Stats 102C - Lecture 4-2: Rejection Sampling

Miles Chen, acknowledgements Michael Tsiang

Week 4 Wednesday

Section 1

Rejection Sampling

Rejection Sampling

Rejection sampling is another method for generating random values from a distribution.

We might use rejection sampling when R does not have a built in function and when we can't use inverse CDF method because $F(x)$ or $F^{-1}(x)$ can't be derived.

Rejection Sampling

Goal: generate a random sample from f. We know the PDF of f.

Problem: We don't know $F^{-1}(x)$. We can't generate directly from f.

Requirements:

- Find a candidate distribution $g(x)$ that we can sample from.
- The support ($\mathcal{X}$) of f must be fully contained inside the support of g.
- There is a constant M such that $Mg(x) \geq f(x)$ for all $x \in \mathcal{X}$.

Because we require $Mg(x) \geq f(x)$, the most efficient choice of M is max of $\frac{f(x)}{g(x)}$ (and nothing larger). Anything less than the max of $\frac{f(x)}{g(x)}$ will not satisfy the condition $Mg(x) \geq f(x)$

Rejection Sampling

Algorithm:

1. Generate $X \sim g(x)$
2. Calculate

$$r(X) = \frac{f(X)}{Mg(X)}$$

3. Generate $U \sim \text{Unif}(0, 1)$
4. If $U \leq r(X)$ accept X as a sample from $f(X)$. Otherwise, reject X and repeat.

Example: Folded Normal

Suppose we have the PDF of the folded normal:

$$f(x) = 2 \cdot \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \text{ for } x \geq 0$$

For our candidate distribution, we will use Exponential(1) distribution, which has PDF

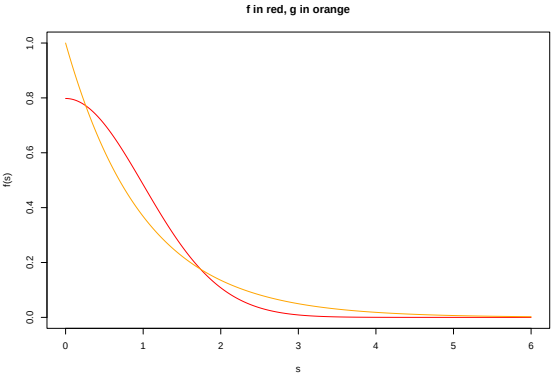
$$g(x) = e^{-x}, \text{ for } x \geq 0$$

The support is compatible with f.

Folded Normal

We must find a constant M such that $Mg(x) \geq f(x)$ for all $x \in \mathcal{X}$. The most efficient choice of M is max of $\frac{f(x)}{g(x)}$

```
f <- function(x){2 * dnorm(x)}
g <- function(x){dexp(x)}
s <- seq(0, 6, by = 0.005)
plot(s, f(s), type = "l", col = "red", main = "f in red, g in orange", ylim = c(0, 1))
lines(s, g(s), col = "orange")
```



Folded Normal

We must find a constant M such that $Mg(x) \geq f(x)$ for all $x \in \mathcal{X}$. The most efficient choice of M is max of $\frac{f(x)}{g(x)}$

$$\begin{aligned} M &= \max \frac{f(x)}{g(x)} \\ &= \max \frac{\frac{2}{\sqrt{2\pi}} e^{-x^2/2}}{e^{-x}} \\ &= \max \frac{2}{\sqrt{2\pi}} e^{-(x^2/2 - x)} \end{aligned}$$

The max of $e^{-(x^2/2-x)}$ occurs at the minimum of $(x^2/2 - x)$. With a little calculus, we can see the minimum occurs at $x = 1$.

