MAST90104: Introduction to Statistical Learning

Week 11 Lab and Workshop

1 Lab

1. **Proportional odds in ordinal regression.** From last week: Suppose that Y_i takes values in the ordered set $\{1, \ldots, J\}$. Using a logit link, our model for $\gamma_{ij} = \mathbb{P}(Y_i \leq j)$ is

$$\gamma_{ij} = \texttt{logit}^{-1}(\theta_j - \mathbf{x}_i^T \boldsymbol{\beta}).$$

Thinking of γ_{ij} as a function of $\mathbf{x}_i$, we can rewrite it as $\gamma_j(\mathbf{x}_i) = \mathbb{P}(Y \leq j | \mathbf{x}_i)$.

Recall the odds for an event A are given by $\mathbb{P}(A)/(1 - \mathbb{P}(A))$. By relative odds we mean the ratio of two odds. In question **??** below, you will show that the relative odds for $\{Y \leq j | \mathbf{x}_A\}$ and $\{Y \leq j | \mathbf{x}_B\}$ do not depend on j. For this reason, this model is often called the proportional odds model.

New part: This independence of the odds ratio on j can be used to check the suitability of the model. Fit a proportional odds model to the `nes96` data, as in lectures. For $j = 1$ and $j = 2$, calculate the difference between the *observed* log odds at income level 1.5 and levels 4, 6, 8, 9.5, ... (the values in the vector `inca`). Do they look roughly the same?

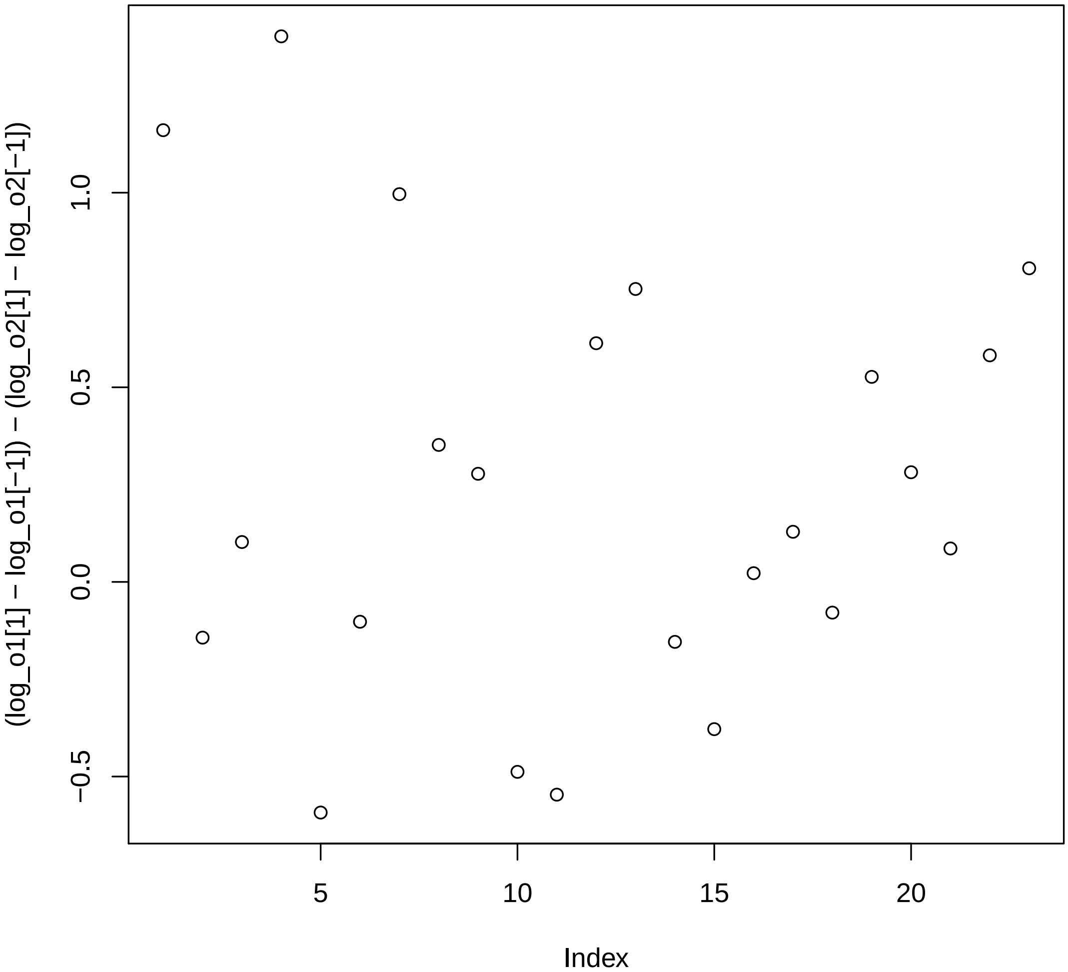
Solution:

First calculate `log_o1`, the observed log odds of being Democrat, and `log_o2`, the observed log odds of being Democrat or Independent. This is done for each income level.

```
library(faraway)
data(nes96)
sPID <- nes96$PID
levels(sPID) <- c("Democrat","Democrat","Independent","Independent",
                  "Independent","Republican","Republican")
inca <- c(1.5,4,6,8,9.5,10.5,11.5,12.5,13.5,14.5,16,18.5,21,23.5,
          27.5,32.5,37.5,42.5,47.5,55,67.5,82.5,97.5,115)
nincome <- inca[unclass(nes96$income)]
obs <- prop.table(table(nincome, sPID), 1)
p1 <- obs[,1]
p2 <- obs[,1] + obs[,2]
log_o1 <- log(p1/(1-p1))
log_o2 <- log(p2/(1-p2))
```

For the log odds of being Democrat, the difference between income level 1.5 and the rest is just given by `log_o1[1] - log_o1[-1]`. This should look the same as `log_o2[1] - log_o2[-1]`, which we check by taking differences. These differences are not particularly close to zero, but at least they don't display any trend.

```
plot((log_o1[1] - log_o1[-1]) - (log_o2[1] - log_o2[-1]))
```



2. The following program performs a simulation experiment to estimate $\mathbb{E}X$ where the function `sim.X()` simulates a random value of X .

```
# seed position 1
# set.seed(7)
mu <- rep(0, 6)
for (i in 1:6) {
# seed position 2
# set.seed(7)
X <- rep(0, 1000)
for (j in 1:1000) {
# seed position 3
# set.seed(7)
X[j] <- sim.X
}
mu[i] <- mean(X)
}
spread <- max(mu) - min(mu)
mu.estimate <- mean(mu)
```

(a) What is the value of `spread` used for?

