Chapter 3: Methods for Simulating Data

Statisticians (and other users of data) need to simulate data for many reasons.

For example, I simulate as a way to check whether a model is appropriate. If the observed data are similar to the data I generated, then this is one way to show my model may be a good one.

It is also sometimes useful to simulate data from a distribution when I need to estimate an expected value (approximate an integral). - ch. 5

R can already generate data from many (named) distributions:

```
set.seed(400) #reproducibility
```

```
rnorm(10) # 10 observations of a N(0,1) r.v.
```

```
##  [1] -1.0365488  0.6152833  1.4729326 -0.6826873 -0.6018386 -1.3526097
##  [7]  0.8607387  0.7203705  0.1078532 -0.5745512
```

```
rnorm(10, 0, 5) # 10 observations of a N(0,5^2) r.v.
```

```
##  [1] -4.5092359  0.4464354 -7.9689786 -0.4342956 -5.8546081  2.7596877
##  [7] -3.2762745 -2.1184014  2.8218477 -5.0927654
```

```
rexp(10) # 10 observations from an Exp(1) r.v.
```

```
##  [1] 0.67720831 0.04377997 5.38745038 0.48773005 1.18690322 0.92734297
##  [7] 0.33936255 0.99803323 0.27831305 0.94257810
```

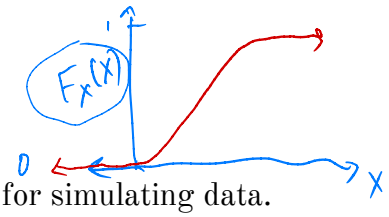
But what about when we don't have a function to do it?

↳ we need to write our own functions to simulate draws from distributions.

1 Inverse Transform Method

"PIT"

Theorem 1.1 (Probability Integral Transform) If X is a continuous r.v. with cdf F_X, then $U = F_X(X) \sim \text{Uniform}[0, 1]$.



This leads to to the following method for simulating data.

$U = F_X(X)$

$F_X^{-1}(U) = F_X^{-1}(F_X(X)) = X$

Inverse Transform Method:

First, generate u from Uniform$[0, 1]$. Then, $x = F_X^{-1}(u)$ is a realization from F_X.

Note: F^{-1} may not be available in closed form. If that's the case, use something else.

1.1 Algorithm

1. Derive the inverse function F_X^{-1}. To do this, let $F(x) = u$. Then solve for x to find $x = F^{-1}(u)$.
2. Write a function to compute $x = F_X^{-1}(u)$.
   ↳ in R
3. For each realization, → simulated value.
   a. generate a random value u from unif$(0, 1)$.
   b. Compute $x = F^{-1}(u)$.

Typically repeat a-b many times.

Example 1.1 Simulate a random sample of size 1000 from the pdf $f_X(x) = 3x^2, 0 \le x \le 1$.

1. Find the cdf F.

$$F(x) = \int_0^x 3y^2 dy = y^3 \Big|_0^x = x^3 \quad x \in [0,1].$$

2. Find F^{-1}

$$u = F(x) = x^3 \Rightarrow u^{1/3} = x = F^{-1}(u) \qquad 0 \le u \le 1$$

↑ range of $F(x)$.

3.
```
# write code for inverse transform example
# f_X(x) = 3x^2, 0 <= x \<= 1
```

① Write a function for F^{-1}
② Sample 1000 u values from Unif(0,1)
③ evaluate $x_i = F^{-1}(u_i)$ for $i = 1, \ldots, 1000$.

1.2 Discrete RVs

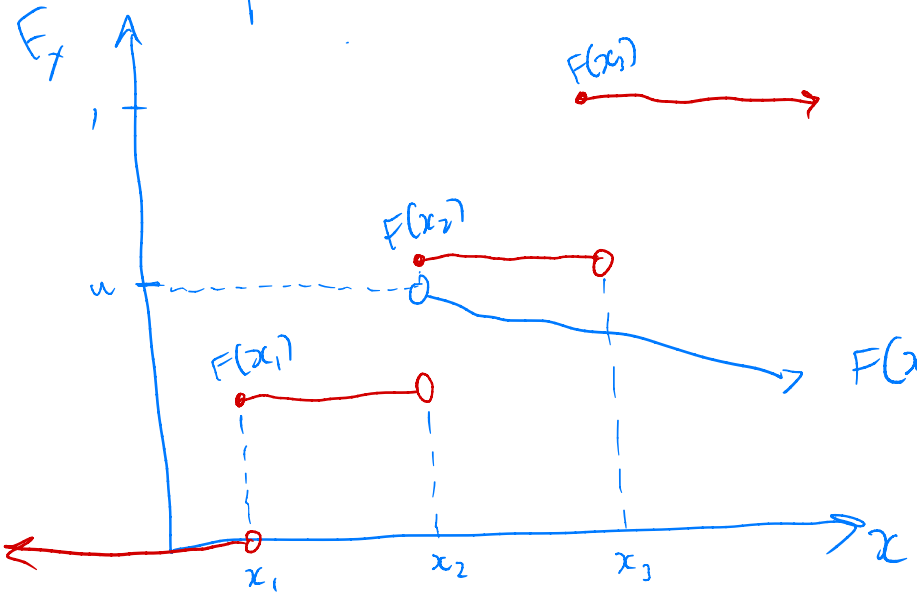
If X is a discrete random variable and $\cdots < x_{i-1} < x_i < \cdots$ are the points of discontinuity (↙ where cdf jumps) of $F_X(x)$, then the inverse transform is $F_X^{-1}(u) = x_i$ where $F_X(x_{i-1}) < u \le F_X(x_i)$. This leads to the following algorithm:

