Section 1: Computational Refresher Exercises (Solutions)

Advanced Quantitative Methods (PLSC 504)

Fall 2017

Exercise 1. The Birthday Problem

In a class of k students, what is the probability that at least two students share a common birthday? Assume that there are 365 possible birthdays (ignoring Feb. 29). The sample space of possible birthdays is $S = 365^k$. Assume that each of these outcomes is equally likely. Let A denote the event that at least two students in the class have the same birthday. Calculate $\Pr(A)$ for a class of $k = 23$ students analytically. Write code to convince yourself of the result you obtained using simulation. Hint: use the `sample()` function to sample from the 365 days with replacement.

Analytic solution:

$$\Pr(A) = 1 - \Pr(A^c) = 1 - \frac{\#(A^c)}{\#(S)}$$

The denominator is $\#(S) = 365^k$. Why? For each person, there are 365 ways to assign that person's birthday. The numerator $\#(A)$ is the number of ways to assign k birthdays in such a way that no two birthdays match. There are 365 ways to assign a birthday to person 1. Conditional on this, there are only 364 ways to assign a birthday to person 2. Then there are 363 ways to assign a birthday to person 3, etc. For the last person, person k, we have already assigned $k-1$ birthdays, so there are $365-(k-1)$ ways to assign a birthday to person k. Therefore,

$$\Pr(A) = 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k} = 1 - \frac{365!/(365-k)!}{365^k} = 1 - \frac{k!}{365^k}\binom{365}{k}$$

Even `R` can't handle 365! (try it). This necessary simplification exercise follows from the fact that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ for $k \leq n$. For $k = 23$ we have

$$\Pr(A) = 1 - \frac{23!}{365^{23}}\binom{365}{23}$$

which gives $\Pr(A) > 0.50$,

```r
1-(factorial(23)/365^23)*choose(365,23)
```

```
## [1] 0.5072972
```

There is also a birthday function in base `R`,

```r
pbirthday(n = 23, classes = 365, coincident = 2)
```