Solution: To measure the variability of the estimates `mi[i]`.

(b) If we uncomment the command `set.seed(7)` at seed position 3, then what is `spread`?
Solution: 0

(c) If we uncomment the command `set.seed(7)` at seed position 2 (only), then what is `spread`?
Solution: 0

(d) If we uncomment the command `set.seed(7)` at seed position 1 (only), then what is `spread`?
Solution: > 0

(e) At which position should we set the seed?
Solution: 1

3. For $X \sim \text{Poisson}(\lambda)$ let $F(x) = \mathbb{P}(X \le x)$ and $p(x) = \mathbb{P}(X = x)$. Show that the probability function satisfies

$$p(x+1) = \frac{\lambda}{x+1} p(x).$$

Using this write a function to calculate $p(0), p(1), \ldots, p(x)$ and $F(x) = p(0) + p(1) + \cdots + p(x)$.

If X is a random variable with non-negative integer values and `F(x)` is a function in R that returns the cdf F of X, then as discussed in lectures X can be simulated using the following code:

```
F.rand <- function () {
u <- runif(1)
x <- 0
while (F(x) < u) {
x <- x + 1
}
return(x)
}
```

In the case of the Poisson distribution, this code can be made more efficient by calculating F jrecursively. By using two new variables, `p.x` and `F.x` for $p(x)$ and $F(x)$ respectively, modify this program so that instead of using the function `F(x)` it updates `p.x` and `F.x` within the `while` loop. Your code should have the form where you fill in the question marks:

```
F.rand <- function(lambda) {
u <- runif(1)
x <- 0
p.x <- ?
F.x <- ?
while (F.x < u) {
x <- x + 1
p.x <- ?
F.x <- ?
}
return(x)
}
```

Your code needs to ensure that at the start of the `while` loop you always have `p.x` equal to $p(x)$ and `F.x` equal to $F(x)$.

Check that your simulation works by choosing a parameter, generating a large number of random variables, using them to estimate the probability mass function, and comparing your estimates to the true values (which you can get using `dpois`).

Solution: We have

$$p(x+1) = \frac{e^{-\lambda}\lambda^{x+1}}{(x+1)!} = \frac{e^{-\lambda}\lambda^{x}}{x!}\frac{\lambda}{x+1} = p(x)\frac{\lambda}{x+1}$$

Thus we can calculate $F(x)$ as follows

```r
Fpois <- function(x, la) {
# poisson(la) cdf at x (assumed integer)
y <- 0
py <- exp(-la)
Fy <- py
while (y < x) {
y <- y + 1
py <- la*py/y
Fy <- Fy + py
}
return(Fy)
}
sapply(0:4, Fpois, la=2) # check it works

## [1] 0.1353353 0.4060058 0.6766764 0.8571235 0.9473470

ppois(0:4, 2) # using R's built in function

## [1] 0.1353353 0.4060058 0.6766764 0.8571235 0.9473470
```

To simulate from this cdf we have the following

```r
F.rand <- function(lambda) {
u <- runif(1)
x <- 0
p.x <- exp(-lambda)
F.x <- p.x
while (F.x < u) {
x <- x + 1
p.x <- lambda*p.x/x
F.x <- F.x + p.x
}
return(x)
}

n <- 100000
lambda <- 2
F.sample <- rep(0, n)
for (i in 1:n) F.sample[i] <- F.rand(lambda)
table(F.sample)/n

## F.sample
##       0       1       2       3       4       5       6       7       8
## 0.13561 0.27088 0.26812 0.18138 0.09117 0.03548 0.01263 0.00370 0.00083
##       9      10      11
## 0.00016 0.00002 0.00002

round(dpois(0:9, 2), 5)

##  [1] 0.13534 0.27067 0.27067 0.18045 0.09022 0.03609 0.01203 0.00344
##  [9] 0.00086 0.00019
```

4. (a) Here is some code for simulating a discrete random variable Y. What is the probability mass function (pmf) of Y?

```
Y.sim <- function() {
U <- runif(1)
Y <- 1
while (U > 1 - 1/(1+Y)) {
Y <- Y + 1
}
return(Y)
}
```

Let N be the number of times you go around the while loop when `Y.sim()` is called. What is $\mathbb{E}N$ and thus what is the expected time taken for this function to run?

(b) Here is some code for simulating a discrete random variable Z. Show that Z has the same pmf as Y

```
Z.sim <- function() {
Z <- ceiling(1/runif(1)) - 1
return(Z)
}
```

Will this function be faster or slower that `Y.sim()`?

Solution: We have, for $y \geq 1$,

$$\mathbb{P}(Y = y) = 1 - \frac{1}{1+y} - \left(1 - \frac{1}{y}\right) = \frac{1}{y} - \frac{1}{1+y} = \frac{1}{y(1+y)}.$$

Thus

$$\begin{aligned} \mathbb{E}N &= \sum_{n=1}^{\infty}(n-1)\mathbb{P}(Y = n) \\ &= \sum_{n=1}^{\infty} \frac{n-1}{n(1+n)} \\ &= \left(\sum_{n=1}^{\infty} \frac{1}{1+n}\right) - 1 \\ &= \infty. \end{aligned}$$

So the expected amount of time required for this algorithm to complete is infinite!

Put $Z = \lceil 1/U \rceil - 1$ where $U \sim U(0, 1)$, then

$$Z = z \iff z < 1/U \leq z + 1 \iff 1/(1 + z) \leq U < 1/z.$$

Thus $\mathbb{P}(Z = z) = 1/z - 1/(1 + z)$ as required. Clearly `Z.sim` will be much faster than `Y.sim` on average.

5. Consider the continuous random variable X with pdf given by:

$$f_X(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} \qquad -\infty < x < \infty.$$

X is said to have a standard logistic distribution. Find the cdf for this random variable. Show how to simulate a rv with this cdf using the inversion method.

