# 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 (later)

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

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

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

(pseudo random).

```
##  [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.
```

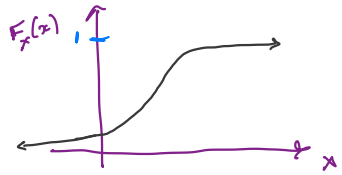
```
##  [1] 0.67720831 0.04377997 5.38745038 0.48773005 1.18690322 0.92734297
##  [7] 0.33936255 0.99803323 0.27831305 0.94257810
```

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

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

# 1 Inverse Transform Method

**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.

**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 $x = F^{-1}(u)$.

recommend think about vectorization in R.

**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 $F_X$:

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

2. Find $F_X^{-1}$:

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

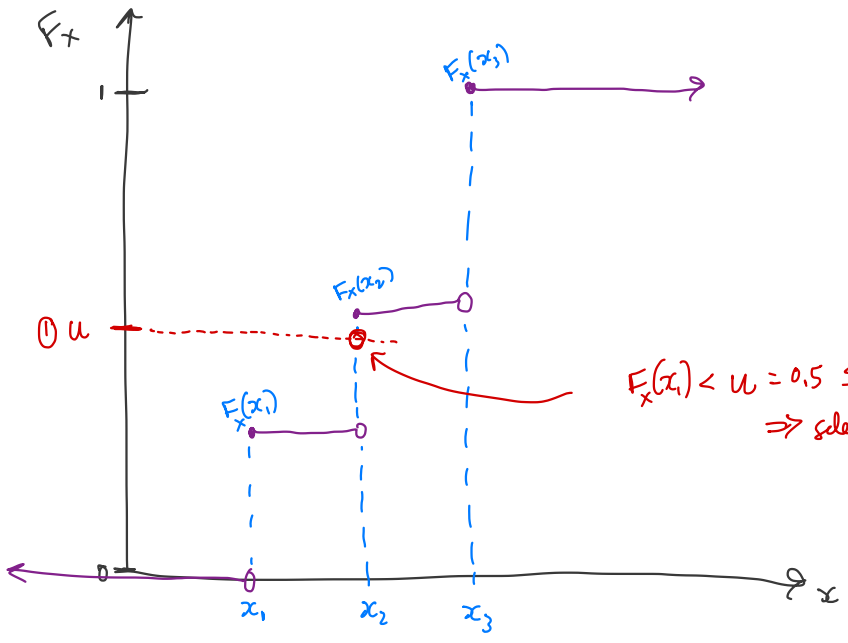
$$\Rightarrow F_X^{-1}(u) = u^{1/3} \qquad 0 \leq u \leq 1 \text{ (range of } F(x)\text{)}$$

3.
```
# write code for inverse transform example
# f_X(x) = 3x^2, 0 <= x \<= 1
```
① Write function for $F^{-1}$ in R.
② Sample values from Unif(0,1).
③ evaluate $x = F^{-1}(u)$.

## 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 in step function (support of R.V)) of $F_X(x)$, then the inverse transform is $F_X^{-1}(u) = x_i$ 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 $x_i$ where $F_X(x_{i-1}) < U \leq F_X(x_i)$.




① If $u = 0.5$ e.g.

$F_X(x_1) < u = 0.5 \leq F_X(x_2)$
$\Rightarrow$ select $x_2$.

YOUR TURN

**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 to do this using sample() function.

↓

remember to allow replacement and specify the prob vector

Something we can try if we can't find $F^{-1}$

# 2 Acceptance-Reject Method

the dist we want samples from.

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

Most cdfs cannot be inverted in closed form. => inverse transform method not possible.

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 reject the rest.

The method is outlined below:

requirements for g.

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

"proposal distribution"

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.

(3) support of g must include support of f!

support

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.

Question:
- What might be hard about this?
- What might be slow about this?

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

We can use this to evaluate efficiency of our algorithm.

## 2.1 Algorithm

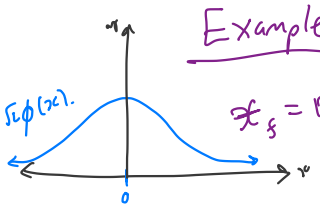
find $c$ s.t. $c \cdot g(x) \geq f(x)$.

