Chapter 3: Methods for Simulating Data

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

→ Goodness of fit test.

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
```

→ set starting point for pseudo random number generator ⇒ reproduce results.

```
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
```

sd.

```
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
```

λ=1

```
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?

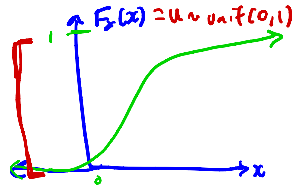
↳ we will need to write our own functions to simulate draws from other distributions.

1 Inverse Transform Method

easiest method
when it works

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

discrete version exists as well.



This leads to to the following method for simulating data.

Inverse Transform Method:

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

Note:

F_X^{-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 $F_X(x) = u$ Then solve for x to find $x = F_X^{-1}(u)$.

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 $x = F_X^{-1}(u)$
   simulated draw from $F_X(x)$.

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

1. Find the cdf F_X

$$F_X(x) = \int_0^x 3y^2\,dy = y^3\Big|_0^x = \begin{cases} 0 & x < 0 \\ x^3 & x \in [0,1]. \\ 1 & x > 1. \end{cases}$$

2. Find F_X^{-1}

$u = F_X(x) = x^3 \Rightarrow u^{1/3} = x = F_X^{-1}(u)$.

so $F_X^{-1}(u) = u^{1/3}$ for $0 \leq u \leq 1$ ← range of $F_X(x)$.

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

a) Write function for F^{-1}

b) sample u values from unif$(0,1)$] 1000 times.

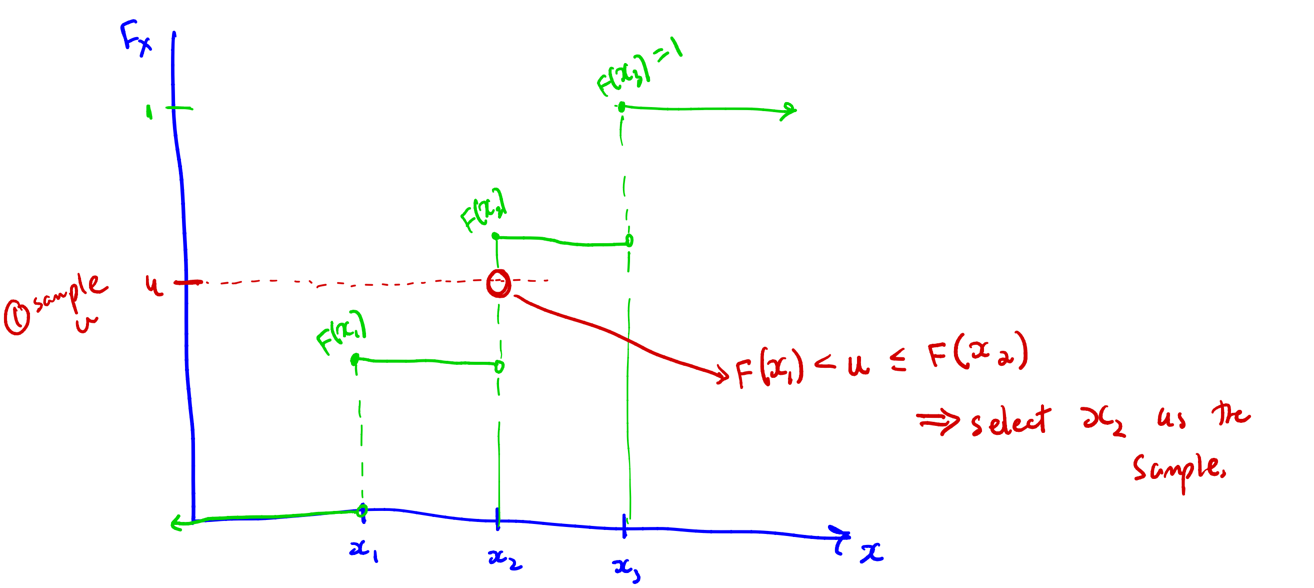
c) evaluate $x = F_X^{-1}(u)$.

