STAT 33B Homework 4

Nick Ulle

This assignment is due **March 20, 2020** by 11:59pm.

The purpose of this assignment is to practice writing functions, if-statements, and loops.

Edit this file, knit to PDF, and:

- Submit the Rmd file on bCourses.
- Submit the PDF file on Gradescope.

If you think you'll need help with submission, please ask in office hours *before* the assignment is due.

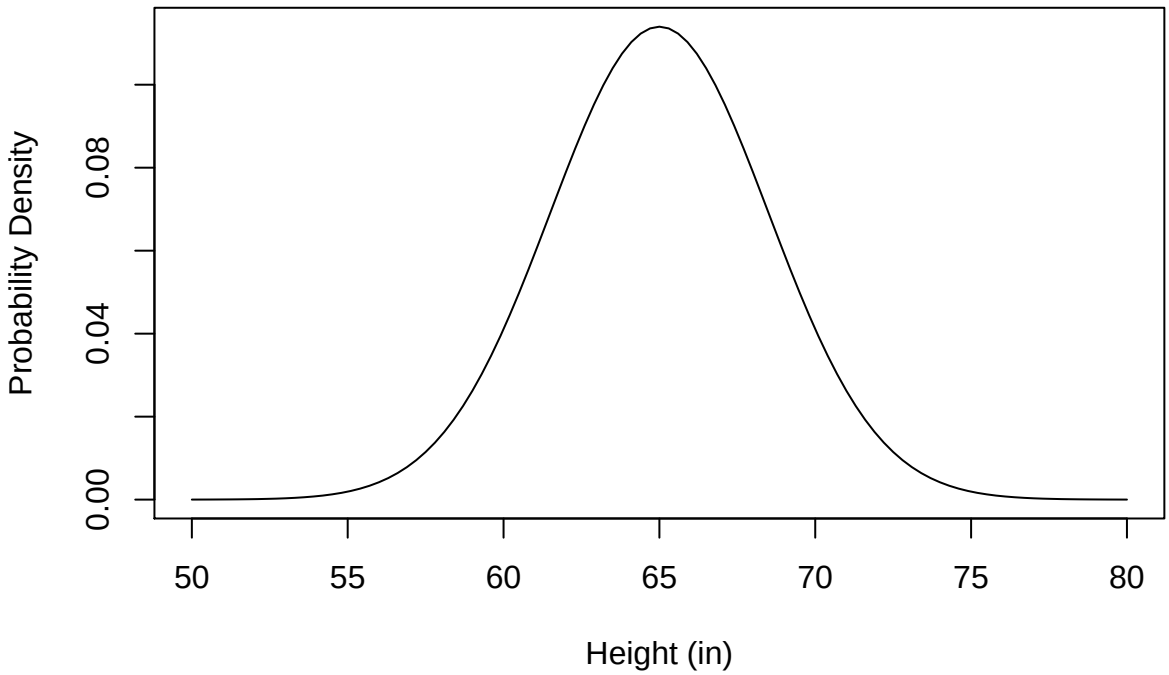
Answer all questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like. Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

Rejection Sampling

Many scientific simulations depend on being able to randomly sample values that follow a particular distribution.

For example, the heights of adult women in the United States approximately follow a normal distribution (a "bell curve") with center (mean) at 65 inches and spread (standard deviation) 3.5 inches. Here's the code to plot that distribution:

```
curve(dnorm(x, 65, 3.5), 50, 80, xlab = "Height (in)",
  ylab = "Probability Density")
```



Heights where the probability density is higher are more likely in a random sample of women's heights.

In R, we can randomly sample values from a normal distribution with the `rnorm()` function. The code to sample 10 values from the noraml distribution with mean 65 and standard deviation 3.5 is:

```
rnorm(10, 65, 3.5)
```

```
##  [1] 63.78077 62.25939 63.05953 64.88162 70.33632 67.09713 61.32496 60.90618
##  [9] 64.44533 66.31758
```