1. Find a suitable (proposal) 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.

** Requirement: The support of g must include the support of f **.

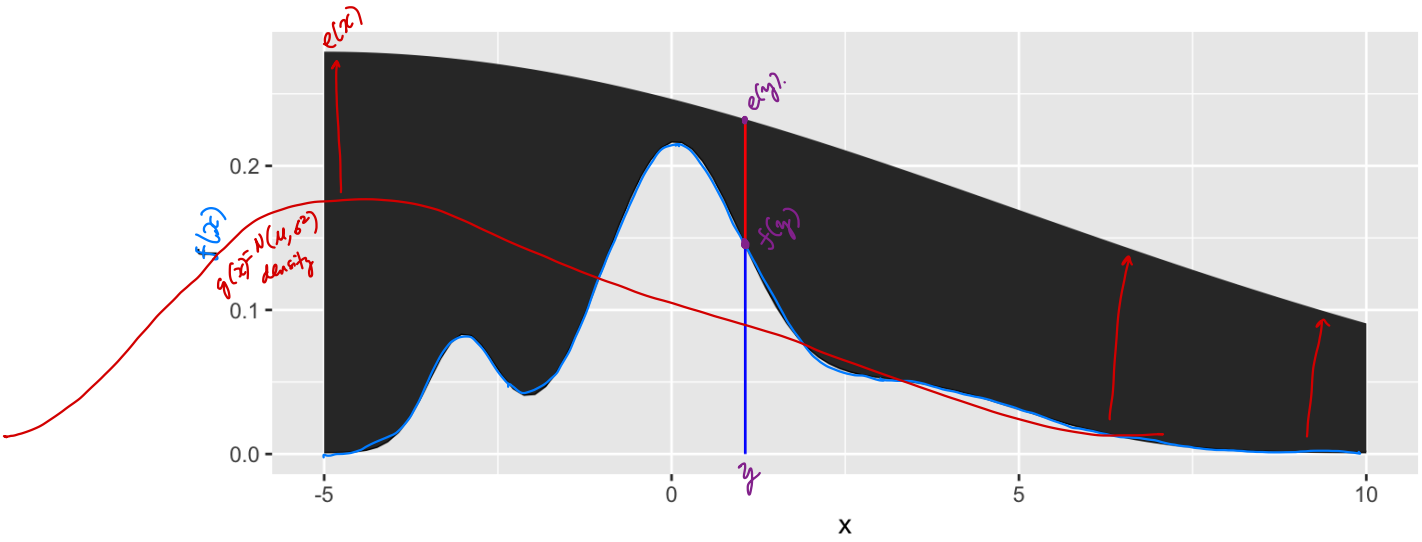
Example: If $f = N(0,2)$ and $g = \text{Unif}(-10, 10)$. BAD idea.

$\mathcal{X}_g = [-10, 10]$.




$\mathcal{X}_f = \mathbb{R}$ is it true that $[-10, 10] \geq \mathbb{R}$? No.

$\mathcal{X}_g$ $\mathcal{X}_f$




## 2.2 Envelopes

Good envelopes have the following properties:

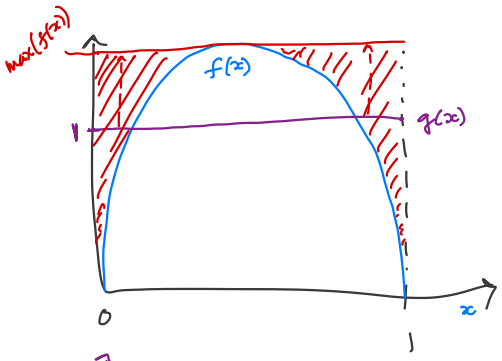
1. envelope exceeds target everywhere ← support of g MUST include the support of f.
2. Easy to sample from g.
3. Generate few rejected draws (save time).

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

Say the support of f is $0 \le x \le 1$.

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

Find $\max_{0 \le x \le 1}(f(x))$ and let $c = \max_{0 \le x \le 1}(f(x))$




* This is ONLY relevant if $\mathcal{X}_f = [0,1]$ *

plotting is your friend for other supports. to help find c.

this isn't always the most efficient way but it will always work when $\mathcal{X}_f = [0,1]$.