1. Generate a r.v. U from Unif$(0,1)$.
2. Select x_i where $F_X(x_{i-1}) < U \le F_X(x_i)$.

} repeat many times.

↑ discontinuity pt

① If $u = 0.5$ for example.



$F(x_1) < u \le F(x_2)$.

⇒ ② select x_2.

Example 1.2 Generate 1000 samples from the following discrete distribution.

```
x <- 1:3
p <- c(0.1, 0.2, 0.7)
```

x	1.0	2.0	3.0
f	0.1	0.2	0.7

```
# write code to sample from discrete dsn
n <- 1000
```

There is a simpler way using sample() function in R.

2 Acceptance-Reject Method

The goal is to generate realizations from a *target density*, f.

Most cdfs cannot be inverted in closed form.

The Acceptance-Reject (or "Accept-Reject") samples from a distribution that is *similar* to f and then adjusts by only accepting a certain proportion of those samples.

The method is outlined below:

Let g denote another density from which we **know how to sample** and we can **easily calculate** $g(x)$.

Let $e(\cdot)$ denote an *envelope*, having the property $e(x) = cg(x) \geq f(x)$ for all $x \in \mathcal{X} = \{x : f(x) > 0\}$ for a given constant $c \geq 1$.

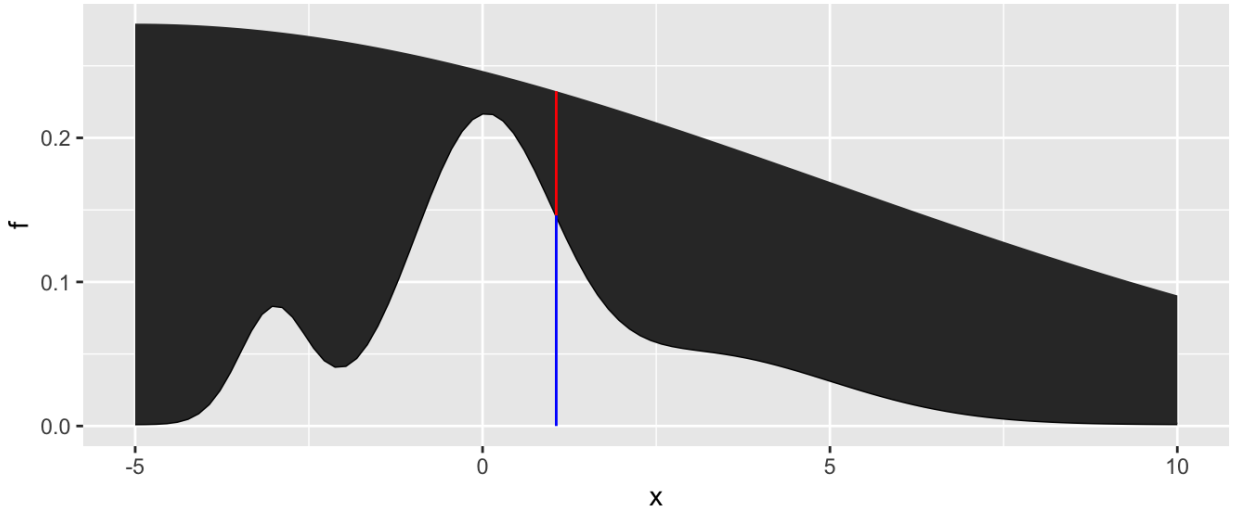
The Accept-Reject method then follows by sampling $Y \sim g$ and $U \sim \text{Unif}(0, 1)$.

If $U < f(Y)/e(Y)$, accept Y. Set $X = Y$ and consider X to be an element of the target random sample.

Note: $1/c$ is the expected proportion of candidates that are accepted.

2.1 Algorithm

1. Find a suitable density g and envelope e.
2. Sample $Y \sim g$.
3. Sample $U \sim \text{Unif}(0, 1)$.
4. If $U < f(Y)/e(Y)$, accept Y.
5. Repeat from Step 2 until you have generated your desired sample size.