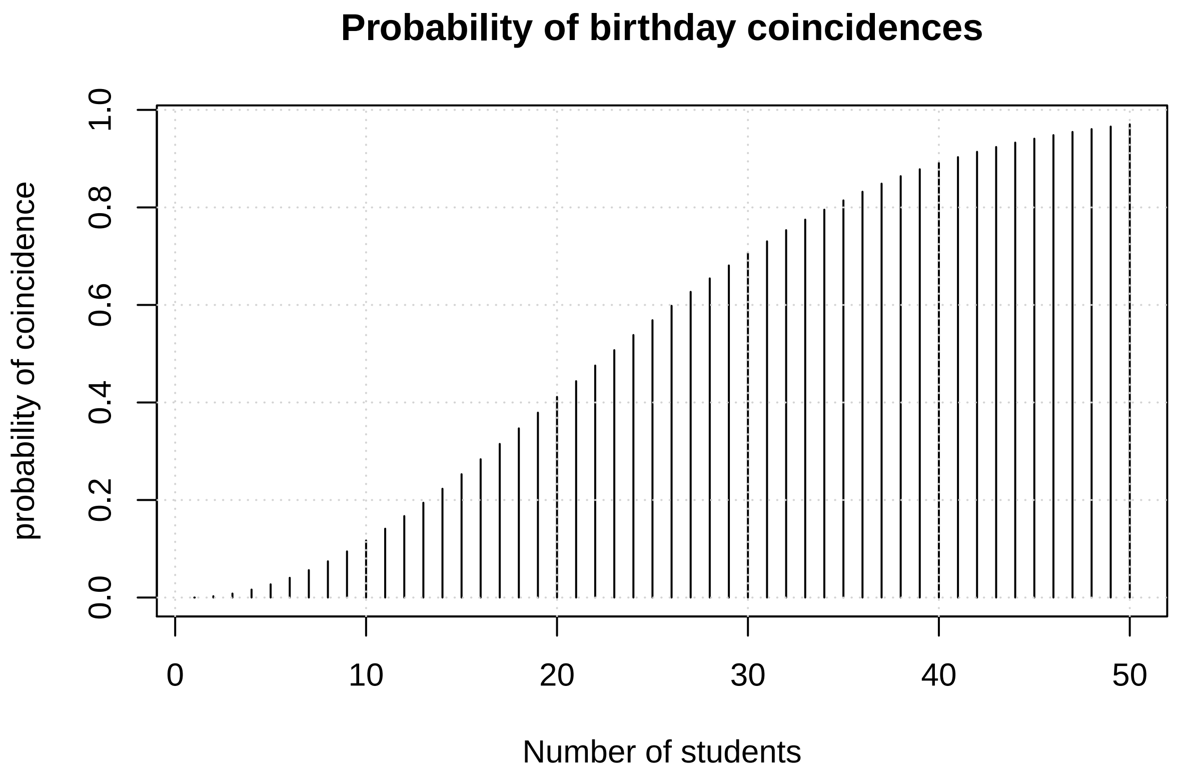
```
## [1] 0.5072972
```

Let's write a function to see what happens for different values of k, say 1 to 50,

```r
# Birthday function
birthday <- function(k) {
  1 - (factorial(k) / 365^k)*choose(365, k)
  }

k <- 1:50
p <- birthday(k)

# Plot the results
plot(k, p, main = "Probability of birthday coincidences",
     xlab = "Number of students", ylab = "probability of coincidence",
     type="h")
grid()
```



Simulation solution:

```r
set.seed(123)

# Number of students in the class
k <- 23

# Number of times to replicate our experiment
sims <- 1000

# A vector to store the output
maxs <- rep(0, sims)

# The simulation. Use the max and table functions to tally the number
#   of matches. If max() returns a value > 1 this means we have
#   atleast one match!
for(i in 1:sims) {
  bdays <- sample(x = 1:365, size = k, replace = TRUE)
  maxs[i] <- max(table(bdays))
}

# Compute proportion of sims with atleast 1 match
mean(maxs > 1)
```

```
## [1] 0.515
```

Alternative method using the `replicate()` function,

```r
set.seed(123)
k <- 23
nsims <- 10^4
reps <- replicate(nsims, max(table(sample(1:365, k, replace = TRUE))))
mean(reps >=2)
```

```
## [1] 0.5046
```

Exercise 2. The Monty Hall Problem

The Monty Hall Problem[1] is a very famous puzzle that you may have learned about in a probability class. The puzzle was originally stated as follows:

> *Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?*

As you may know from the movie "21"[2], the correct answer is to switch. This puzzle generated a bunch of controversy. When Marilyn vos Savant presented the correct solution in her column for *Parade* magazine in 1990, thousands of people wrote in to tell her she was wrong[3].

To avoid confusion for the problem, make the following standard assumptions:

1. The host (Monty) must always open a door that was not picked by the contestant.
2. Monty must always open a door that has a goat behind it and never the car! If he has a choice to make between two goat doors, he chooses at random.
3. Monty must always offer the chance to switch between the originally chosen door and the remaining closed door.

Show that contestants who switch have a better chance of winning the car than contestants who stick with their initial choice. First, provide an analytic solution. Second, convice yourself of this using simulation. For the simulation solution, you should write a function called `monty_hall` that takes two arguments: `sims` for the number of simulations to run and `strategy` for one of three strategy choices: "stay", "switch" or "random". The "random" strategy should just pick one of the three doors at random. The input to `sims` should be an integer (≥ 1000) and the `strategy` argument should be a character. In the function, assume that your initial choice of door is random.

There is no "right" way to do the simulation. You don't need to use any random number generators. The only function you need to use is `sample()`. I would recommend using a for loop and taking advantage of nested if statements. This may not be the most efficient way, but I think it is the easiest. There are actually two methods (that I know of) for the analytic solution. I will show both in section.

A sketch for the algorithm,

```r
monty_hall <- function(strategy = "switch", sims = 1000, ...) {
  # Initialize door vector.

  # Initialize vector to keep track of wins.

  # Inner loop.
  for(i in 1:sims) {
  # Randomly determine winning door.

  # Randomly determine your first choice.

  # The host reveals one of the doors you did not pick (which contains a
  #   goat). [Caution: what happens if you get lucky and pick the winning
  #   door first?]

  # Result if you decide to stay.
```

[1] https://en.wikipedia.org/wiki/Monty_Hall_problem
[2] clip: https://www.youtube.com/watch?v=Zr_xWfThjJ0
[3] have a look at some of the comments she received here: http://marilynvossavant.com/game-show-problem/

```
  # Or if you decide to switch.

  # Or if you pick a random strategy.

  # Tally the outcome for this sim. End loop.
  }

  # Return the win percentage, averaged across all sims.
  return(...)
}
```

Analytic solution:

Label the doors 1 through 3. Assume you pick door 1 (it doesn't matter what door we use here). The host opens another door, with a goat behind it. Let C_i be the event that the car is behind door i. The Law of Total Probability (LOTP) states,

$$\Pr(A) = \sum_i \Pr(A|B_i)\Pr(B_i)$$

Then,

$$\Pr(\text{win car}) = \Pr(\text{win car}|C_1) \cdot \frac{1}{3} + \Pr(\text{win car}|C_2) \cdot \frac{1}{3} + \Pr(\text{win car}|C_3) \cdot \frac{1}{3}$$