Solution: X has cdf

$$\begin{aligned} F(x) &= \int_{-\infty}^{x} f_X(y)dy \\ &= \left.\frac{1}{1+e^{-y}}\right|_{-\infty}^{x} \\ &= \frac{1}{1+e^{-x}} \end{aligned}$$

Thus $F^{-1}(y) = \log(y/(1-y)) = -\log((1/y) - 1)$ and we can simulate a sample of size n from the distribution of X as follows

```
X.sim <- function(n=1) -log(1/runif(n) - 1)
```

6. The double exponential or Laplace distribution has the following density, for some $\lambda > 0$,

$$f(x) = \frac{\lambda}{2} e^{-\lambda|x|}, \text{ for } -\infty < x < \infty.$$

Plot the density and the cumulative distribution function, F.

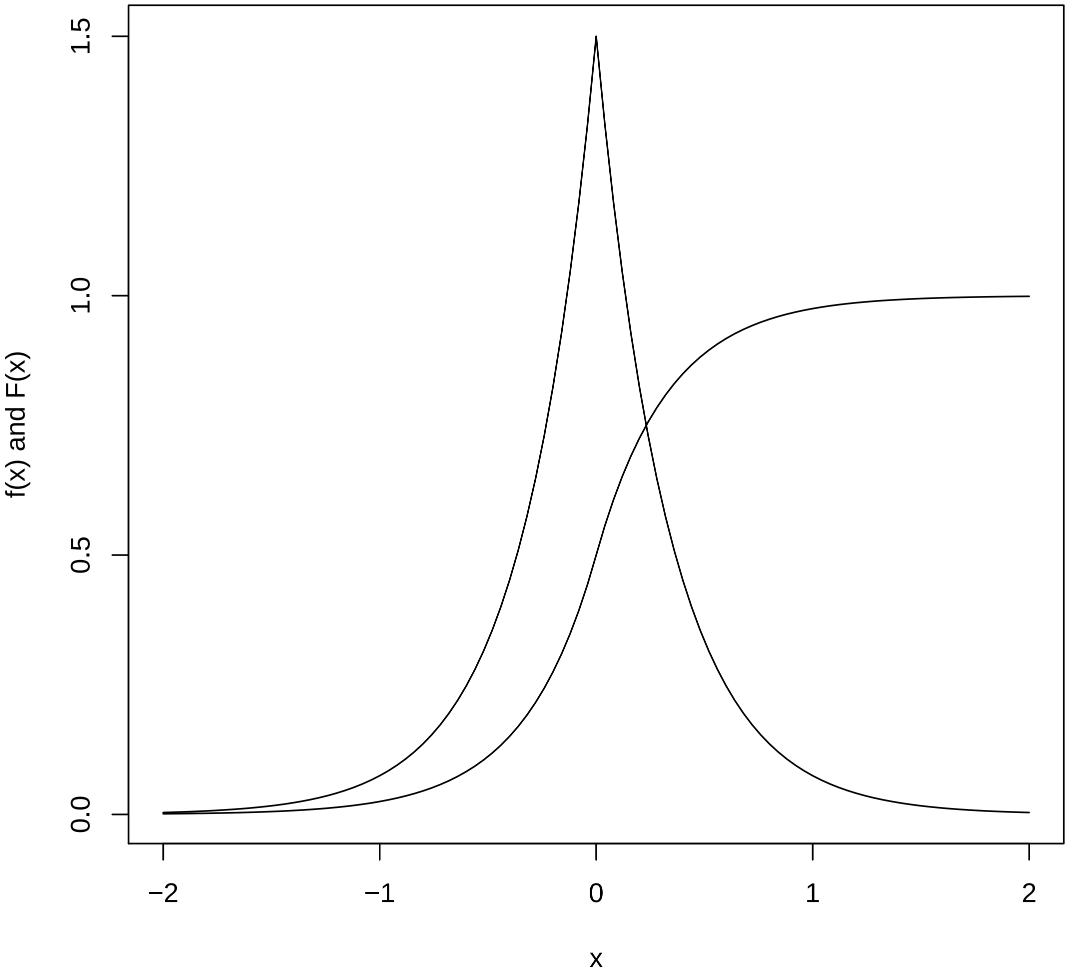
Suppose that A has an exponential distribution with rate λ, and that B is independent of A and takes on values $+1$ and -1 with equal probability. Show that AB has a Laplace distribution with parameter λ, then write a function to simulate Laplace rv's.

Solution: Integrating we get the cdf

$$F(x) = \begin{cases} \frac{1}{2}e^{\lambda x} & x < 0 \\ 1 - \frac{1}{2}e^{-\lambda x} & x \geq 0 \end{cases}$$

We plot the density and curvature for $\lambda = 3$.

```
fx <- function(x, la) la*exp(-la*abs(x))/2
Fx <- function(x, la) ifelse(x < 0, exp(la*x)/2, 1 - exp(-la*x)/2)
la <- 3
curve(fx(x, la), -2, 2, ylab="f(x) and F(x)")
curve(Fx(x, la), -2, 2, add=TRUE)
```



Now

$$\begin{aligned}\mathbb{P}(AB \in (x, x+dx)) &= \tfrac{1}{2}\mathbb{P}(A \in (x, x+dx)|B=1) + \tfrac{1}{2}\mathbb{P}(A \in (-x-dx, -x)|B=-1)\\ &= \tfrac{1}{2}\lambda e^{-\lambda x} dx\, I_{x\geq 0} + 0\, I_{x<0} + 0\, I_{x\geq 0} + \tfrac{1}{2}\lambda e^{-\lambda x} dx\, I_{x<0}\\ &= \tfrac{1}{2}\lambda e^{-\lambda |x|}\end{aligned}$$