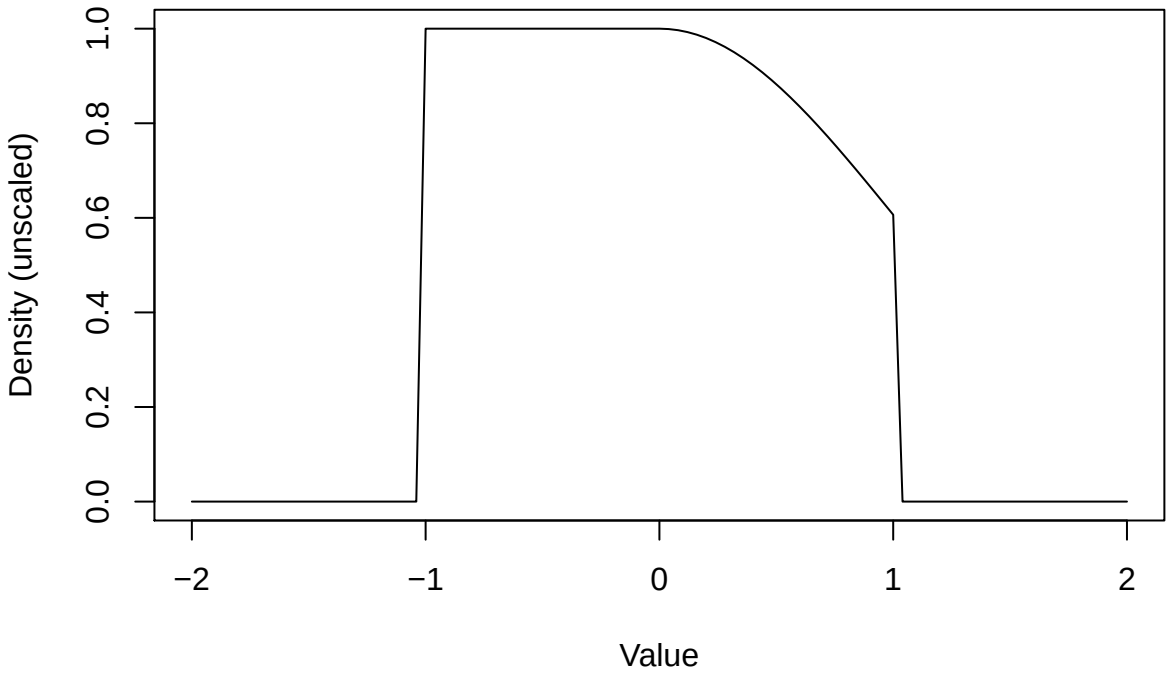
R provides functions to sample from a variety of well-known distributions. The names of these functions usually begin with `r` for "random". For instance, `rbinom()` samples from a binomial distribution, and `runif()` samples from a (continuous) uniform distribution.

What if we want to sample from a distribution that's not well-known?

For example, suppose we want to sample from the distribution on -1 to 1 shown in the plot produced by this code:

```
dslide = function(x) {
  ifelse(x > 1, 0,
    ifelse(x > 0, dnorm(x) / dnorm(0), dunif(x, -1, 0))
  )
}

curve(dslide, -2, 2, xlab = "Value", ylab = "Density (unscaled)")
```



The distribution is flat (uniform) between -1 and 0, then smoothly curves down between 0 and 1. In other words, values closer to 1 are less likely than values between -1 and 0. We'll call this distribution the "slide" distribution, since it resembles the silhoutte of a playground slide.

We can sample from distributions that are not well-known by using a statistical technique called rejection sampling. The idea is to choose a shape that completely encloses the density curve for the distribution, and then randomly, uniformly sample points within that shape. If a point falls below the density curve, then the point is accepted and its x-coordinate is a new sample value. If a point falls above the density curve, then it is rejected (and discarded). This produces the correct distribution because relatively more points will be accepted in places where the density curve is taller.

It's convenient to make the enclosing shape a rectangle, because then we can sample x and y coordinates for points with the `runif()` function. For instance, for the slide distribution, we can use a rectangle with opposite corners (-1, 0) and (1, 1).

The exact steps in rejection sampling are:

1. Sample an x coordinate.
2. Sample a y coordinate.
3. Test whether the y value falls below the target distribution's density curve. If it does, then x is a new sample value. If it does not, then x is rejected.
4. Repeat steps 1-3 until reaching the desired number of sample values.

Exercise 1

Write a rejection sampler for the slide distribution. You can use the `dslide()` function provided above to compute the height of the distribution's density curve at a given x-coordinate.

```
# Your code goes here.

n = 1e6
samp = numeric(n)
accepted = 0
while (accepted < n) {
  x = runif(1, -1, 1)
  y = runif(1)
```

```
  if (y < dslide(x)) {
    accepted = accepted + 1
    samp[accepted] = x
  }
}
```

Exercise 2

Turn your rejection sampler from Exercise 1 into a function called `rslide()`. Your function should return the final sampled values, and should have a parameter `n` that controls the number of values to sample.

```
# Your code goes here.

rslide = function(n) {
  samp = numeric(n)
  accepted = 0
  while (accepted < n) {
    x = runif(1, -1, 1)
    y = runif(1)

    if (y < dslide(x)) {
      accepted = accepted + 1
      samp[accepted] = x
    }
  }

  samp
}
```

Exercise 3

In R, you can use `plot(density(x))` to plot the estimated density curve for a given sample `x`. Sample 1 million points from the slide distribution and plot the estimated density.