**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.

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

$f(x) = 60x^3(1-x)^2$

$f'(x) = 60[3x^2(1-x)^2 - 2x^3(1-x)]$

$= 60x^2(1-x)[3(1-x) - 2x]$

$= 60x^2(1-x)(3-5x) = 0$ at $x=0$, $x=1$, or $x = \frac{3}{5}$

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

$\max f(x)$ occurs at $x = \frac{3}{5} \Rightarrow c = \max_{x\in[0,1]} f(x) = f\left(\frac{3}{5}\right) = 60\left(\frac{3}{5}\right)^3\left(1-\frac{3}{5}\right)^2 = 2.0736.$

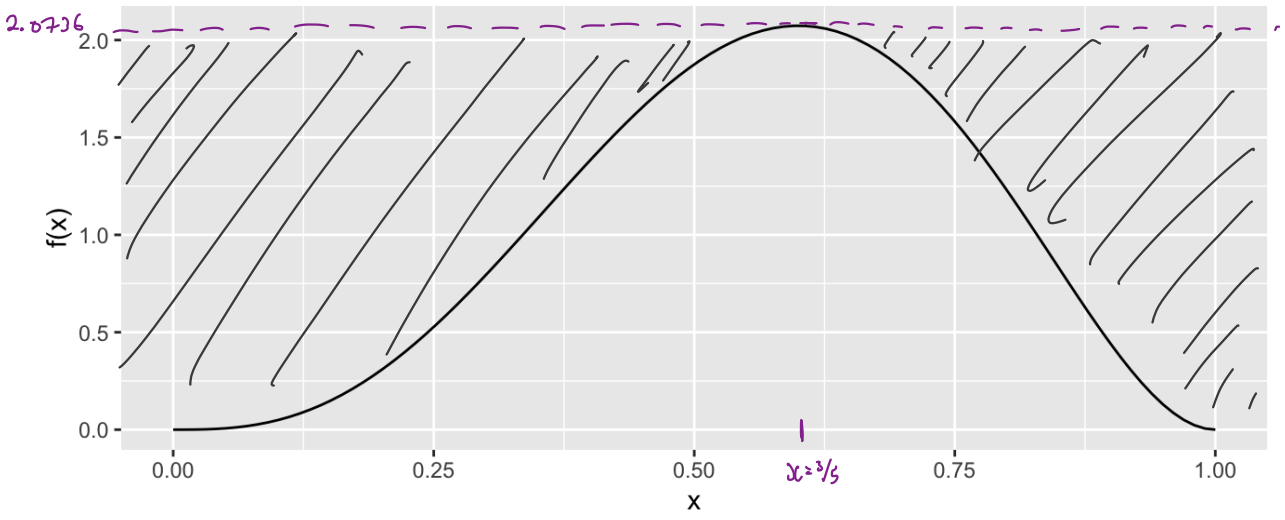
```
# pdf function, could use dbeta() instead
f <- function(x) {
    60*x^3*(1-x)^2
}

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

in base R.

make sequence of x values.

draw black line

evaluate f at each x.

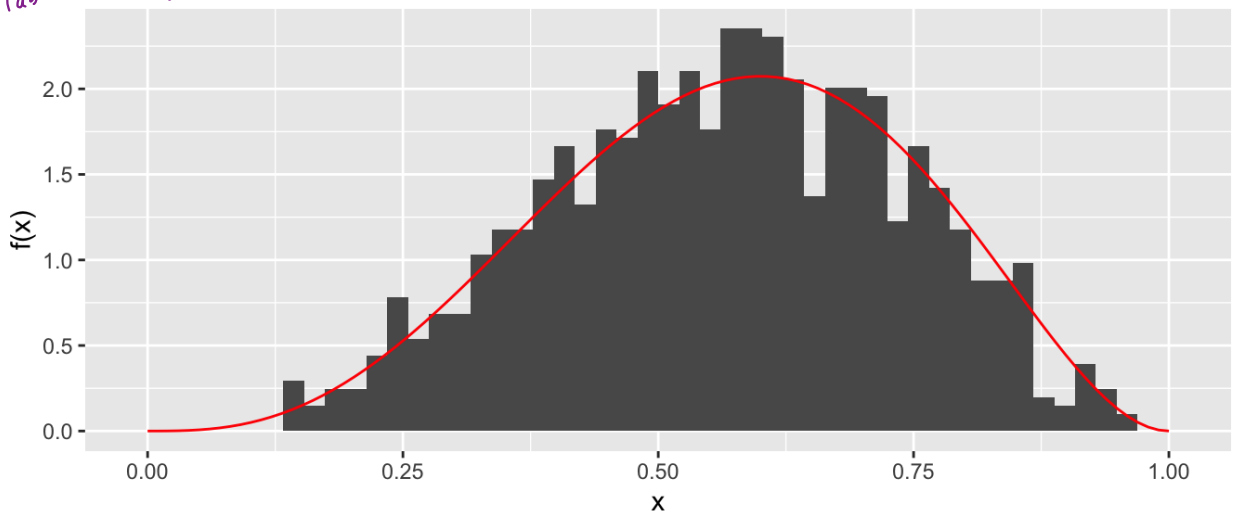


