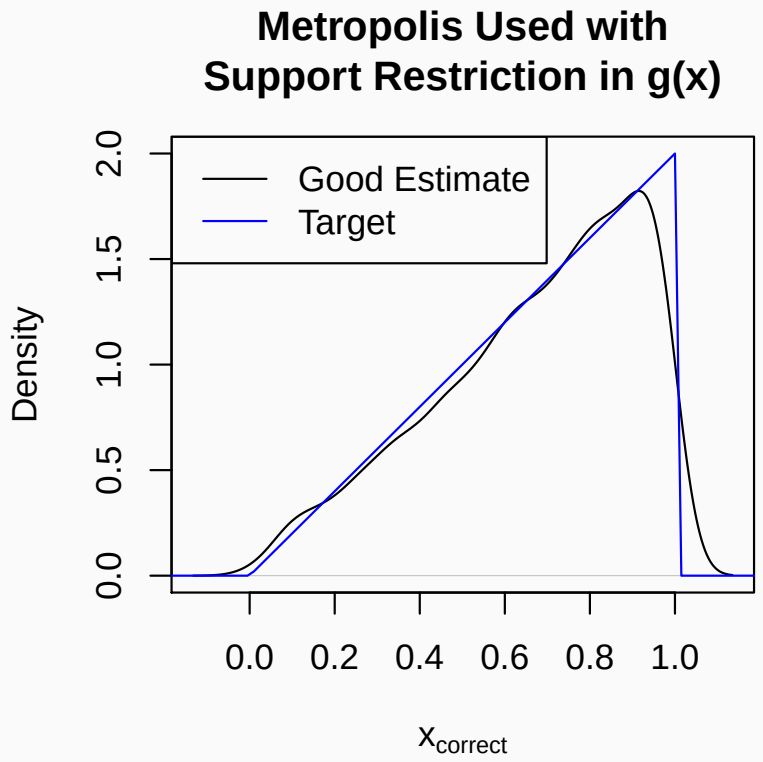
```
plot(xs.corr, type="l", main=
     "Trace Plot with Correct g(x) function")
```



Beta(2,1) random variables must be between 0 and 1, and look at the xs in this sequence!

Comparison of Metropolis-Estimated Densities





Why is it okay to only identify $g(x)$?

How target density features in implementing Metropolis Algorithm

- Directly impacts which families of proposal distributions are reasonable (support of proposal should contain support of target density)
- Involved in calculation of acceptance ratio: $\alpha = \min(1, f(x_{prop})/f(x_{curr}))$
- For a given family of proposal distributions, affects which values of the tuning parameter are reasonable

Why is it okay to only identify $g(x)$ (that is, why is it not necessary to know the normalizing constant, c, for the algorithm to work)?

- Support of $g(x)$ is exactly the same as support of $f(x)$
- Acceptance ratio should really be $\alpha = \min(1, f(x_{prop})/f(x_{curr}))$, but note that this equals $\min(1, [cg(x_{prop})]/[cg(x_{curr})]) = \min(1, g(x_{prop})/g(x_{curr}))$.
- Probability of accepting proposal is same whether we are using $f(x)$ or $g(x)$ in numerator/denominator of acceptance ratio, and therefore optimal tuning parameter is invariant to whether using $f(\cdot)$ or $g(\cdot)$.

Compare with the previous slide to see that using $g(x)$ is sufficient to generate a random sample from the distribution with density $f(x)$.

Your Turn to Practice

Your Turn: Modify provided code to do the following:

- The target distribution is $f(x) = c \exp[-x^6](1 + |2x|)^3$.
  - Start the sequence at -3 and run as before. Examine the trace plot for convergence.
  - Start the sequence at -3 and change the variance of the proposal distribution to be $0.2^2 = 0.04$ instead of $0.5^2 = 0.25$. Examine the trace plot and acf plot, and compute the acceptance proportions.
  - Start the sequence at -3 and change the variance of the proposal distribution to be 2^2 instead of 0.5^2. Examine the trace plot and acf plot, and compute the acceptance proportions.
  - Change the number of iterations to be 100K and rerun. Compare the estimated density to the target density function, $f(x)$.
  - Try starting the sequence at -15 and running as before. Is there any problem? If so, how could it be fixed?