We choose M to be equal to the function at $x = 1$:

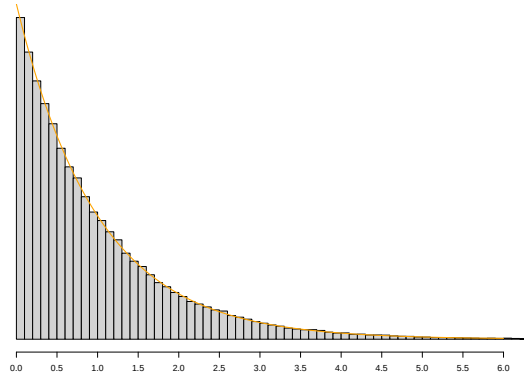
$$M = \frac{2}{\sqrt{2\pi}} e^{-(1/2-1)} = \frac{2}{\sqrt{2\pi}} e^{1/2} \approx 1.32$$

Run the algorithm

```
M <- 1.32
# f <- function(x){2 * dnorm(x)} # 2 * normal PDF
# g <- function(x){dexp(x)} # exponential PDF
set.seed(1)
N <- 10^5
proposed_x <- rexp(N) # proposal is the exponential dist
r_x <- f(proposed_x)/(M*g(proposed_x))
U <- runif(N)
accepted <- U < r_x
accepted_x <- proposed_x[accepted]
```

A little bit of intuition

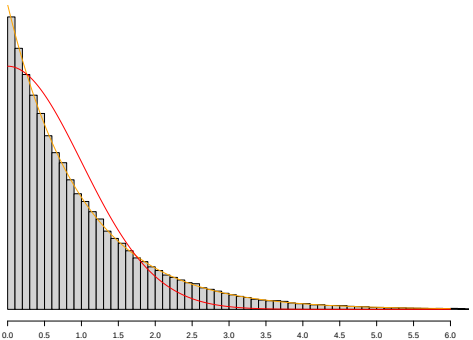
We generate random values from the distribution $g(x)$. In our case, this is the exponential distribution.



This histogram shows what we would expect: values that are exponentially distributed. Values near 0 appear frequently. Values like 2 show up with less frequency, and large values like 6 appear much less frequently.

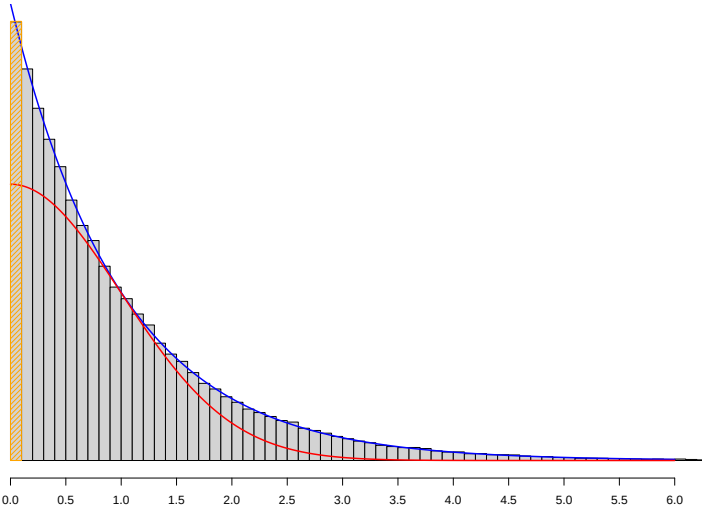
But our goal is a sample that looks like the folded-normal distribution.

What we want to do is keep only some of the values. The rest will be rejected or thrown away. We want to keep the values in such a way that what remains will look like the folded normal distribution.



If I use the exponential density and folded-normal density directly, we run into a problem. The exponential distribution does not produce "enough" values around 0.5 and 1.5.

The solution is to "scale" the density function. I can scale down the folded normal density by multiplying it by a constant $\frac{1}{M}$. This is mathematically equivalent to multiplying the exponential density by the constant M.



With the density function multiplied by a constant, the folded-normal density fits entirely under the exponential density.

Look at the left-most bar in the histogram. It spans [0, 0.1] and is centered at 0.05.