2.2 Envelopes

Good envelopes have the following properties:

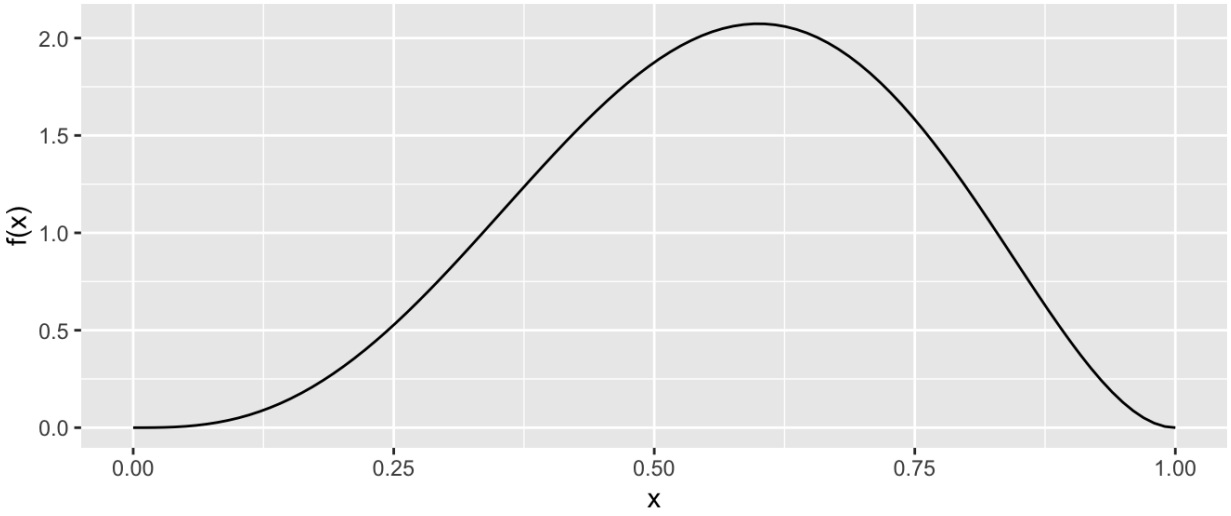
A simple approach to finding the envelope:

Example 2.1 We want to generate a random variable with pdf $f(x) = 60x^3(1-x)^2$, $0 \leq x \leq 1$. This is a Beta$(4, 3)$ distribution.

Can we invert $F(x)$ analytically?

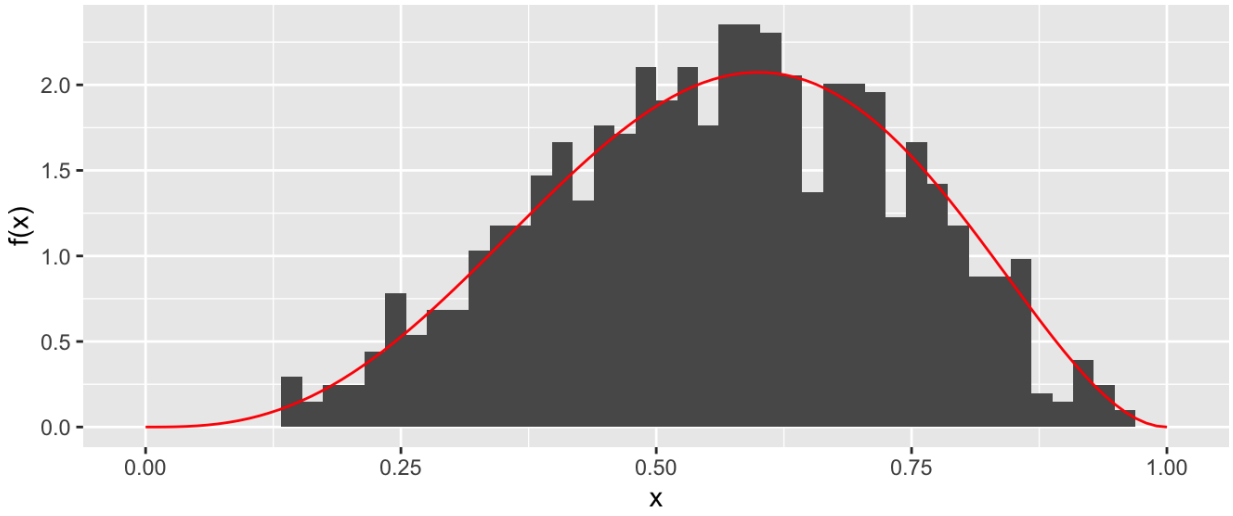
If not, find the maximum of $f(x)$.

```
# pdf function, could use dbeta() instead
f <- function(x) {
    60*x^3*(1-x)^2
}

# plot pdf
x <- seq(0, 1, length.out = 100)
ggplot() +
  geom_line(aes(x, f(x)))
```



```
envelope <- function(x) {
  ## create the envelope function
}

# Accept reject algorithm
n <- 1000 # number of samples wanted
accepted <- 0 # number of accepted samples
samples <- rep(NA, n) # store the samples here

while(accepted < n) {
  # sample y from g

  # sample u from uniform(0,1)
  u <- runif(1)

  if(u < f(y)/envelope(y)) {
    # accept
    accepted <- accepted + 1
    samples[accepted] <- y
  }
}
```

```
ggplot() +
  geom_histogram(aes(sample, y = ..density..), bins = 50, ) +
  geom_line(aes(x, f(x)), colour = "red") +
  xlab("x") + ylab("f(x)")
```