Suppose we switch. If the car is in fact behind door 1 (our first choice), then we get a goat, so $\Pr(\text{win car}|C_1) = 0$. If, however, the car is **not** behind door 1, then the revelation of the goat behind door 2 or 3 gives us some valuable information: the car is behind the unopened door. Therefore, switching is a good idea since

$$\Pr(\text{win car}) = 0 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3} = \frac{2}{3}$$

Another solution to this uses Bayes' rule,

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$$

Suppose Monty opens door 2. We can use this to find the conditional probability of success by switching, given the evidence from the door opening. Let M_j be the event that he opens door $j \in \{2, 3\}$. Suppose he opens door 2. Then,

$$\Pr(C_1|M_2) = \frac{\Pr(M_2|C_1)\Pr(C_1)}{\Pr(M_2)}$$

We know that $\Pr(M_2) = \Pr(M_3) = 1/2$ and $\Pr(C_1) = 1/3$. But what is $\Pr(M_2|C_1)$? Well we assume Monty picks at random so his decision is independent of where the car is. Thus $\Pr(M_2|C_1) = 1/2$. Putting this all together,

$$\Pr(C_1|M_2) = \frac{(1/2)(1/3)}{(1/2)} = \frac{1}{3}$$

So, given that he opens door 2, the probability that the car is behind the door you originally chose is 1/3. The probability that the car is behind door number 3 is $1 - 1/3 = 2/3$.

Simulation solution:

```r
monty_hall <- function(strategy = "switch", sims = 1000, first_choice = NULL) {

  # Initialize door vector.
  doors <- 1:3

  # Initialize vector to keep track of wins.
  prize <- NULL

  # Inner loop.
  for(i in 1:sims) {

    # Randomly determine winning door.
    win_door <- sample(doors, 1)

    # Randomly determine your first choice.
    if(is.null(first_choice))
      first_choice <- sample(doors, 1)
    else
      first_choice <- first_choice

    # The host reveals one of the doors you did not pick (which contains
    # a goat).
    if(win_door != first_choice){
      opened_door <- doors[-c(win_door, first_choice)]
    } else {
      opened_door <- sample(doors[-c(win_door, first_choice)], 1)
    }

    # Result if you decide to stay.
    if(strategy == "stay"){
      result <- first_choice
    } else if(strategy == "switch") {
      # Or if you decide to switch.
      result <- doors[-c(opened_door, first_choice)]
    } else {
      # Or if you pick a random strategy.
      result <- sample(doors[-opened_door], 1)
    }

    # Tally the outcome for this sim.
    if(result == win_door) {
      prize[i] <- "Car!"
      } else
        prize[i] <- "Goat!"
    }

  # Return the win percentage, averaged across all sims.
  return(mean(prize == "Car!"))
}
```

Now let's compare the strategies,

```r
set.seed(123)

## Suppose I always pick door number 1 first. What happens if I stay?
monty_hall(strategy = "stay", sims = 10^4, first_choice = 1)
```

```
## [1] 0.3302
```

```r
## What if I switch?
monty_hall(strategy = "switch", sims = 10^4, first_choice = 1)
```

```
## [1] 0.6611
```

```r
## What if I pick my second choice a random?
monty_hall(strategy = "random", sims = 10^4, first_choice = 1)
```

```
## [1] 0.5041
```

```r
## Suppose instead I pick the first choice at random. Does this change
## anything? (Spoiler: No!)
monty_hall(strategy = "stay", sims = 10^4)
```

```
## [1] 0.3348
```

```r
monty_hall(strategy = "switch", sims = 10^4)
```

```
## [1] 0.6652
```

```r
monty_hall(strategy = "random", sims = 10^4)
```

```
## [1] 0.5001
```

As expected, I'm better off switching than staying. I could actually do better than staying by flipping a coin to decide. Note that this win probability is just the average of the other two win probabilities $1/2 \cdot (1/3 + 2/3)$, or just the expected value of a coin flip over two choices :-).