1.2 Discrete RVs

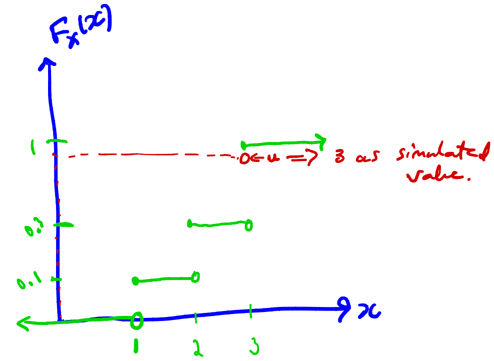
If X is a discrete random variable and $\cdots < x_{i-1} < x_i < \cdots$ are the points of discontinuity (, jumps.) of $F_X(x)$, then the inverse transform is $F_X^{-1}(u) = x_i$ (↑ jump point) where $F_X(x_{i-1}) < u \leq F_X(x_i)$. This leads to the following algorithm:

1. Generate a r.v. U from Unif$(0,1)$.

2. Select x_i (↑ jump point) where $F_X(x_{i-1}) < U \leq F_X(x_i)$.



Example 1.2 Generate 1000 samples from the following discrete distribution.




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

pmf

x	1.0	2.0	3.0
f	0.1	0.2	0.7

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

There is a really simple way to do this in R.

using sample()

* remember to allow replacement and specify the prob. vector.

Something we can try when we can't find F^{-1} (in closed form)

2 Acceptance-Reject Method

the distribution we want to sample from.

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.

target

and rejecting the rest.

The method is outlined below:

requirements for $g(\cdot)$.

Let g denote another density from which we ① **know how to sample** and we can ② **easily calculate** $g(x)$.

target

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 envelope covers all of f

support of target

③ This implies that the support of $g(\cdot)$ MUST INCLUDE the support of f!

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.

We can use this to evaluate the efficiency of our algorithm.

What might be hard/slow?

(slow) - depending on efficiency, we might have to draw a LOT of samples just to keep a few.

(slow) - could be slow to evaluate f

(hard) - choose g & e

2.1 Algorithm

hard step

1. Find a suitable density g (proposal) and envelope e.

2. Sample $Y \sim g$.

3. Sample $U \sim \text{Unif}(0, 1)$.

slow step*

4. If $U < f(Y)/e(Y)$, accept Y.

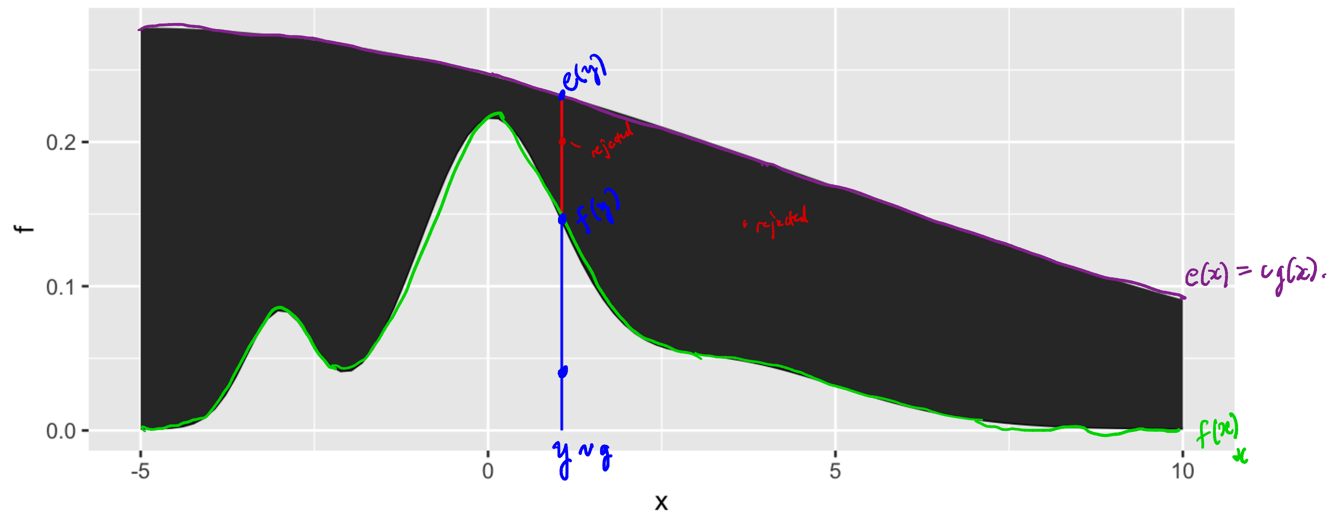
slow step.