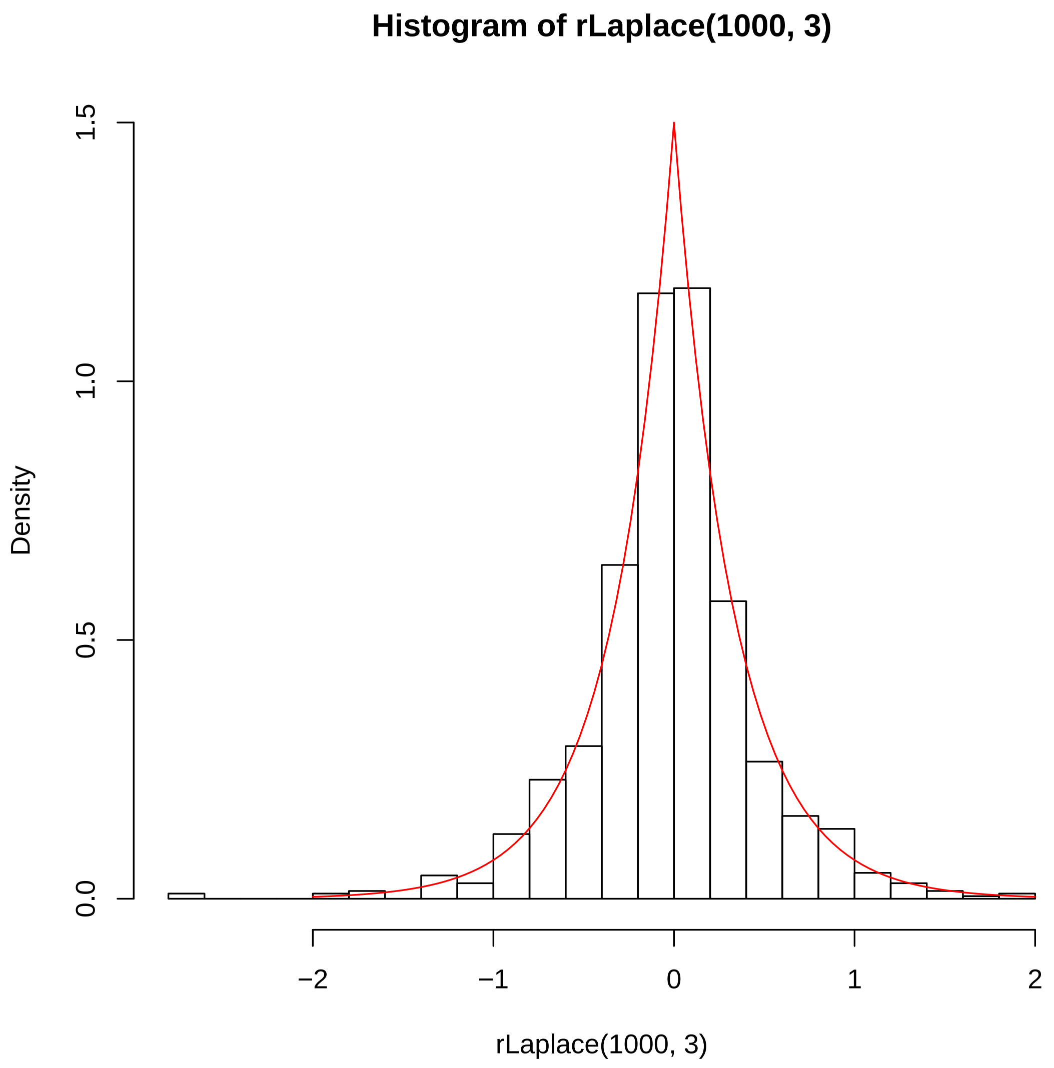
Thus, rather than apply the inverse method to F, we can simulate an $\exp(\lambda)$ then toss a coin and with probability $1/2$ multiply by -1. For example

```
rLaplace <- function(la) {
A <- -log(runif(1))/la
B <- sign(runif(1) - 0.5)
return(A*B)
}
```

Or, in vectorised form,

```
rLaplace <- function(n, la) {
A <- -log(runif(n))/la
B <- sign(runif(n) - 0.5)
```

```
return(A*B)
}
hist(rLaplace(1000, 3), breaks=20, freq=FALSE, ylim=c(0,1.5))
curve(fx(x,3), -2, 2, add=TRUE, col="red")
```



7. The Cauchy distribution with parameter α has pdf

$$f_X(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)} \qquad -\infty < x < \infty.$$

Write a program to simulate from the Cauchy distribution using the inversion method.

Now consider using a Cauchy envelope to generate a standard normal random variable using the rejection method. Find the values for α and the scaling constant k that minimise the probability of rejection. Write an R program to implement the algorithm.

Note: this is not a very efficient way of generating normals.

Solution: The Cauchy distribution has cdf

$$F(x) = \int_{-\infty}^{x} f_X(x)dx = \tfrac{1}{2} + \frac{1}{\pi}\arctan(x/\alpha).$$

Thus, for $y \in (0,1)$,

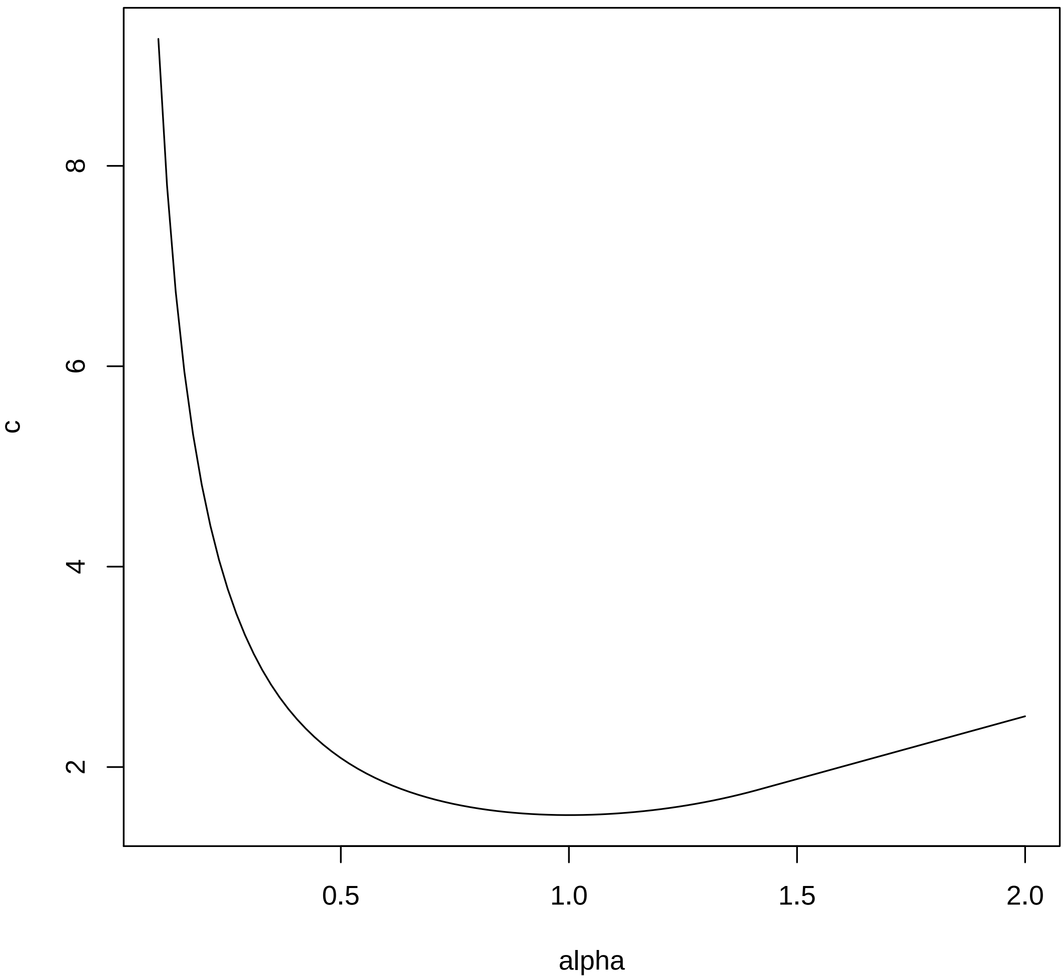
$$F^{-1}(y) = \alpha\tan(\pi(y - \tfrac{1}{2})).$$