2.3 Why does this work?

Recall that we require

$$cg(y) \geq f(y) \qquad \forall y \in \{y : f(y) > 0\}.$$

Thus,

The larger the ratio $\frac{f(y)}{cg(y)}$, the more the random variable Y looks like a random variable distributed with pdf f and the more likely Y is to be accepted.

2.4 Additional Resources

See p.g. 69-70 of Rizzo for a proof of the validity of the method.

3 Transformation Methods

We have already used one transformation method – **Inverse transform method** – but there are many other transformations we can apply to random variables.

1. If $Z \sim N(0,1)$, then $V = Z^2 \sim$
2. If $U \sim \chi^2_m$ and $V \sim \chi^2_n$ are independent, then $F = \frac{U/m}{V/n} \sim$
3. If $Z \sim N(0,1)$ and $V \sim \chi^2_n$ are independendent, then $T = \frac{Z}{\sqrt{V/n}} \sim$
4. If $U \sim \text{Gamma}(r, \lambda)$ and $V \sim \text{Gamma}(s, \lambda)$ are independent, then $X = \frac{U}{U+V} \sim$

Definition 3.1 A *transformation* is any function of one or more random variables.

Sometimes we want to transform random variables if observed data don't fit a model that might otherwise be appropriate. Sometimes we want to perform inference about a new statistic.

Example 3.1 If $X_1, \ldots, X_n \overset{iid}{\sim} \text{Bernoulli}(p)$. What is the distribution of $\sum_{i=1}^n X_i$?

Example 3.2 If $X \sim N(0,1)$, what is the distribution of $X + 5$?

Example 3.3 For $X_1, \ldots, X_n$ iid random variables, what is the distribution of the median of $X_1, \ldots, X_n$? What is the distribution of the order statistics? $X_{[i]}$?

There are many approaches to deriving the pdf of a transformed variable.

But the theory isn't always available. What can we do?

3.1 Algorithm

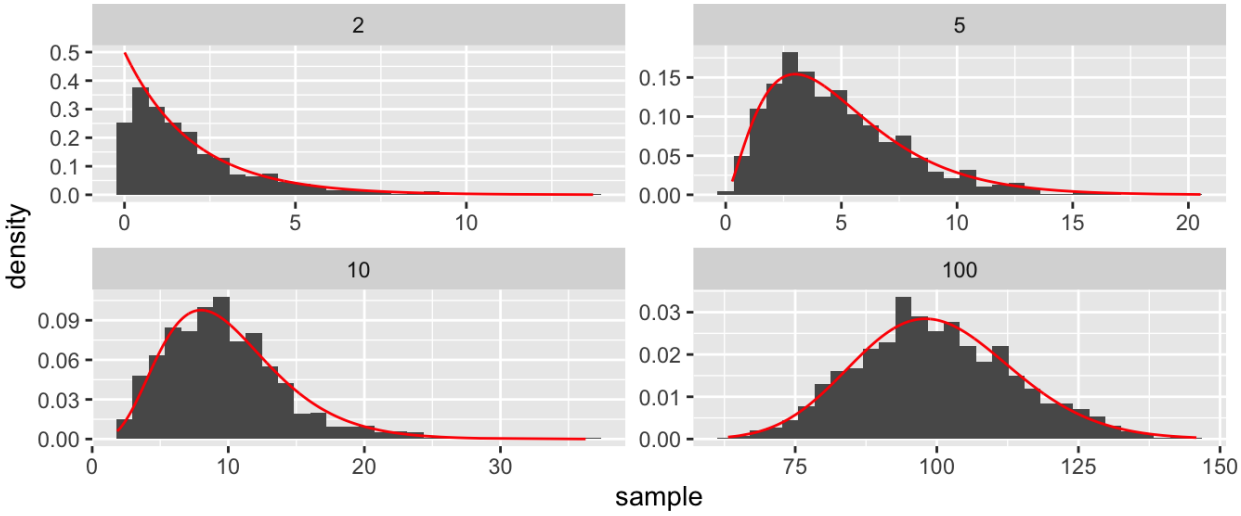
Let $X_1, \ldots, X_p$ be a set of independent random variables with pdfs $f_{X_1}, \ldots, f_{X_p}$, respectively, and let $g(X_1, \ldots, X_p)$ be some transformation we are interested in simulating from.