5. Repeat from Step 2 until you have generated your desired sample size.

* Requirement: the support of g MUST INCLUDE the support of f *

(BAD) example: If $f \equiv N(0,2)$ and $g \equiv \text{Unif}(-10,10)$.

This would not be appropriate because support of f is $(-\infty, \infty)$ and support of g is $[-10,10]$.



2.2 Envelopes

Good envelopes have the following properties:

1. envelope exceeds target everywhere ← support of g must include support of f; choose c to make this ① happen.

2. Easy to sample from g.

3. Generate few rejected draws (save time).

This always works when $\mathcal{X} = [0,1]$.

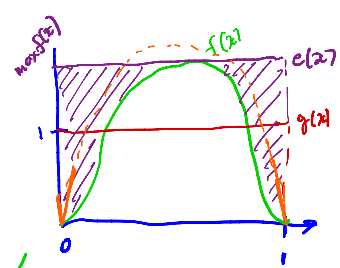
A simple approach to finding the envelope: ← in some cases.

Say the support of f is $\mathcal{X} = \{x : 0 \le x \le 1\}$

Find $\max_x (f(x))$ and let $c = \max_x (f(x))$

Let $g(x) \equiv \text{Unif}(0,1) = \begin{cases} 1 & \text{if } x \in [0,1] \\ 0 & \text{otherwise} \end{cases}$ ← support of g matches the support of f ✓

$e(x) = c\,g(x)$



This is often not efficient. If you know more about the shape of f you can select a better envelope.

Plotting is your friend here!

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

→ could just use rbeta() in R.

Can we invert $F(x)$ analytically?

No. Let's use accept-reject! let $g \sim \text{Unif}(0,1)$.

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

$$f'(x) = 60\left(3x^2(1-x)^2 - 2x^3(1-x)\right)$$
$$= 60x^2(1-x)\left[3(1-x) - 2x\right]$$
$$= 60x^2(1-x)(3-5x) = 0 \quad \text{when} \quad x=0,\ x=1, \text{ or } \boxed{x = \tfrac{3}{5}}$$