The tails of the Cauchy decay more slowly than the normal, and both have bounded densities, so we can use the Cauchy density to give an envelope for the standard normal. Let ϕ be the standard normal density, then we need to find c s.t. $cf_X(x) \geq \phi(x)$ for all x. The optimal choice is $c = \sup_x \phi(x)/f_X(x)$. Taking the derivative w.r.t. x and setting this to zero, we see that the supremum occurs at either 0 or $\sqrt{2-\alpha^2}$, giving

$$c = \begin{cases} \sqrt{\frac{\pi}{2}}\alpha & \alpha^2 > 2 \\ \frac{\sqrt{2\pi}}{\alpha} e^{-(1-\alpha^2/2)} & \alpha^2 \leq 2 \end{cases}$$

c is the expected number of times the algorithm loops, so we choose α to minimise c and thus maximise the efficiency.

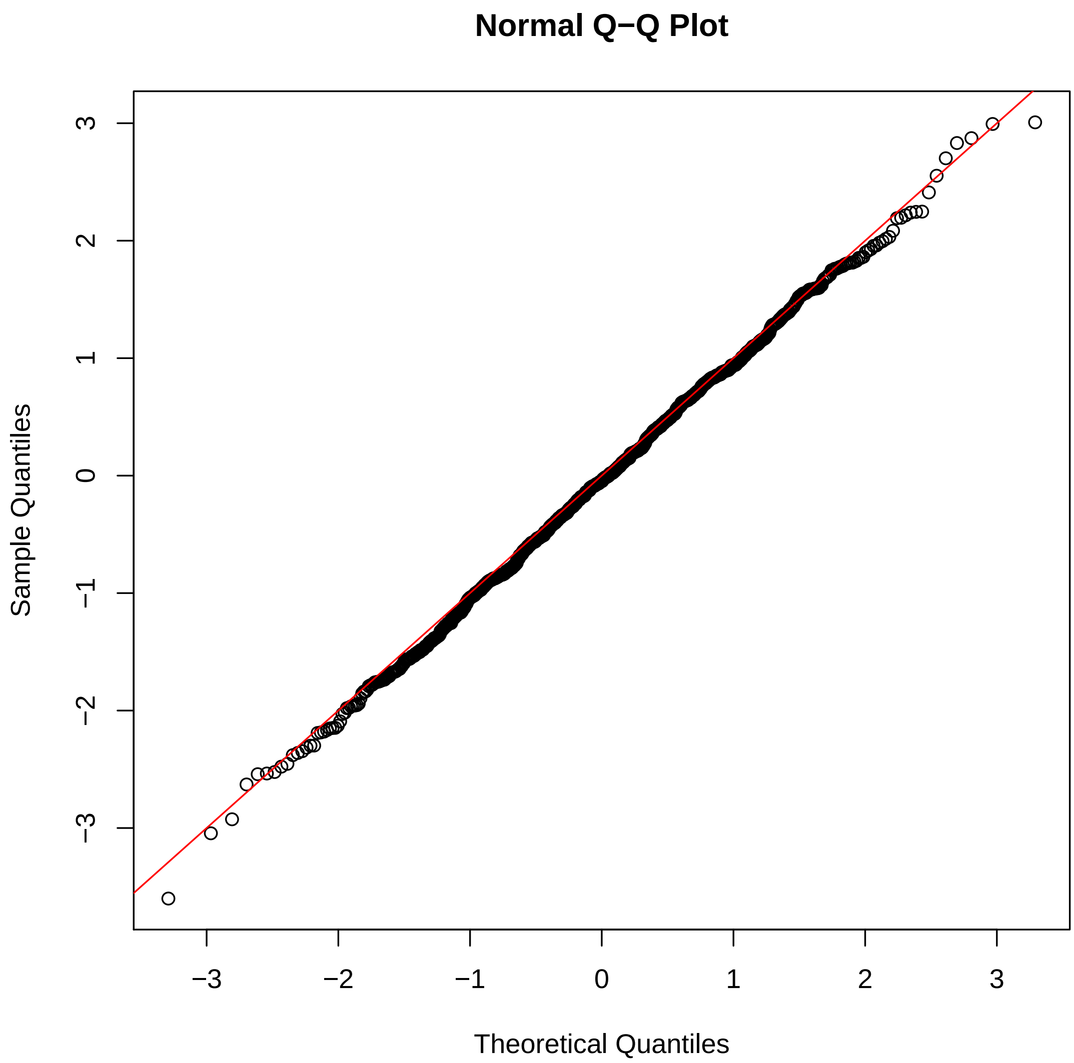
```
curve(ifelse(x^2 <= 2, sqrt(2*pi)/x*exp(x^2/2 - 1), sqrt(pi/2)*x), .1, 2,
xlab="alpha", ylab="c")
```



It is easy to check that $\min_\alpha c = \sqrt{2\pi}e^{-1/2} = 1.52035$, occuring at $\alpha = 1$. We can now implement and test the rejection algorithm.

```
cauchy_sim <- function(n, alpha=1) {
# simulate n r.v.s from the Cauchy distribution with density
# alpha/pi/(alpha^2 + x^2)
```

```
alpha*tan(pi*runif(n, -0.5, 0.5))
}

phi <- function(x) exp(-x^2/2)/sqrt(2*pi)

Z_sim <- function() {
# simulate a standard normal
c <- 1.52035
while (TRUE) {
x <- cauchy_sim(1)
y <- runif(1, 0, c/(1 + x^2)/pi)
if (y < phi(x)) return(x)
}
}

set.seed(7)
n <- 1000
x <- rep(0, n)
for (i in 1:n) x[i] <- Z_sim()
qqnorm(x)  # should see a straight line slope 1
abline(0, 1, col="red")  # adds a reference line
```



8. (a) Construct an acceptance-rejection sampling algorithm to generate a truncated exponential distribution, which has the pdf $p(z) = \frac{e^{-z}}{1-e^{-1}}$, $0 < z < 1$.