```
envelope <- function(x) {
  ## create the envelope function
}

# 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
  }
}
```

c · unif(0,1) pdf
c = f(3/5).
while we don't have enough samples accepted, keep running the loop.
y ← runif(1).
always from unif(0,1).
sample from proposal.
increment accepted counter so loop eventually ends.
accept y as a sample from f ⇒ store it

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

necessary so that histogram is on the scale of density instead of raw counts.
important for your homework.
theoretical pdf.
labels for x, y axes.



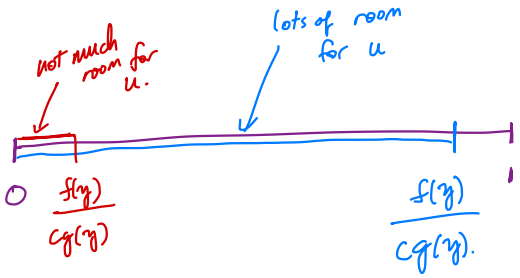
## 2.3 Why does this work?

Recall that we require

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

Thus,

$$0 \leq \frac{f(y)}{c\,g(y)} \leq 1$$




The larger the ratio $\frac{f(y)}{cg(y)}$, the more the random variable $Y \sim g$ 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 $V = Z^2 \sim \chi^2_1$
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 $X = \frac{U}{U+V} \sim \text{Beta}(r,s)$.

**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 $\sum_{i=1}^n X_i$?

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

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

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

**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? $X_{[i]}$?

This is more complex...

but possible.

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

- change of variable ("continuous")

If $g$ is monotone, then for cts $X$ and $Y = g(X)$,

$$f_Y(y) = \begin{cases} f_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right| & y \in \mathcal{Y} \\ 0 & \text{o.w.} \end{cases}$$

- Moment generating function

$M_X(t) = E(e^{tX})$.

- Convolution theorem

$Z = X + Y$.

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

Use computational statistical methods. we can simulate from transformed distribution.

## 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 $X_1 \sim f_{X_1}, \ldots, X_p \sim f_{X_p}$. ← either straightforward (named) or inverse cdf, accept-reject.
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 $Z$?

① Simulate $X_1, \ldots, X_p \sim N(0,1)$.
② Compute $\sum X_i^2$
③ repeat 0-2.

```
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),
```

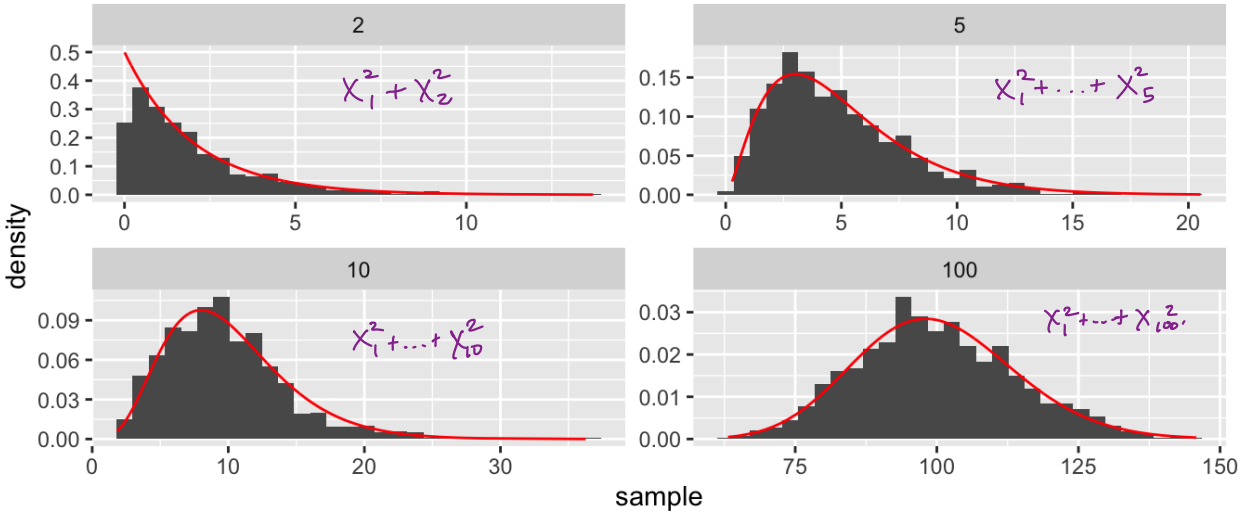
# of r.v.'s
df $\chi^2_p$.

This is n samples of p N(0,1) indep. r.v.'s.

df