$f(0) = f(1) = 0.$

$$c = \max_x f(x) = f\left(\tfrac{3}{5}\right) = 2.0736$$

density for beta

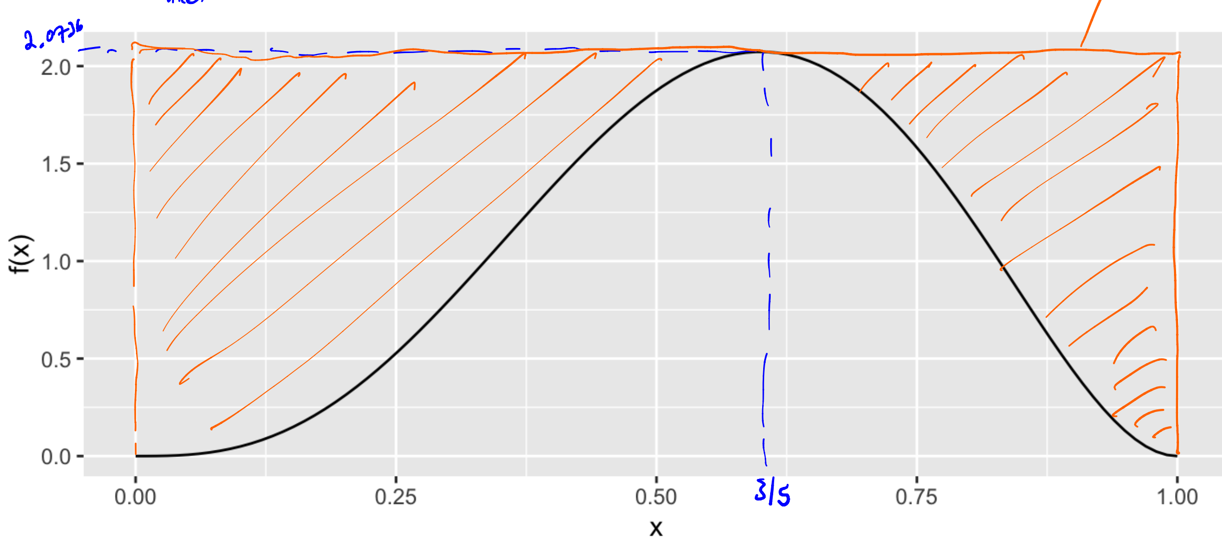
```
# pdf function, could use dbeta() instead
f <- function(x) {
  60*x^3*(1-x)^2
}

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

make x values

← f evaluated at x values

↑ line.

$$e(x) = \begin{cases} 2.0736 & x \in [0,1] \\ 0 & \text{o.w.} \end{cases}$$



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

c·unif pdf
= c·1
= f(3/5)

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

we don't know how many iterations it will take => for loop not helpful. run loop until we have accepted enough (n).

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

unif(0,1)

y <- runif(1).

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

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

increment accepted so loop ends eventually.

store samples.

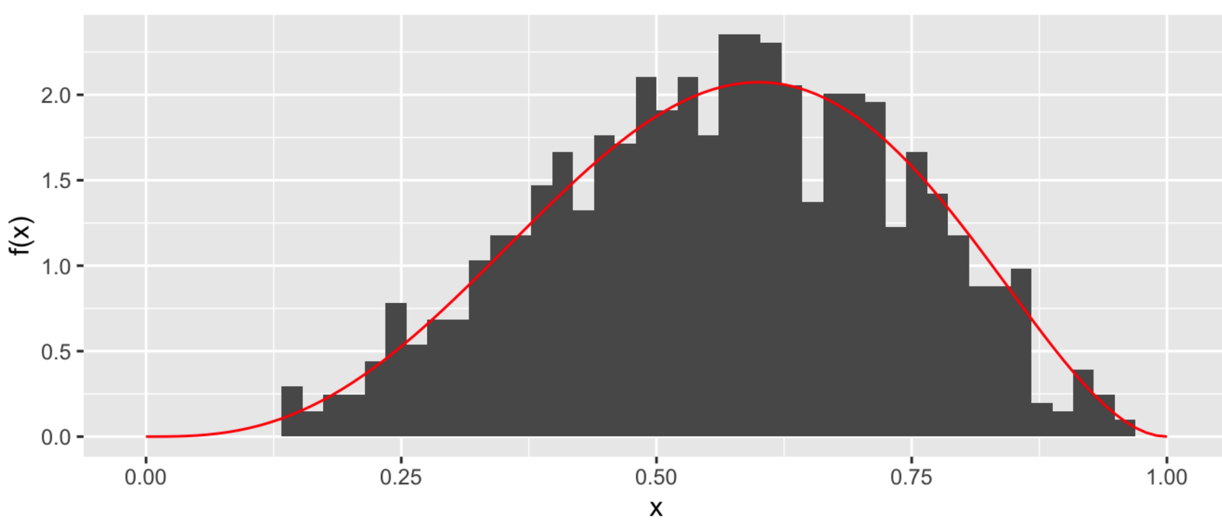
necessary so that histogram is on the same scale as the density function instead of raw counts.

samples from f

* important for next homework.

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

theoretical pdf



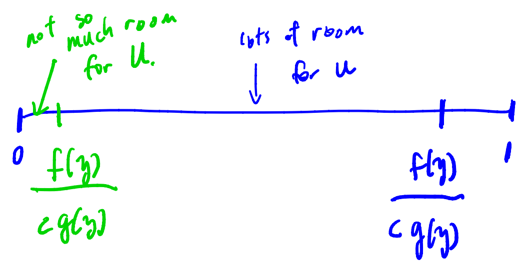
2.3 Why does this work?

Recall that we require

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

Thus,

$$0 \leq \frac{f(y)}{e(y)} = \frac{f(y)}{cg(y)} \leq 1$$



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

should be a copy w/ library.

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

How to choose envelope (if support is not $[0,1]$):

consulting a table of dist's (wikipedia)

(1) start w/ support of f. $\Rightarrow$ list of potential g's either w/ same support or larger.

making plots.

(2) plot f to get a sense of the shape. Try to pick a g from my list w/ similar shape.

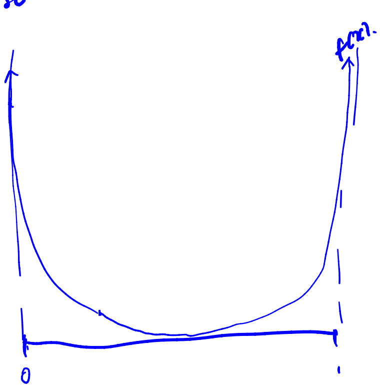
(3) pick c s.t. $cg(x) \geq f(x)$ $\forall x$.

↳ picking a bunch of c's, plotting $c \cdot g(x)$ vs. $f(x)$

evaluating $c \cdot g(x)$ vs. $f(x)$ at a wide range of x's.

Choose the smallest c I can that makes $c \cdot g(x) \geq f(x)$ $\forall x$.

$\boxed{\text{SORRY I Lied to you}}$



we can't evaluate max $f(x)$

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 $V = Z^2 \sim \chi^2_1$
2. If $U \sim \chi^2_m$ and $V \sim \chi^2_n$ are independent, then $F = \frac{U/m}{V/n} \sim F_{m,n}$
3. If $Z \sim N(0,1)$ and $V \sim \chi^2_n$ are independendent, then $T = \frac{Z}{\sqrt{V/n}} \sim t_n$
4. If $U \sim \text{Gamma}(r,\lambda)$ and $V \sim \text{Gamma}(s,\lambda)$ are independent, then $X = \frac{U}{U+V} \sim \text{Beta}(r,s)$.
5. If $X \sim F$, then $F^{-1}(X) \sim \text{Unif}(0,1)$. (PIT, leads to inverse transform method).

$X \longrightarrow g(X)$.

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

Can derive $\sum X_i \sim \text{Binomial}(n,p)$.

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

$X+5 \sim N(5,1)$.

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

This is more complex...

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

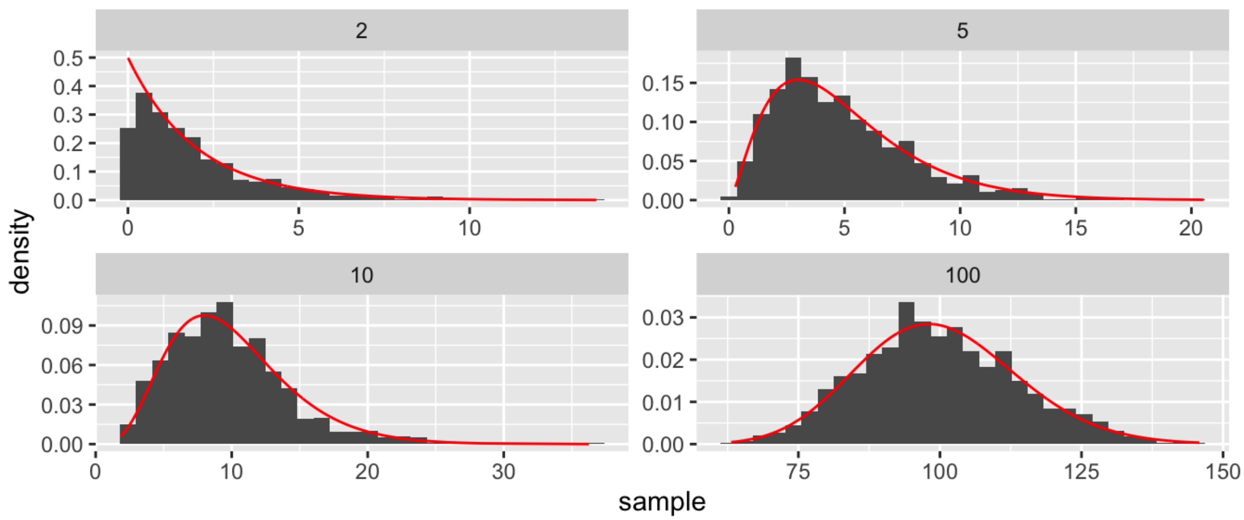
Let $X_1, \dots, X_p$ be a set of independent random variables with pdfs $f_{X_1}, \dots, f_{X_p}$, respectively, and let $g(X_1, \dots, X_p)$ be some transformation we are interested in simulating from.

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

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