   (b) Calculate the mean and variance for the pdf $p(z)$ in (a).

   (c) Write an R program to implement the algorithm in (a) and use it to generate a sample of 1000 observations. Plot a histogram of the sample. Calculate the sample mean and variance, and compare them with the results in (b).

   (d) Show that the following algorithm also simulates from the distribution in (a).

   1° Generate U from Unif(0,1);

   2° If $U > e^{-1}$ then deliver $Z = -\ln(U)$; otherwise go to 1°.

Solution: When $0 < z < 1$, $p(z) < (1 - e^{-1})^{-1} < \infty$, so we can use a rectangular envelope to contain the density. We get the following rejection algorithm:

1° Generate $X \sim (0, 1)$.

2° Generate $Y \sim (0, 1/(1 - e^{-1}))$ independently.

3° If $Y \leq p(X)$ return X; otherwise go to 1°.

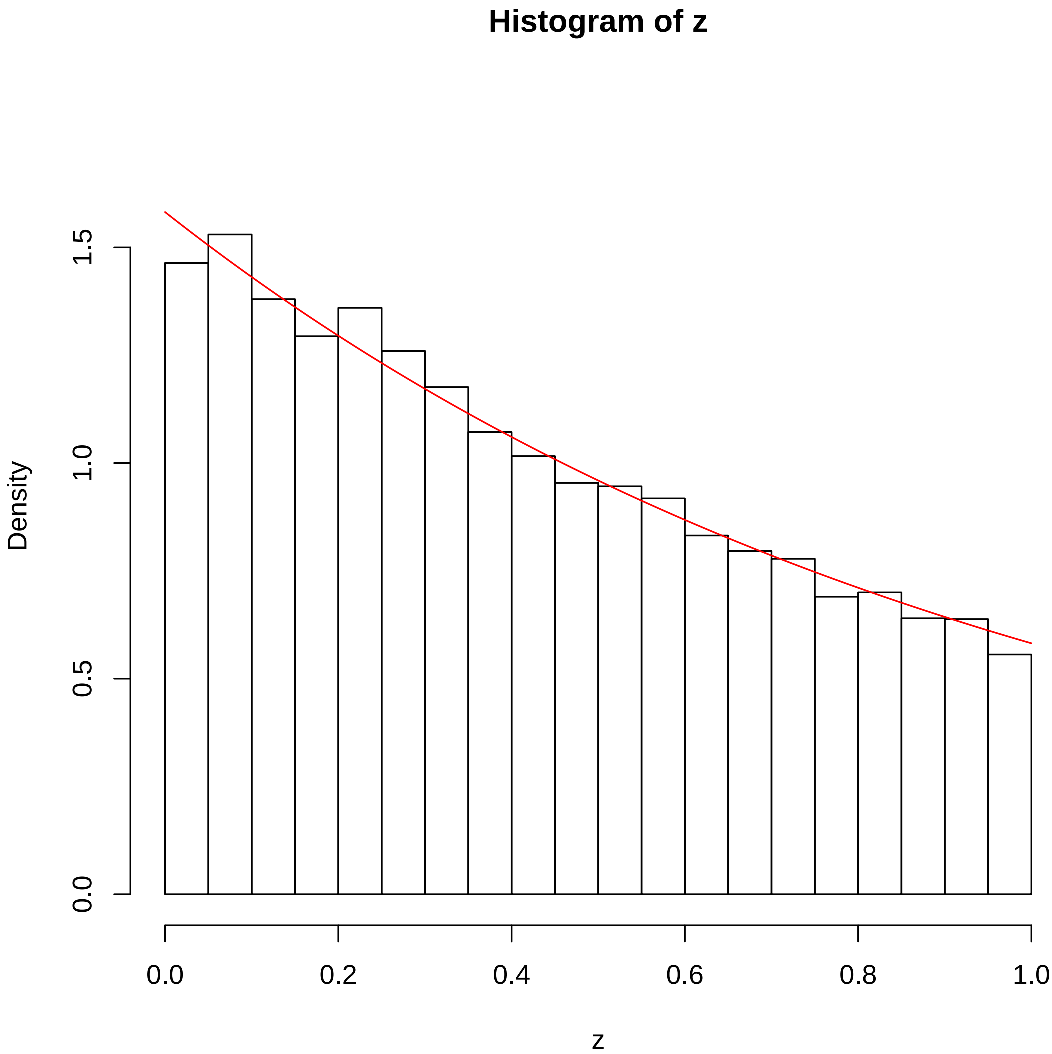
If Z has a truncated exponential distribution as above, then

$$E(Z) \quad = \quad \int_0^1 z\frac{e^{-z}}{1-e^{-z}}dz = \frac{1-2e^{-1}}{1-e^{-1}} \quad = \quad 0.418023.$$

$$
\begin{aligned}
E(Z^2) &= \int_0^1 z^2 \frac{e^{-z}}{1-e^{-z}} dz = \frac{2-5e^{-1}}{1-e^{-1}}. \\
\mathrm{Var}(Z) &= E(Z^2) - (EZ)^2 \\
&= \frac{2-5e^{-1}}{1-e^{-1}} - \left(\frac{1-2e^{-1}}{1-e^{-1}}\right)^2 \;=\; 0.079326.
\end{aligned}
$$