```
                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 in red.
  facet_wrap(~df, scales = "free")
```

density y scale (not count scale).



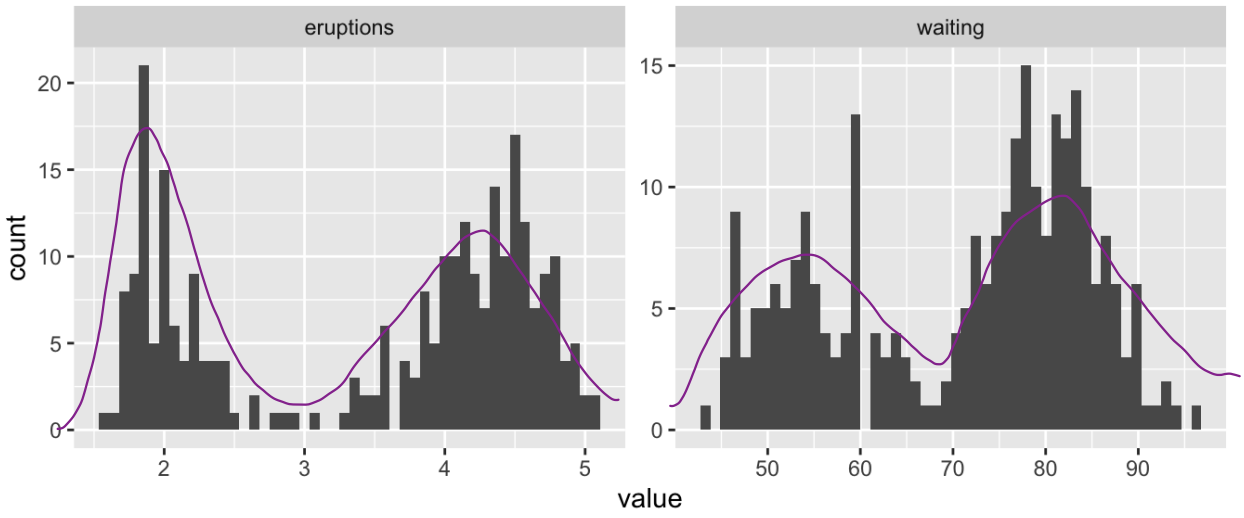
# 4 Mixture Distributions

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?

Bimodal i.e. two modes.

**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 $\sum \theta_i = 1$.

of distributions.

For 2 r.v.s,

$$f_y(y) = \theta f_{X_1}(y) + (1-\theta) f_{X_2}(x)$$

two different distributions!

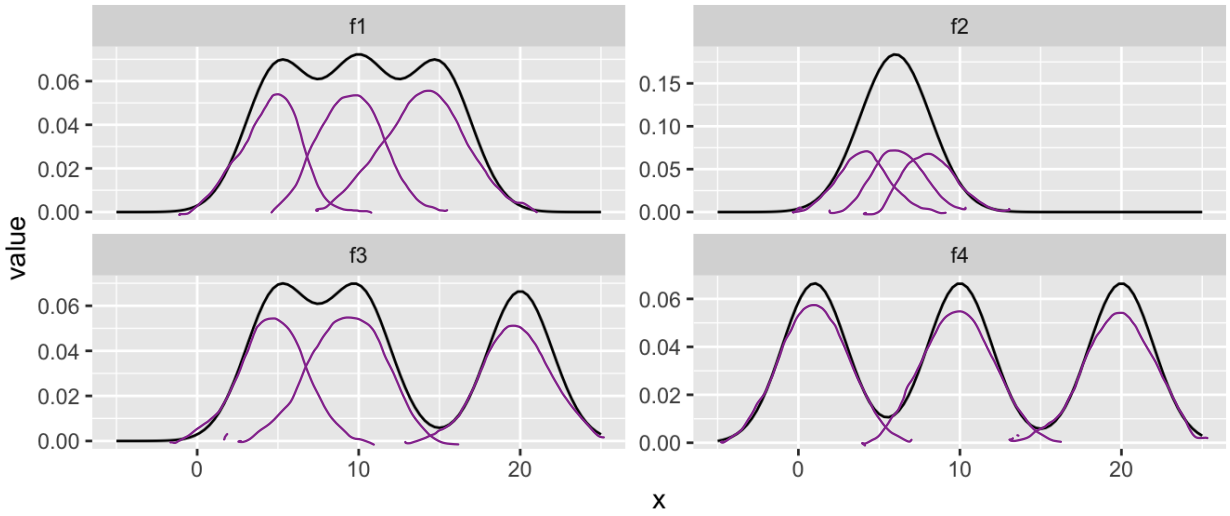
How do we simulate from this distribution?

There are 2 sources of variability:

$$Z \sim \text{Bernoulli}(\theta) \longrightarrow \text{if } \begin{cases} Z=1 & Y \sim f_{X_1} \\ Z=0 & Y \sim f_{X_2} \end{cases}$$