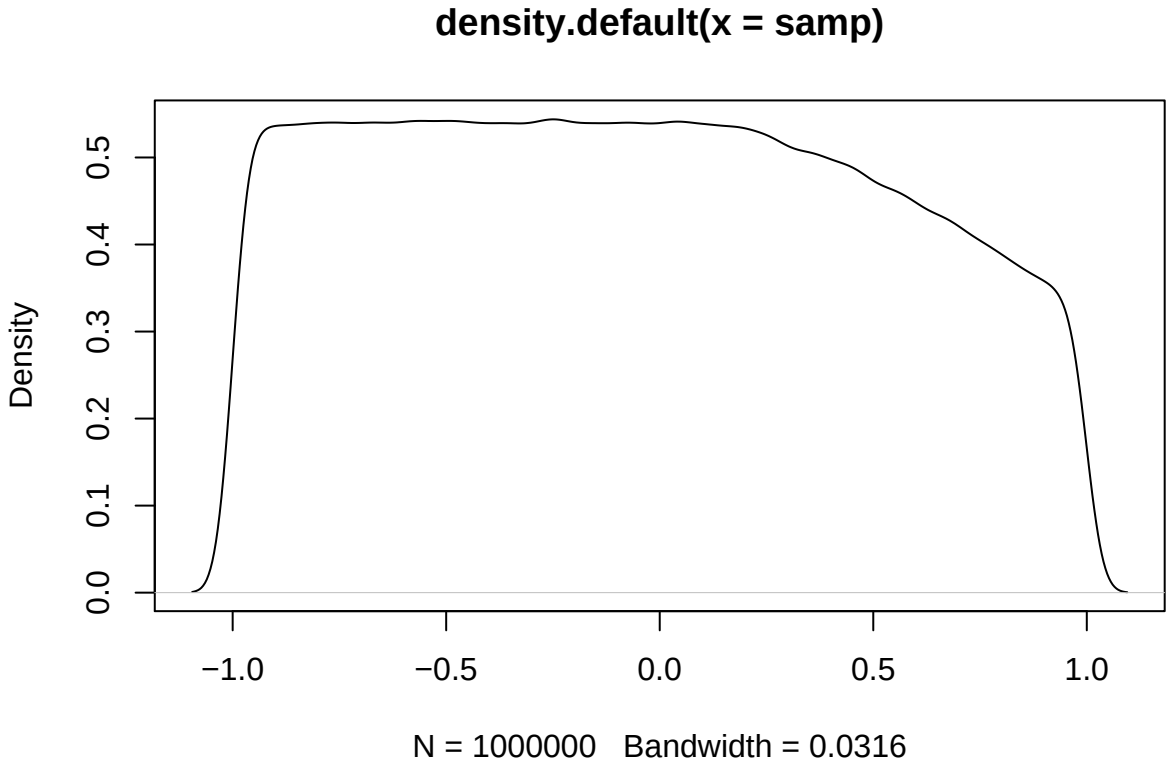
How does the shape of your estimated density curve compare to the shape of the actual density curve for the slide distribution (see above)?

Note: It's important for scientific results to be reproducible. You can set the "seed" in R's pseudorandom number generator with `set.seed()` to make sure anyone that runs your code will get the same result, even though the result is based on random sampling. You can add a call to `set.seed()` before you call `rslide()` here to ensure the output in your knitted PDF will be the same every time.

```
# Your code goes here.

set.seed(33)
samp = rslide(1e6)

plot(density(samp))
```



YOUR WRITTEN ANSWER GOES HERE:

The estimated density curve is slightly rougher than the actual density curve. The tails of the estimated density curve are longer, even though none of the sampled points fall outside of the -1 to 1 range. Overall, the estimate is a good match and suggests the sampler is working correctly.

Exercise 4

Use a sample of 1 million points to estimate the mean and standard deviation of the slide distribution.

```
summary(samp)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.00000 -0.53580 -0.07288 -0.05814  0.40005  1.00000
```

```
sd(samp)
```

```
## [1] 0.5564798
```

Exercise 5

Create a new version of your `rslide()` function called `rslide_rejected()`. Instead of returning the sampled vaues, the `rslide_rejected()` function should keep track of and return the number of points that were rejected.

What percentage of points is typically rejected for this rejection sampler? Use an average across multiple runs to get a more accurate estimate. The `replicate()` function may be helpful for calling `rslide_rejected()` multiple times.

```
# Your code goes here.

rslide_rejected = function(n) {
  accepted = 0
```

```
  rejected = 0
  while (accepted < n) {
    x = runif(1, -1, 1)
    y = runif(1)

    if (y < dslide(x)) {
      accepted = accepted + 1
    } else {
      rejected = rejected + 1
    }
  }

  rejected
}

n = 1000
nreject = replicate(100, rslide_rejected(n))

mean(nreject / (n + nreject))
```

```
## [1] 0.07294729
```

YOUR WRITTEN ANSWER GOES HERE:

Based on 100 runs, each sampling 1000 points, approximately 7% of the points are rejected.