1. Simulate $X_1 \sim f_{X_1}, \ldots, X_p \sim f_{X_p}$.
2. Compute $G = g(X_1, \ldots, X_p)$. This is one draw from $g(X_1, \ldots, X_p)$.
3. Repeat Steps 1-2 many times to simulate from the target distribution.

Example 3.4 It is possible to show for $X_1, \ldots, X_p \overset{iid}{\sim} N(0,1)$, $Z = \sum_{i=1}^{p} X_i^2 \sim \chi^2_p$. Imagine that we cannot use the `rchisq` function. How would you simulate Z?

```
library(tidyverse)

# function for squared r.v.s
squares <- function(x) x^2

sample_z <- function(n, p) {
  # store the samples
  samples <- data.frame(matrix(rnorm(n*p), nrow = n))

  samples %>%
    mutate_all("squares") %>% # square the rvs
    rowSums() # sum over rows
}

# get samples
n <- 1000 # number of samples

# apply our function over different degrees of freedom
samples <- data.frame(chisq_2 = sample_z(n, 2),
                      chisq_5 = sample_z(n, 5),
                      chisq_10 = sample_z(n, 10),
```

```
                chisq_100 = sample_z(n, 100))

# plot results
samples %>%
  gather(distribution, sample, everything()) %>% # make easier to
  plot w/ facets
  separate(distribution, into = c("dsn_name", "df")) %>% # get the df
  mutate(df = as.numeric(df)) %>% # make numeric
  mutate(pdf = dchisq(sample, df)) %>% # add density function values
  ggplot() + # plot
  geom_histogram(aes(sample, y = ..density..)) + # samples
  geom_line(aes(sample, pdf), colour = "red") + # true pdf
  facet_wrap(~df, scales = "free")
```



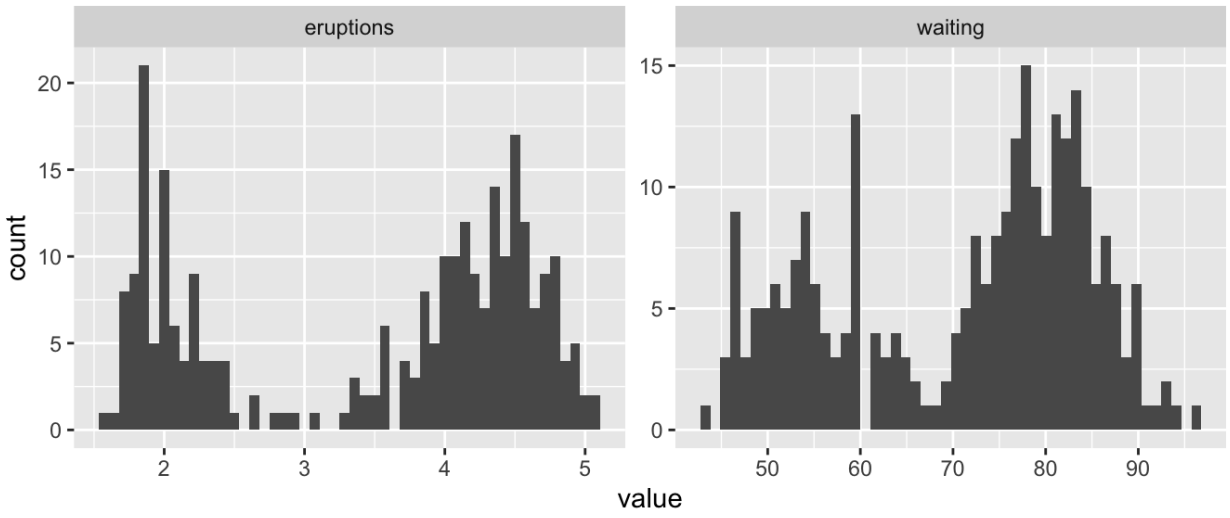
4 Mixture Distributions

The `faithful` dataset in `R` contains data on eruptions of Old Faithful (Geyser in Yellowstone National Park).

```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```
faithful %>%
  gather(variable, value) %>%
  ggplot() +
  geom_histogram(aes(value), bins = 50) +
  facet_wrap(~variable, scales = "free")
```