**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.

Mixtures are weighted sums of distributions

NOT distributions of weighted sums!

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

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

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

$= \frac{1}{2}EX_1 + \frac{1}{2}EX_2 = \frac{1}{2}(0+4) = 2.$

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

$\overset{\text{indep}}{=} \frac{1}{4}Var\,X_1 + \frac{1}{4}Var\,X_2 = \frac{1}{4}(1+1) = \frac{1}{2}$

Can in fact show $S = \frac{1}{2}(X_1+X_2) \sim N(2, \frac{1}{2})$ this is a unimodal dsn.

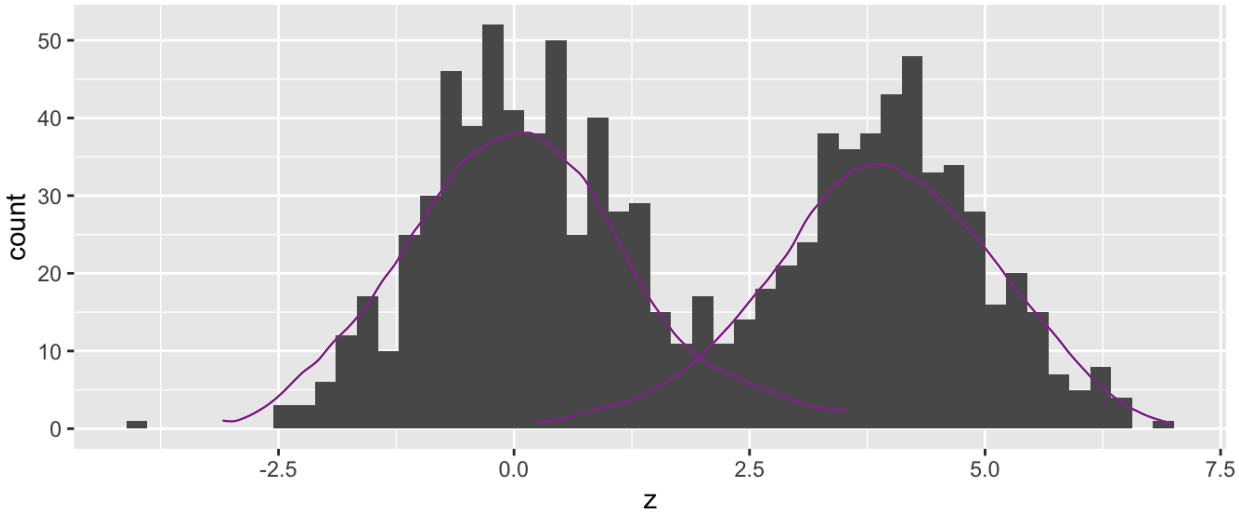
$Z$ such that $f_Z(z) = 0.5f_{X_1}(z) + 0.5f_{X_2}(z)$.

is a mixture of the two.