We check our simulation algorithm by generating a sample and comparing the sample mean and variance to these theoretical values.

```
trunc_exp <- function(){
# generate a truncated exponetial random variable using the rejection algorithm
x <- runif(1)
y <- runif(1, 0, 1/(1 - exp(-1)))
while (y > exp(-x)/(1 - exp(-1))){
x <- runif(1)
y <- runif(1, 0, 1/(1 - exp(-1)))
}
return(x)
}

# generate a sample of size n
set.seed(30027)
n <- 10000
z <- rep(0, n)
for (i in 1:n) z[i] <- trunc_exp()
# sample mean and variance
mean(z)

## [1] 0.4159879

var(z)

## [1] 0.07836399

# histogram with density on top
hist(z, freq=FALSE, ylim=c(0,1.8))
curve(exp(-x)/(1 - exp(-1)), 0, 1, add=TRUE, col="red")
```



Now consider the alternative algorithm. According to the algorithm, the cdf of Z is

$$
\begin{aligned}
F_Z(z) &= P(Z \le z) \\
&= P(-\ln(U) \le z | U > e^{-1}) \\
&= P(U \ge e^{-z} | U > e^{-1}) \\
&= \frac{P(U \ge \max\{e^{-z}, e^{-1}\})}{P(U > e^{-1})} \\
&= \begin{cases} \frac{P(U \ge 1)}{1-e^{-1}} = 0 & \text{if } z \le 0, \\ \frac{P(U \ge e^{-z})}{1-e^{-1}} = \frac{1-e^{-z}}{1-e^{-1}} & \text{if } 0 < z < 1, \\ \frac{P(U \ge e^{-1})}{1-e^{-1}} = 1 & \text{if } z \ge 1 \end{cases}
\end{aligned}
$$

Thus the pdf of Z is $F'_Z(z) = \frac{e^{-z}}{1-e^{-1}}$, $0 < z < 1$, the same as $p(z)$.

Solution: According to the algorithm, the cdf of X is

$$
\begin{aligned}
F(x) &= P(X \le x) \\
&= P(g(U) \le x | U + V < 1) \\
&= P(U \ge g^{-1}(x) | U + V < 1) \quad \text{(because } g(u) \text{ is decreasing)} \\
&= \frac{P(U \ge g^{-1}(x), U < 1 - V)}{P(U + V < 1)}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{\int_0^{1-g^{-1}(x)} \int_{g^{-1}(x)}^{1-v} 1 du dv}{\int_0^1 \int_0^{1-v} 1 du dv} \\
&= \frac{\frac{1}{2}(1-g^{-1}(x))^2}{\frac{1}{2}} \quad = \quad (1-g^{-1}(x))^2.
\end{aligned}
$$

9. A random number X is to be generated by the following rejection algorithm

   1° Generate two independent Unif$(0,1)$ random numbers U and V.

   2° If $U^2+V^2 \leq 1$, deliver $X = U$; otherwise return to 1°.

   (a) Find the cdf and pdf of X. Also find the mean and variance of X.
   HINT: The integral $\int_0^x \sqrt{1-u^2}du = \frac{1}{2}(\arcsin x + x\sqrt{1-x^2})$.

   (b) What is the efficiency of the algorithm? Denote by N the number of times that step 1° is to be executed to get one X value. Name the distribution of N. Also write down the mean and standard deviation of N.

   (c) Write an R program to implement the algorithm. Then generate a sample of 1000 X values, and give it a numeric and a graphic summary.

Solution: The support of X is $(0,1)$, the same as that of U. The cdf of X is

$$
\begin{aligned}
F(x) &= P(X \leq x) \\
&= P(U \leq x | U^2+V^2 \leq 1) \\
&= \frac{\int_0^x \int_0^{\sqrt{1-u^2}} 1 dv du}{\int_0^1 \int_0^{\sqrt{1-u^2}} 1 dv du} \\
&= \frac{\int_0^x \sqrt{1-u^2}du}{\int_0^1 \sqrt{1-u^2}du} \\
&= \frac{2}{\pi}(\arcsin x + x\sqrt{1-x^2}), \quad 0 < x < 1.
\end{aligned}
$$

The pdf of X is

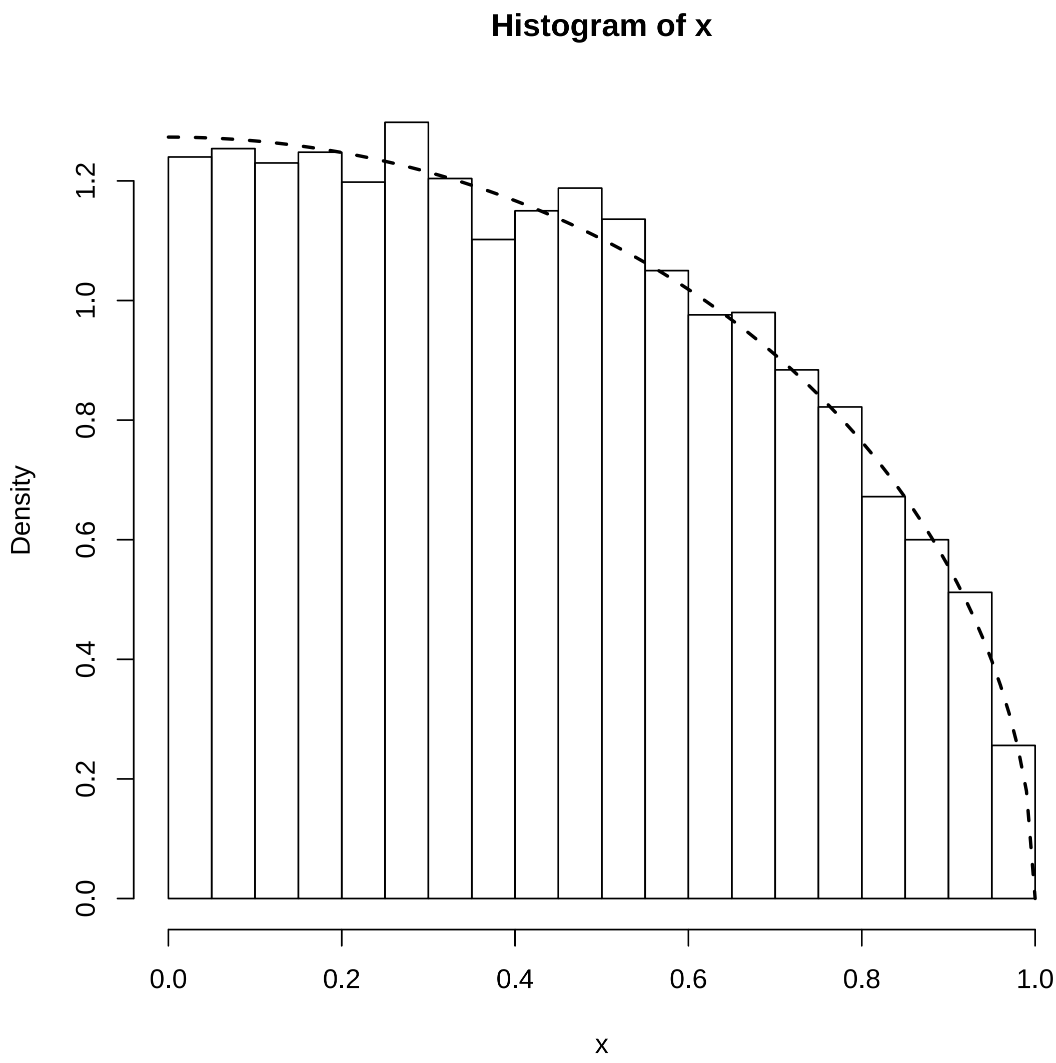
$$
f(x) = F'(x) = \frac{4}{\pi}\sqrt{1-x^2}, \quad 0 < x < 1.
$$