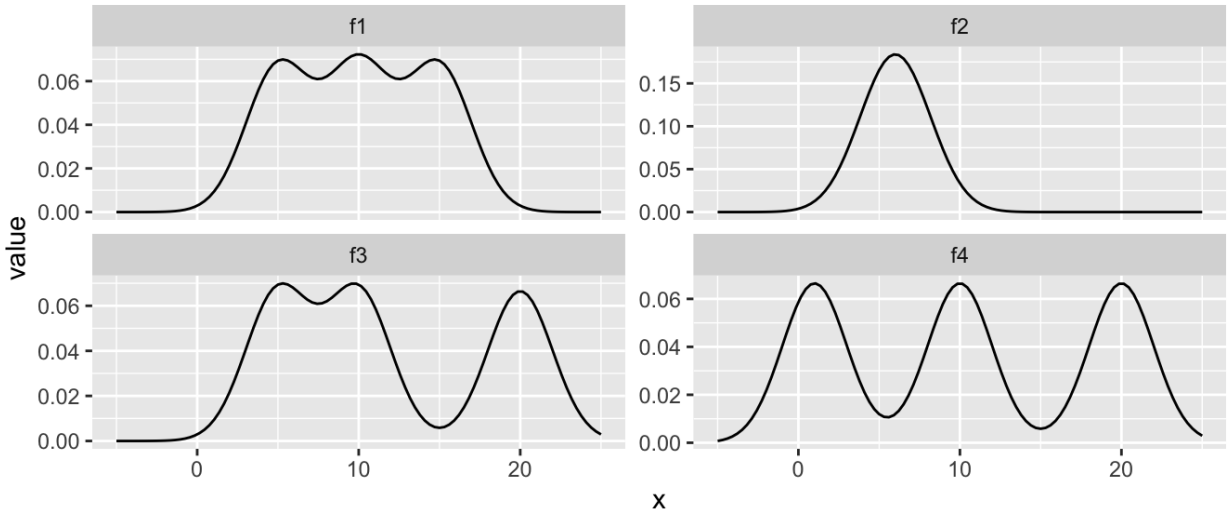
What is the shape of these distributions?

Definition 4.1 A random variable Y is a discrete mixture if the distribution of Y is a weighted sum $F_Y(y) = \sum \theta_i F_{X_i}(y)$ for some sequence of random variables $X_1, X_2, \ldots$ and $\theta_i > 0$ such that $\sum \theta_i = 1$.

For 2 r.v.s,

Example 4.1

```
x <- seq(-5, 25, length.out = 100)

mixture <- function(x, means, sd) {
  # x is the vector of points to evaluate the function at
  # means is a vector, sd is a single number
  f <- rep(0, length(x))
  for(mean in means) {
    f <- f + dnorm(x, mean, sd)/length(means) # why do I divide?
  }
  f
}

# look at mixtures of N(mu, 4) for different values of mu
data.frame(x,
           f1 = mixture(x, c(5, 10, 15), 2),
           f2 = mixture(x, c(5, 6, 7), 2),
           f3 = mixture(x, c(5, 10, 20), 2),
           f4 = mixture(x, c(1, 10, 20), 2)) %>%
  gather(mixture, value, -x) %>%
  ggplot() +
  geom_line(aes(x, value)) +
  facet_wrap(.~mixture, scales = "free_y")
```



4.1 Mixtures vs. Sums

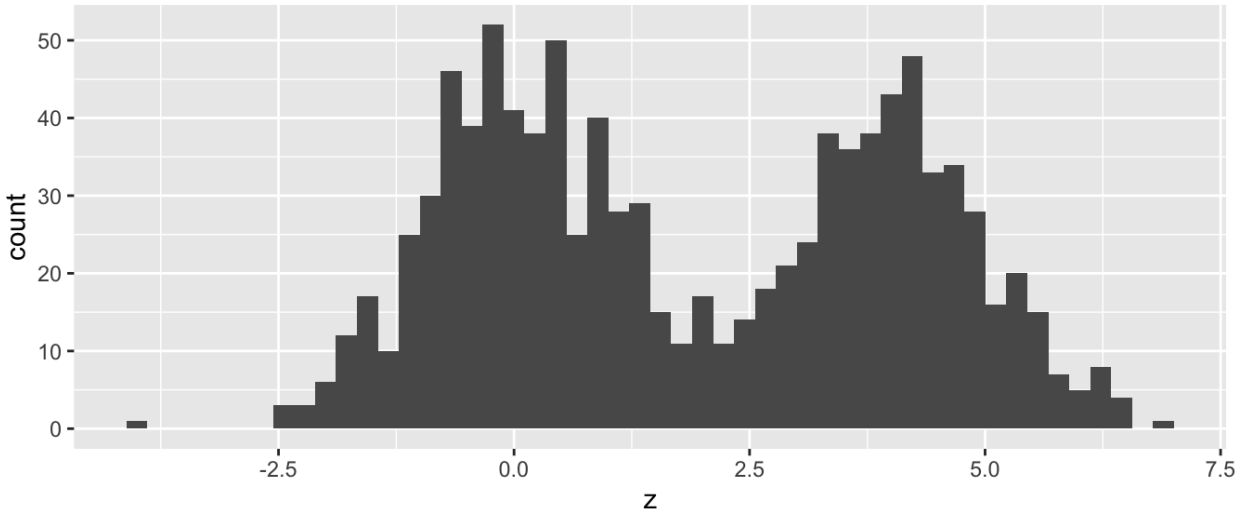
Note that mixture distributions are *not* the same as the distribution of a sum of r.v.s.

