Section 1: Computational Refresher Exercises

Advanced Quantitative Methods (PLSC 504)

Fall 2017

This handout is a refresher of some basic computational exercises that you should (for the most part) already be prepared to tackle. We will work through them together in section, but please try and solve them on your own beforehand. If you haven't seen `R` before, this might look scary. But I think learning by example is the very best way to get started. I saw a tweet[1] the other day that summed up my own personal recipe for learning `R` as a student,

1. Install R
2. Install RStudio
3. Google "How do I [THING I WANT TO DO] in R?"

There are many good introductions to `R` on the internet. Some of my favorite books (in order of difficulty) are,

1. Introduction to Probability and Statistics Using R (https://cran.r-project.org/web/packages/IPSUR/vignettes/IPSUR.pdf). This is a very comprehensive (free) introduction. See Chapter 2 for the basics.
2. The Art of R Programming (https://www.nostarch.com/artofr.htm). I think this is available through Yale Library as an e-book. It's a very useful reference to have.
3. The R Infero (http://www.burns-stat.com/pages/Tutor/R_inferno.pdf). Author's description: "If you are using R and you think you're in hell, this is a map for you." This is a very quirky book, and I found it useful (and funny) when I first started learning.
4. Advanced R (http://adv-r.had.co.nz/). This is a free resource from Hadley Wickham. It's a very good reference for advanced topics.

There are also several good websites to check out for help. These are my favorites,

- https://stackoverflow.com/questions/tagged/r
- https://www.r-bloggers.com/

A couple of useful background references for probability and statistics are,

1. Blitzstein, Joseph K., and Jessica Hwang. *Introduction to probability.* CRC Press, 2014.
2. Wasserman, Larry. *All of statistics: a concise course in statistical inference.* Springer, 2013.
3. Aronow, Peter and Ben Miller. *Foundations of Agnostic Statistics.* Forthcoming. Cambridge University Press.

The first one is, in my opinion, the very best modern introduction to probability, and it includes some `R` code as well. I believe it is available through Yale library for download as a PDF (for free!). Many of the examples I use here are from this book. I don't think the second one is available for download, but I would recommend everyone to buy a copy of it. It's my initial go to reference for all things statistics and probability. Those of you who have taken PLSC 500/PLSC 503 will be familiar with Peter and Ben's book. It's a really fantastic reference and covers of all the background concepts in statistics/probability that you need for this course. If you don't already have a copy and want one, I have uploaded the latest version on Canvas. Peter and Ben have given us permission to use this version of the book for the course, under the provision that you do not share the book with anyone outside this course.

[1] https://twitter.com/kierisi/status/898534740051062785?s=03

If you find yourself struggling with these exercises or the basics of simulation and programming, please reach out early in the course. I am here to help! But there are limits to what can be done. We will be covering alot of material in this course, and review of expected knowledge will be necessarily brief. Simultaneous remedial education is possible if you work hard, but I wouldn't recommend this route if you are taking a full course load.

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.

Exercise 2. The Monty Hall Problem

The Monty Hall Problem[2] 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"[3], 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[4].

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.
```

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

```r
  # 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(...)
}
```

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[5]. 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.

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