```r
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")
```



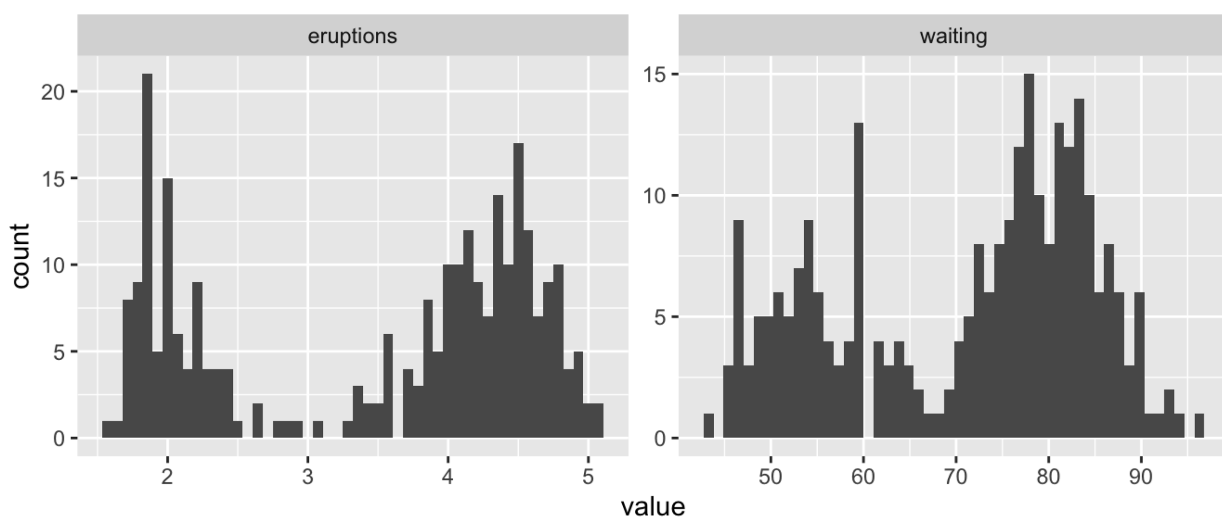
4 Mixture Distributions

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")
```