Example 4.2 Let $X_1 \sim N(0,1)$ and $X_2 \sim N(4,1)$, independent.

$S = \frac{1}{2}(X_1 + X_2)$

Z such that $f_Z(z) = 0.5 f_{X_1}(z) + 0.5 f_{X_2}(z)$.

```
n <- 1000
u <- rbinom(n, 1, 0.5)

z <- u*rnorm(n) + (1 - u)*rnorm(n, 4, 1)

ggplot() +
  geom_histogram(aes(z), bins = 50)
```



What about $f_Z(z) = 0.7 f_{X_1}(z) + 0.3 f_{X_2}(z)$?

4.2 Models for Count Data (refresher)

Recall that the Poisson(λ) distribution is useful for modeling count data.

$$f(x) = \frac{\lambda^x \exp\{-\lambda\}}{x!}, \quad x = 0, 1, 2, \ldots$$

Where X = number of events occuring in a fixed period of time or space.

When the mean λ is low, then the data consists of mostly low values (i.e. $0, 1, 2$, etc.) and less frequently higher values.

As the mean count increases, the skewness goes away and the distribution becomes approximately normal.

With the Poisson distribution,

$$E[X] = VarX = \lambda.$$

Example 4.3

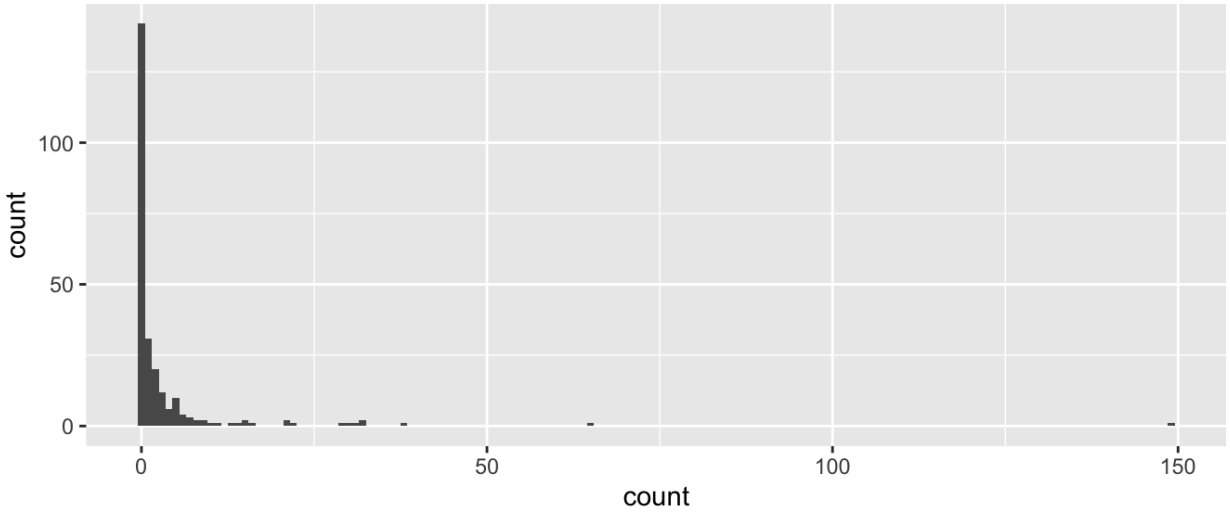
Example 4.4 The Colorado division of Parks and Wildlife has hired you to analyze their data on the number of fish caught in Horsetooth resevoir by visitors. Each visitor was asked - How long did you stay? - How many fish did you catch? - Other questions: How many people in your group, were children in your group, etc.

Some visiters do not fish, but there is not data on if a visitor fished or not. Some visitors who did fish did not catch any fish.