We look at the left-most bar in the histogram. It spans [0, 0.1] and is centered at 0.05.

For the folded normal distribution, the density at 0.05 is:

$$f(0.05) = 2 \cdot \frac{1}{\sqrt{2\pi}} e^{-.05^2/2} \approx 0.79689$$

For the scaled exponential distribution the density at 0.05 is:

$$Mg(0.05) = (1.32)e^{-0.05} \approx 1.25562$$

So for that first bar, we only want to accept some of the values.

$$r(0.05) = \frac{f(0.05)}{Mg(0.05)} \approx 0.6347$$

```
2*dnorm(0.05) / (1.32 * dexp(0.05))
```

```
## [1] 0.6346554
```

So `rexp()` produces a value between 0 and 0.1 too frequently compared to the normal distribution.

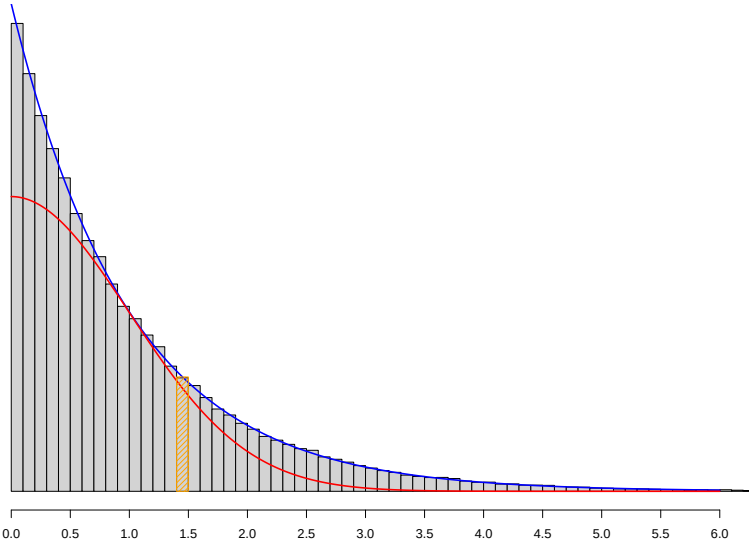
So if `rexp()` proposes a value of 0.05, we will accept only about $r(0.05) \approx 63.5\%$ of them.

How do we ensure that we accept a value like 0.05 with a probability of 63.5%?

We use `runif()` to generate U.

If $U \sim$ Unif, then $U < r(X)$ with probability $r(X)$.

If $U > r(X)$, we reject the proposed value of 0.05 and repeat the process.



Look at the bar that spans [1.4, 1.5] and is centered at 1.45.

For the folded normal distribution, the density at 1.45 is:

$$f(1.45) = 2 \cdot \frac{1}{\sqrt{2\pi}} e^{-1.45^2/2} \approx 0.27886$$

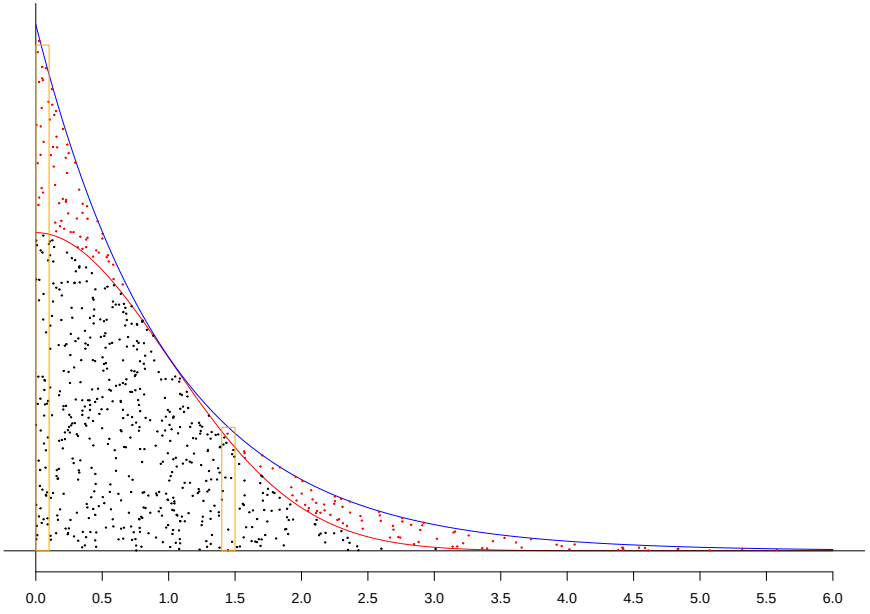
For the scaled exponential distribution the density at 1.45 is:

$$Mg(1.45) = (1.32)e^{-1.45} \approx 0.30963$$

Whenever the exponential distribution proposes a value near 1.45, we will accept the vast majority of them.

$$r(1.45) = \frac{f(1.45)}{Mg(1.45)} \approx 0.9006$$

We do this by generating $U \sim$ Unif with `runif()` and accepting the proposed value if $U < r(X)$



Empirical estimates

```
sum(proposed_x < 0.1) # values between 0 and 0.1 that were proposed by rexp
```

```
## [1] 9596
```

```
sum(accepted_x < 0.1) # number of values accepted
```

```
## [1] 6003
```

```
sum(accepted_x < 0.1) / sum(proposed_x < 0.1) # approximate ratio
```

```
## [1] 0.6255732
```

```
p <- sum( (proposed_x > 1.4) & (proposed_x < 1.5) ); print(p)
```

```
## [1] 2328
```

```
a <- sum( (accepted_x > 1.4) & (accepted_x < 1.5) ); print(a)
```

```
## [1] 2107
```

```
a/p
```

```
## [1] 0.9050687
```

```
length(accepted_x)/length(proposed_x) # overall acceptance ratio
```

```
## [1] 0.75519
```

A more mathematically rigorous proof

We can take a more mathematical approach to show that rejection sampling works.

We need to show that even though we use the function $g(x)$ to propose random values of X, if we only accept some of the proposed values with probability $r(X) = \frac{f(X)}{Mg(X)}$, then the distribution will be equivalent to $f(x)$

We must show $\Pr(X = x | X \text{ is accepted}) = f(x)$

We use Bayes Rule. Keep in mind that the denominator of Bayes' rule can be found by integrating the numerator across all possible values of X

$$\begin{aligned}
\Pr(X = x|X \text{ is accepted}) &= \frac{\Pr(X \text{ is accepted}|X = x)\Pr(X = x)}{\Pr(X \text{ is accepted})} \\
&= \frac{\Pr[U \leq r(x)] \cdot g(x)}{\int_{\mathcal{X}} \Pr[U \leq r(x)] \cdot g(x)dx} \\
&= \frac{\frac{f(x)}{Mg(x)} \cdot g(x)}{\int_{\mathcal{X}} \frac{f(x)}{Mg(x)} \cdot g(x)dx} \\
&= \frac{\frac{f(x)}{M}}{\frac{1}{M}} \\
&= f(x)
\end{aligned}$$

which is what we wanted to show.

Rejection Sampling vs Importance Sampling

Rejection sampling and importance sampling have similarities but are different.

Both methods deal with the problem of not being able to generate values from distribution $f(x)$ directly. Both methods solve the problem by taking a sample from another distribution $g(x)$ and using a ratio (either f/g or $f/(Mg)$).

Rejection sampling is a method used to **generate a random sample** from some distribution f.

Importance sampling is used to **estimate the expected value of a function**.

Rejection sampling throws away values that do not meet the acceptance criteria. Importance sampling does not throw any values away. Instead, importance sampling weighs some values less important than others.

Personally, I think the name "importance sampling" is not a good name. It should be called "weighted Monte Carlo Integration."