```
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)
```

choose which dsn

N(0,1) N(4,1).




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

change `u <- rbinom(n, 1, 0.7)` to choose $f_{X_1}$ w.p. 0.7.

# 4.2 Models for Count Data (refresher)

Recall that the Poisson($\lambda$) 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 $\lambda$ 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,

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

↑ restrict the shape of the dsn.

**Example 4.3**

- # homes sold per day by a real estate company.
- # of calls per minute at a hotel reservation call center.
- # of meows in a 2 minute cat video on youtube.

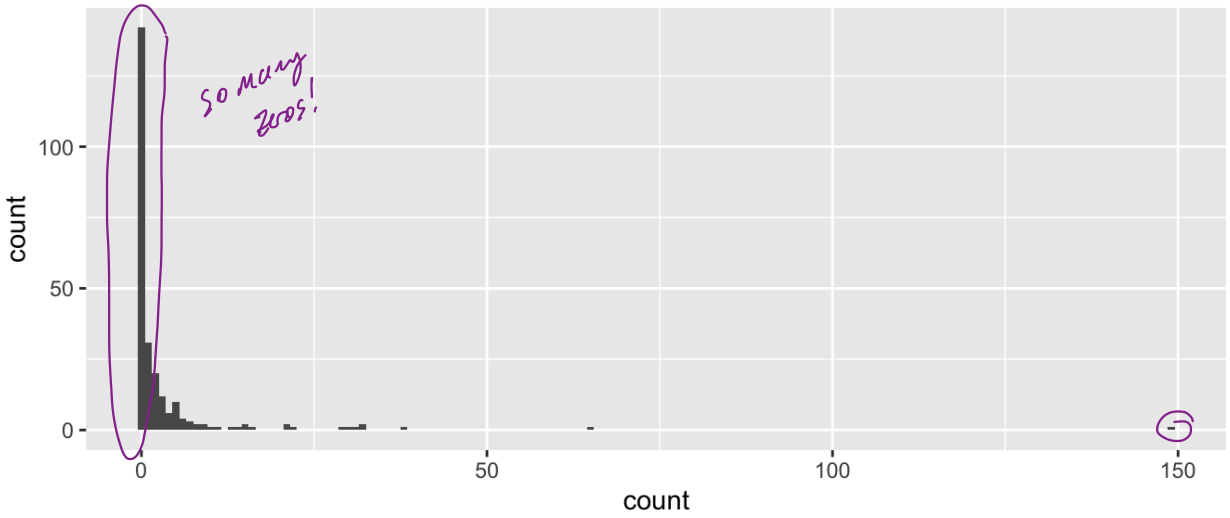
**Example 4.4** The Colorado division of Parks and Wildlife has hired you to analyze their data on the number of fish caught in Horsetooth resevoir by visitors. Each visitor was asked - How long did you stay? - How many fish did you catch? - Other questions: How many people in your group, were children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

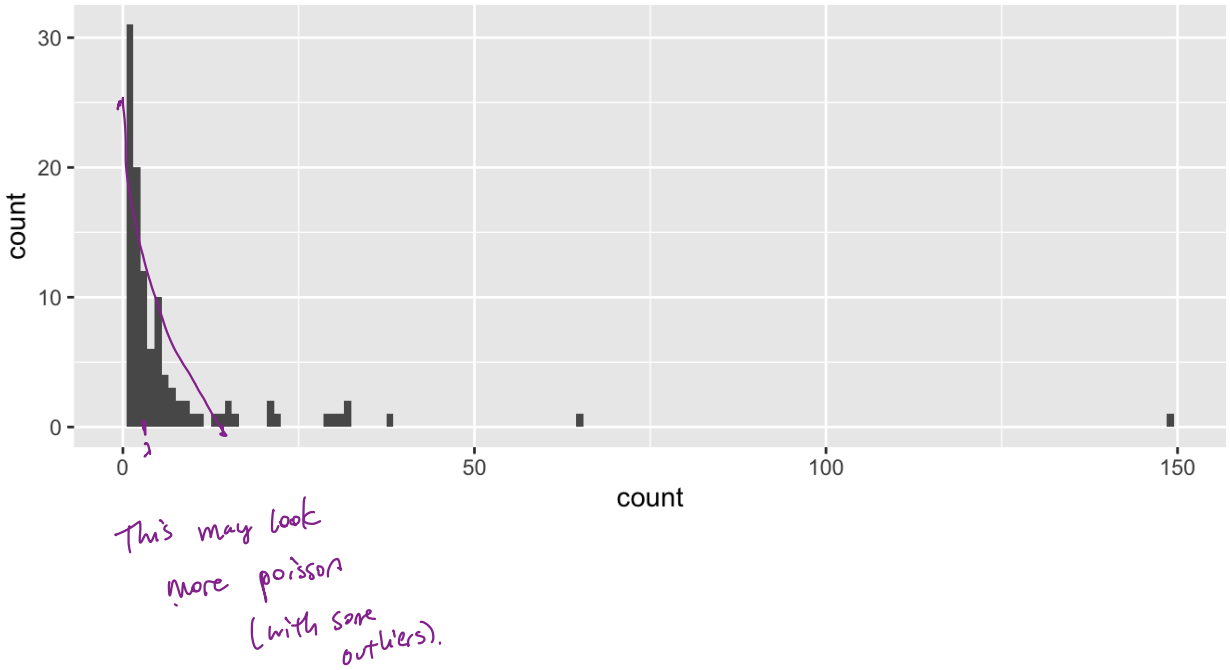
Note, this is modified from https://stats.idre.ucla.edu/r/dae/zip/.