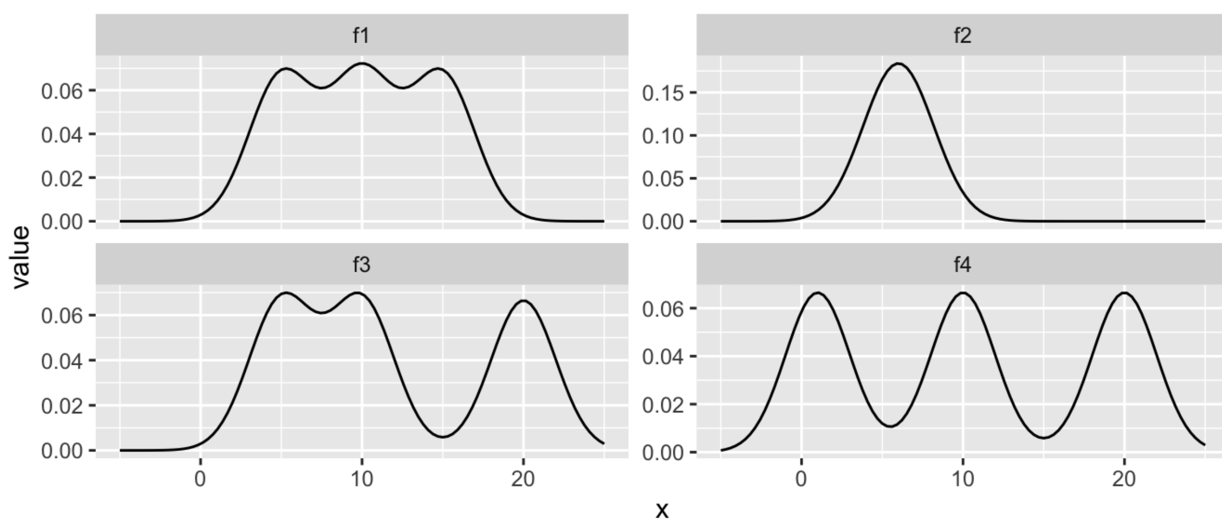
What is the shape of these distributions?