A more fun interactive function that you can play with (from Blitzstein and Hwang pp. 73-74) appears below. Store this function in memory and then type `monty()` into the console to play the game.

```r
monty <- function() {
  doors <- 1:3

  # randomly pick where the car is
  cardoor <- sample(doors, 1)

  # prompt player
  print("Monty Hall says 'Pick a door, any door!'")

  # receive the player's choice of door (should be 1, 2, or 3)
  chosen <- scan(what = integer(), nlines = 1, quiet = TRUE)

  # pick Monty's door (can't be the player's door or the car door)
  if (chosen != cardoor)
    montydoor <- doors[-c(chosen, cardoor)]
  else
    montydoor <- sample(doors[-chosen], 1)

  # find out whether the player wants to switch doors
  print(paste("Monty opens door ", montydoor, "!", sep = ""))
  print("Would you like to switch (y/n)?")
  reply <- scan(what = character(), nlines = 1, quiet = TRUE)

  # interpret what player wrote as "yes" if it starts with "y"
  if (substr(reply, 1, 1) == "y")
    chosen <- doors[-c(chosen, montydoor)]

  # announce the result of the game!
  if (chosen == cardoor)
    print("You won!")
  else
    print ("You lost!")

}
```

Exercise 3. Bootstrapping Feral Camels

The exact size of the Australian feral camel population is unknown. People think it's about 1-1.2 million[4]. Let's assume it is 1.2 million. There are two types, "dromedaries" (*Camelus dromedarius*) and "bactrian" (*Camelus bactrianus*). Suppose we are interested in the proportion of dromedaries in the Australian feral camel population. The target parameter is a propotion. We can't do a camel census, but perhaps we have some valid method of sampling feral camels.

Suppose 30% of the feral camels in Australia are bactrian (B), 65% are dromedaries (D) and 5% are mutants (M). Suppose we only have the resources to sample 1000 camels (they bite, and they are mostly hanging out in the desert). Use the following code to set up the problem,

```r
set.seed(123)

## Create a randomly permuted vector of camel types:
N <- 1.2e6
camel_pop <- sample(c(rep("B", N*0.3), rep("D", N*0.65), rep("M", N*0.05)))

## Confirm it worked
table(camel_pop)/length(camel_pop)
```

```
## camel_pop
##    B    D    M
## 0.30 0.65 0.05
```

The indicator function,

$$\mathbb{1}\{D\} = \begin{cases} 1 & \text{if dromedary} \\ 0 & \text{otherwise} \end{cases}$$

takes a value of 1 if a camel is dromedary, and 0 otherwise. Let $\mathbb{E}[\mathbb{1}\{D\}]$ be our estimator for the proportion of dromedaries.

Most of the time, we only have a single sample from the population to work with, so we can't repeatedly sample from the population. The bootstrap sampling distribution is an estimate of the true sampling distribution. The bootstrap can be used to quantify the uncertainty associated with a given estimator.

There are several different algorithms that we can use to obtain bootstrap confidence intervals. These confidence intervals are approximate, so the probability that some statistic is in the interval is not exactly $1 - \alpha$. Some of the algorithms are more accurate than others. We will use the simplest one, called the "percentile method". The algorithm works like this,

1. Form a dataset of size n, randomly select n observations (with replacement) to produce a bootstrap dataset Z^{*1}.
2. Use Z^{*1} to produce a new bootstrap estimate for θ, which we call $\hat{\theta}^{*1}$.
3. Repeat steps 1 and 2 B times for some large value of B to produce B different bootstrap data sets $Z^{*1}, Z^{*2}, \ldots, Z^{*B}$ and B corresponding estimates $\hat{\theta}^{*1}, \hat{\theta}^{*2}, \ldots, \hat{\theta}^{*B}$.
4. A $1 - \alpha$ bootstrap percentile interval for θ is $C_n = (\theta^*_{\alpha/2}, \theta^*_{1-\alpha/2})$

Take one sample of size 500 from the `camel_pop` vector **without replacement**. Let $B \geq 2500$ and use the algorithm to generate B bootstrap estimates with our estimator from the previous section. Store them in a vector called `theta_boot`. What is the point estimate? What is the bootstrap standard error? Calculate a 95% confidence interval [Hint: use the `quantile()` function]. Plot a histogram and overlay vertical lines for the lower and upper bounds of the 95% interval you obtain.