```
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. → a zero is possible and occurs by chance.
↘ a non-zero is impossible.

**Example 4.5**

Outcome of a study = # cows with foot and mouth disease (FMD) per region in Turkey.

↳ structural zeros – there are no cows in the region

↳ sampling zeros – cows in the region, but no FMD.

Key point: you don't know if region has no cows or no disease.

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.
↑ structural zeroes.

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,

$$Y = \begin{cases} 0 & \text{w.p. } \pi + (1-\pi)\exp(-\lambda) \\ k & \text{w.p. } (1-\pi)\frac{\lambda^k \exp(-\lambda)}{k!} \quad k = 1, 2, \ldots \end{cases}$$

To simulate from this distribution,

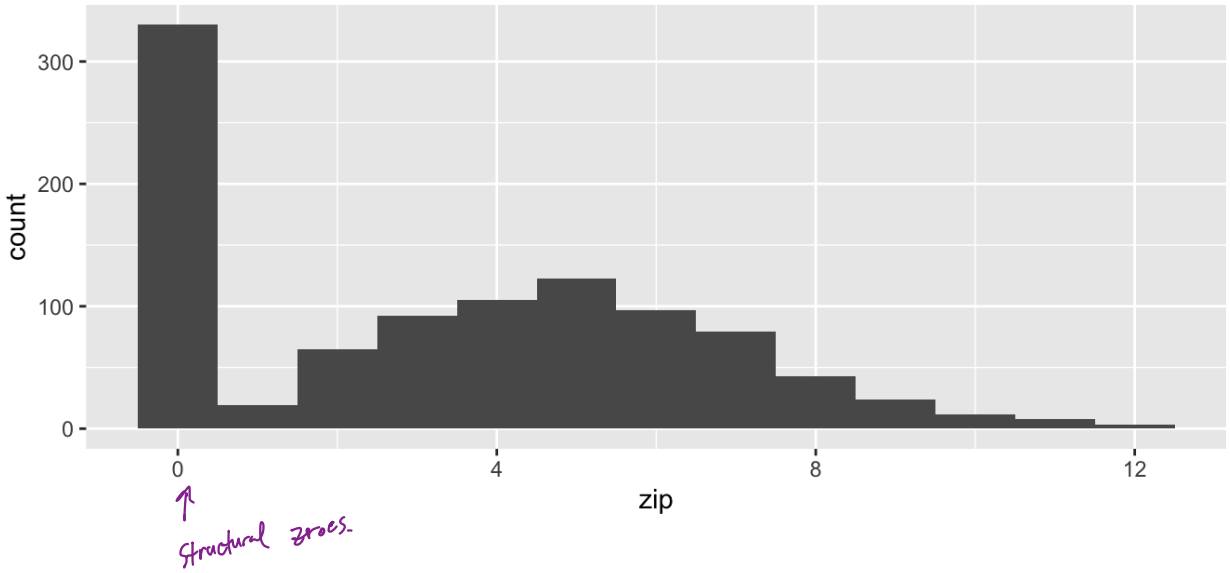
$Z \sim \text{Bern}(\pi)$

if $Z=1$, $Y=0$

$Z=0$, $Y \sim \text{Poisson}(\lambda)$.

```
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)
```