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

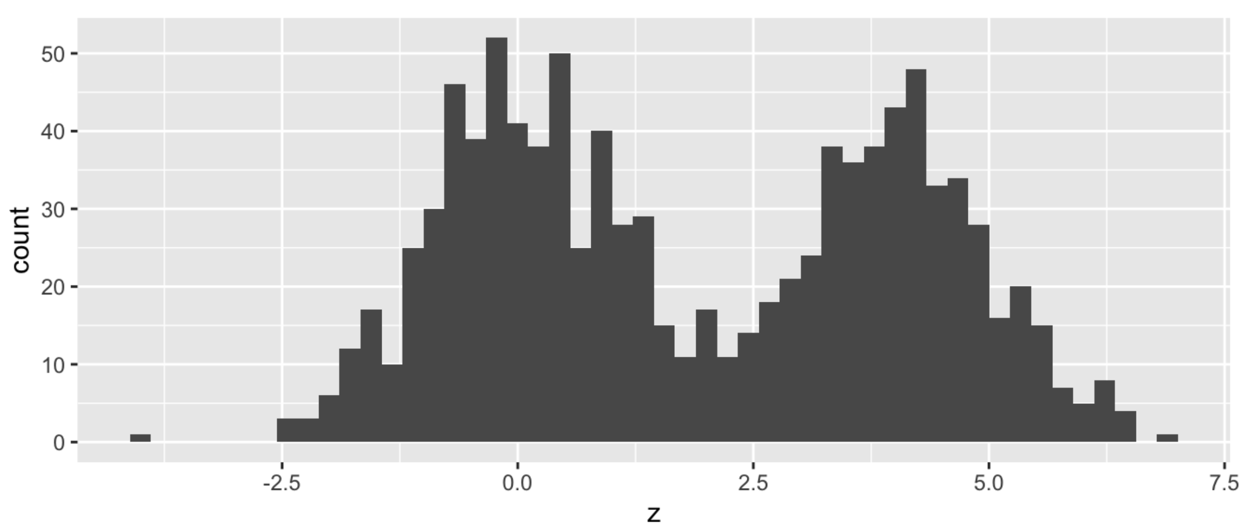
Note that mixture distributions are *not* the same as the distribution of a sum of r.v.s.

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.

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

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

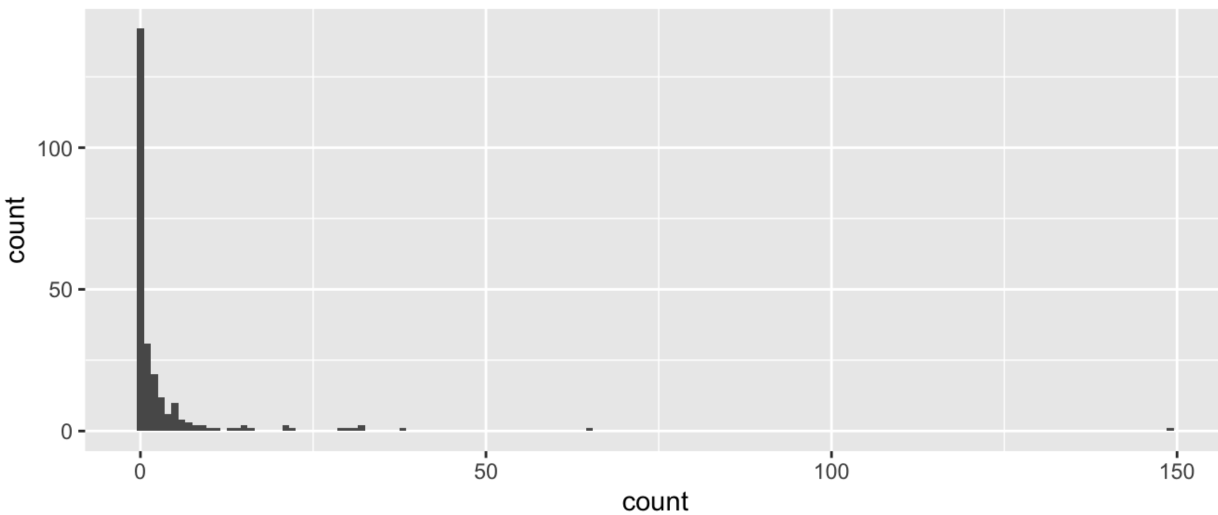
Example 4.4 The Colorado division of Parks and Wildlife has hired you to analyze their data on the number of fish caught in Horsetooth resevoir by visitors. Each visitor was asked - How long did you stay? - How many fish did you catch? - Other questions: How many people in your group, were children in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