$$
\begin{aligned}
E(X) &= \int_0^1 x f(x) dx = \int_0^1 x \frac{4}{\pi}\sqrt{1-x^2}dx \\
&= -\frac{4}{3\pi}(1-x^2)^{\frac{3}{2}} \mid_0^1 \\
&= \frac{4}{3\pi}. \\
E(X^2) &= \int_0^1 x^2 \frac{4}{\pi}\sqrt{1-x^2}dx \\
&= \frac{1}{2\pi}(\arcsin x - \frac{1}{4}\sin(4\arcsin x)) \mid_0^1 \\
&= \frac{1}{4}. \\
\mathrm{Var}(X) &= \frac{1}{4} - \left(\frac{4}{3\pi}\right)^2.
\end{aligned}
$$

The efficiency $\tau = P(U^2+V^2 \leq 1) = \int_0^1 \int_0^{\sqrt{1-u^2}} 1 dv du = \frac{\pi}{4}$.

$$
\begin{aligned}
N &\overset{d}{=} \mathrm{GEO}(\frac{\pi}{4}) \\
E(N) &= \tau^{-1} = \frac{4}{\pi} \\
\mathrm{Var}(X) &= \tau^{-2}(1-\tau) = \frac{16-4\pi}{\pi^2}
\end{aligned}
$$

```r
xsim <- function(){
u <- runif(1)
v <- runif(1)
while(u^2 + v^2 > 1){
u <- runif(1)
v <- runif(1)
}
return(u)
}
######### Test via simulation
set.seed(12345)
n <- 10000
x <- rep(0, n)
for (i in 1:n) x[i] <- xsim()
summary(x)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000519 0.2018006 0.4098974 0.4264121 0.6369295 0.9971188

hist(x, freq=FALSE)
curve((4/pi)*sqrt(1-x^2), from=0, to=1, add=TRUE, lwd=2, lty=2)
```



2 Workshop

10. Suppose that $\mathbf{X} = (X_1, \ldots, X_k) \sim \text{multinomial}(n, \pi)$ where $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$. Since $X_i \sim \text{bin}(n, \pi_i)$, we have $\mathbb{E}X_i = n\pi_i$ and $\operatorname{Var} X_i = n\pi_i(1 - \pi_i)$. Show that for $i \neq j$, $\operatorname{Cov}(X_i, X_j) = -n\pi_i\pi_j$.

    Hint: just as for the binomial, we can write a multinomial(n, $\boldsymbol{\pi}$) as the sum of n independent multinomial(1, π) random variables.

    Alternative hint: $\operatorname{Var}(X + Y) = \operatorname{Var} X + \operatorname{Var} Y + 2\operatorname{Cov}(X, Y)$.

    Solution: If $\mathbf{X} \sim \text{multinomial}(1, \pi)$ then for $i \neq j$ we have $\mathbb{E}X_iX_j = 0$ and thus $\operatorname{Cov}(X_i, X_j) = 0 - \mathbb{E}X_i\mathbb{E}X_j = -\pi_i\pi_j$. If $\mathbf{X} \sim \text{multinomial}(n, \pi)$ then it can be written as the sum of n independent multinomial$(1, \pi)$, whence we can multiply the covariances by n to get the result.

    Alternatively, if we add X_i and X_j it is just as if we combined these two cases into a single case with probability $\pi_i + \pi_j$. Thus

$$\begin{aligned}
\operatorname{Cov}(X_i, X_j) &= \tfrac{1}{2}(\operatorname{Var}(X_i + X_j) - \operatorname{Var} X_i - \operatorname{Var} X_j) \\
&= \tfrac{1}{2}(n(\pi_i + \pi_j)(1 - \pi_i - \pi_j) - n\pi_i(1 - \pi_i) - n\pi_j(1 - \pi_j)) \\
&= -n\pi_i\pi_j
\end{aligned}$$

11. Suppose that $(X, Y, Z) \sim \text{multinomial}(n, (p_1, p_2, p_3))$. Show that

$$Y|\{X = x\} \sim \text{binomial}(n - x, p_2/(1 - p_1)).$$

Hence obtain $\mathbb{E}(Y|X = x)$.

Solution:

$$\begin{aligned}
\mathbb{P}(Y = y|X = x) &= \mathbb{P}(Y = y, Z = n - x - y|X = x) \\
&= \mathbb{P}(X = x, Y = y, Z = n - x - y)/\mathbb{P}(X = x) \\
&= \frac{n!/(x!y!(n-x-y)!)p_1^x p_2^y p_3^{n-x-y}}{n!/(x!(n-x)!)p_1^x (1-p_1)^{n-x}} \\
&= \frac{(n-x)!}{y!(n-x-y)!}\left(\frac{p_2}{1-p_1}\right)^y \left(\frac{p_3}{1-p_1}\right)^{n-x-y}
\end{aligned}$$

But $p_3/(1 - p_1) = 1 - p_2/(1 - p_1)$, so this is of the right form.

We get immediately that $\mathbb{E}(Y|X = x) = (n - x)p_2/(1 - p_1)$. That is, given $X = x$, we divvy up the remaining $n - x$ trials between Y and Z proportionately to p_2 and p_3.