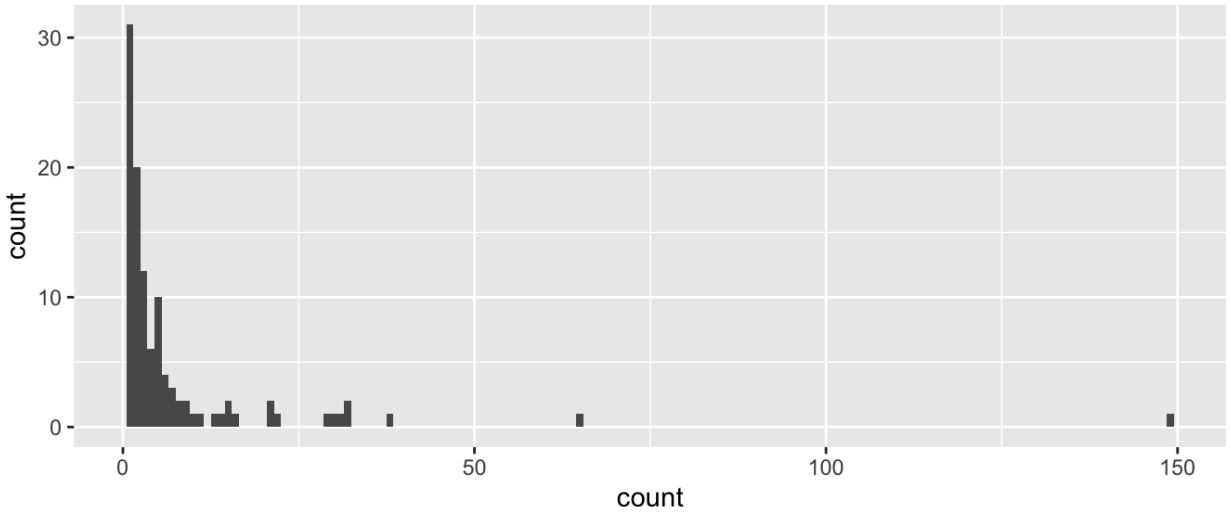
Note, this is modified from https://stats.idre.ucla.edu/r/dae/zip/.

```
fish <- read_csv("https://stats.idre.ucla.edu/stat/data/fish.csv")
```

```
# with zeroes
ggplot(fish) + geom_histogram(aes(count), binwidth = 1)
```




```
# without zeroes
fish %>%
  filter(count > 0) %>%
  ggplot() +
  geom_histogram(aes(count), binwidth = 1)
```



A *zero-inflated* model assumes that the zero observations have two different origins – structural and sampling zeroes.

Example 4.5

A zero-inflated model is a **mixture model** because the distribution is a weighted average of the sampling model (i.e. Poisson) and a point-mass at 0.

For $Y \sim ZIP(\lambda)$,

$$Y \sim \begin{cases} 0 & \text{with probability } \pi \\ \text{Poisson}(\lambda) & \text{with probability } 1-\pi \end{cases}$$

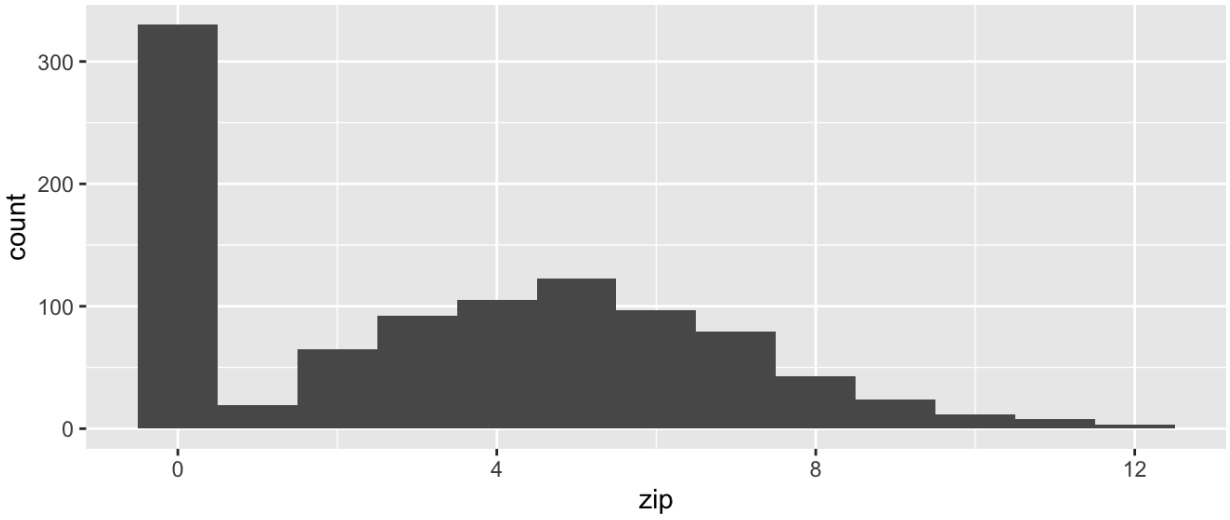
So that,

$$Y =$$

To simulate from this distribution,

```
n <- 1000
lambda <- 5
pi <- 0.3

u <- rbinom(n, 1, pi)
zip <- u*0 + (1-u)*rpois(n, lambda)
```

```
# zero inflated model
ggplot() + geom_histogram(aes(zip), binwidth = 1)
```




```
# Poisson(5)
ggplot() + geom_histogram(aes(rpois(n, lambda)), binwidth = 1)
```