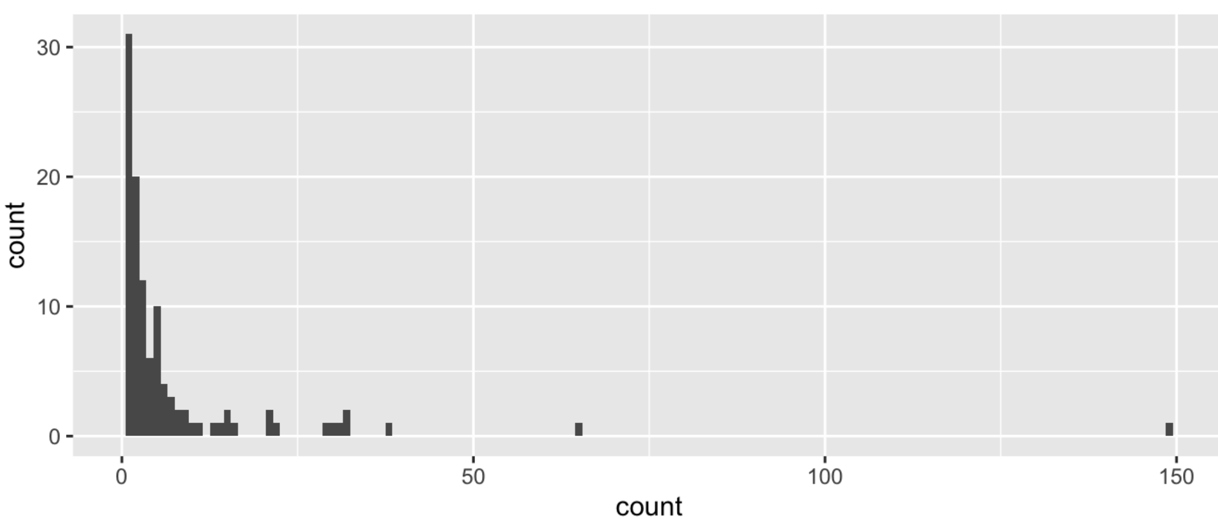
Note, this is modified from https://stats.idre.ucla.edu/r/dae/zip/.

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

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




```r
# 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 $Y \sim ZIP(\lambda)$,

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

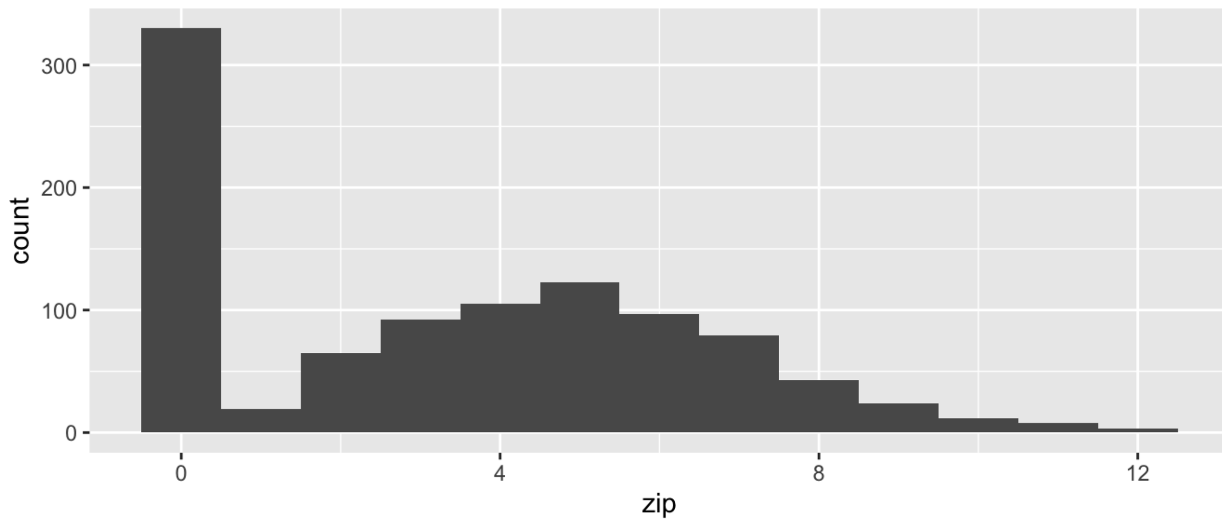
So that,

$$Y =$$

To simulate from this distribution,

```r
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)
```