[4]see: https://www.feralscan.org.au/camelscan/pagecontent.aspx?page=camel_largepopulations

Solution:

```r
set.seed(123)
n <- 1000
camel_samp <-  sample(camel_pop, n, replace = FALSE)

# Number of bootstrap replicates
B <- 5000

# Vector to store estimates
theta_boot <- rep(NA, B)

# Bootstrap w/ a for loop,
for(i in 1:B) {
  Z <- sample(camel_samp, n, replace = TRUE)
  theta_boot[i] <- mean(Z == "D")
}

# 95% intervals
lwr_cl <- quantile(theta_boot, .025)
upr_cl <- quantile(theta_boot, .975)

# Print result
round(c(lwr_cl, upr_cl), 2)
```

```
##  2.5% 97.5%
##  0.62  0.68
```

The point estimate and bootstrap SE are,

```r
# Point estimate
mean(theta_boot)
```

```
## [1] 0.6520152
```

```r
# Approximation for SE
sqrt(var(theta_boot))
```

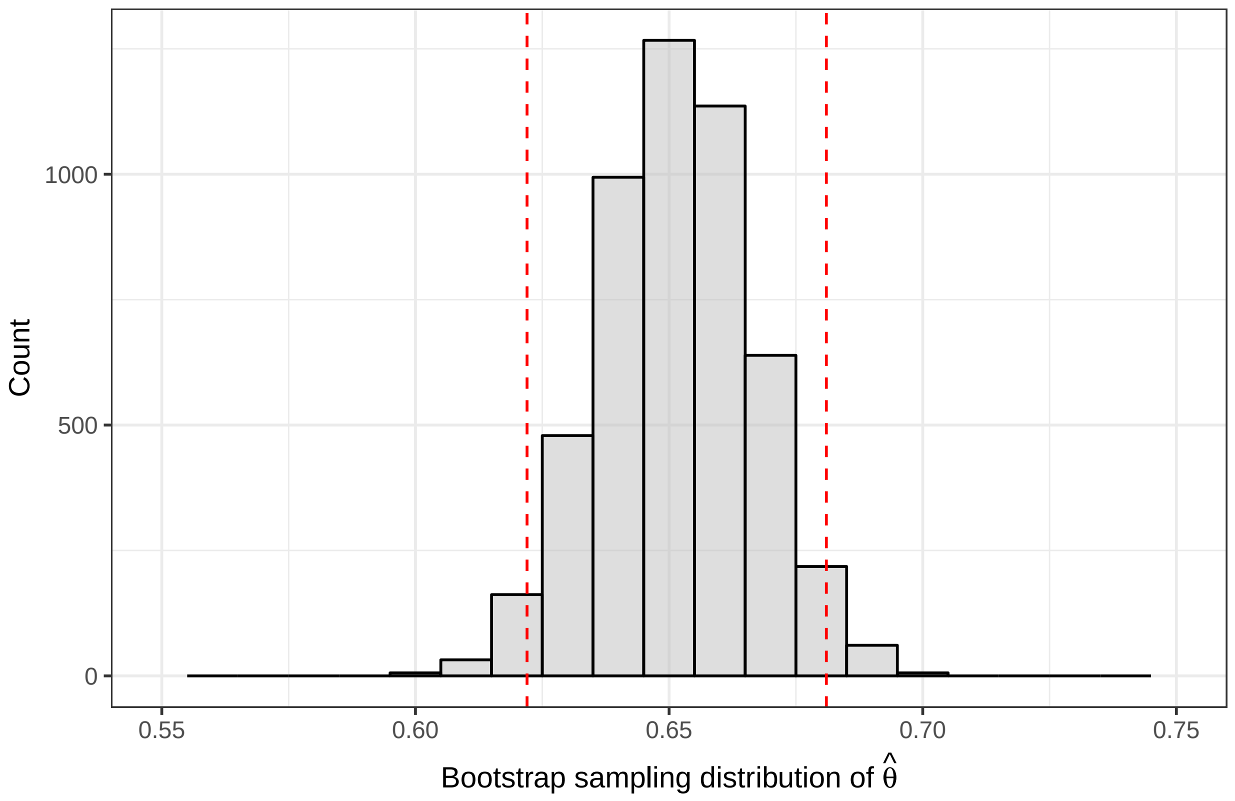
```
## [1] 0.01504848
```

```r
# Using the variance formula for a proportion
sqrt(mean(theta_boot)*(1-mean(theta_boot))*(n-1)^-1)
```

```
## [1] 0.01507045
```

Now make the histogram. Sometimes I like to use `ggplot2()` instead of base `R` because it makes pretty graphs,

```r
library(ggplot2)
qplot(theta_boot, geom="histogram", binwidth = 0.01, main = "",
      xlab = expression(paste("Bootstrap sampling distribution of ",
                              hat(theta))),
      ylab = "Count",
      fill = I("grey"), col = I("black"), alpha = I(0.5), xlim = c(0.55, 0.75)) +
  geom_vline(xintercept = lwr_cl, col = "red", linetype = 2) +
  geom_vline(xintercept = upr_cl, col = "red", linetype = 2) +
  theme_bw()
```


Count
1000
500
0
0.55
0.60
0.65
0.70
0.75
Bootstrap sampling distribution of $\hat{\theta}$