MATH 154 - HW4 - Simulating

your name here

due: Thursday, September 30, 2021

summary

In this assignment, you will think carefully about simulations. In class, we did a small simulation to find the average of a maximum. In the notes and slides, there is a more extensive example using the set-up of blackjack to model different game strategies and their respective chances of winning. Part of the goal of the assignment is to code efficiently and cleanly. There are times when the most efficient code is difficult to follow. Your goal should be to create code that is efficient as well as simple to follow.

Other goals of the simulation assignment are to learn to break down a complex problem into simple tasks and to understand how to check that the code you have written is correct. Running lines one at a time (to understand each line) is an important skill.

requisites

Make sure you've read the notes / slides which detail the simulation code. As we did in class, run the different line of code one at a time.

You might want to read the chapter in R for Data Science called The map functions.

assignment

```
library(tidyverse)
```

0. `reprex()` A problem which is not due. On your exam, you will be required to email me a reproducible example. That is, you'll ask a question (e.g., "why don't the lines connect to the dots?") and have code which demonstrates what the code is doing. Good idea to practice now in creating reprexes! that is, as you get stuck, try to come up with a question you could ask which would help getting unstuck. You don't have to use the `reprex()`function, but sometimes it helps. Here is some advice: https://stackoverflow.com/help/minimal-reproducible-example Email me reprexes! Post reprexes to Discord!

1. **Pod Q** Describe one thing you learned from someone in your pod this week (it could be: content, logistical help, background material, R information, something fun, etc.) 1-3 sentences.

2. Run the following lines of code and explain what the `map()` function is doing (i.e., the input and output). [Hint: some will give you errors, describe what went wrong. Also, if you want the file to compile with the error add `error = TRUE` to your code chunk.]

(a) `map(1:5, runif)`

```r
map(1:5, runif)
```

```
## [[1]]
## [1] 0.6150064
##
## [[2]]
## [1] 0.7614572 0.4501000
##
## [[3]]
## [1] 0.033579291 0.004567483 0.184613362
##
## [[4]]
## [1] 0.1856326 0.3897905 0.8844648 0.1436673
##
## [[5]]
## [1] 0.13706758 0.09237606 0.45356300 0.70799405 0.99969584
```

What happens with this map command is the map functions goes through the [1,2,3,4,5] sequence element by element and input whichever element into the runif function as the first argument which then tells the runif function how many random uniform numbers between 0 and 1 to generate.

(b) `map(seq(from = -10, to = 10, by = 5), rnorm, n = 5)`

```r
map(seq(from = -10, to = 10, by = 5), rnorm, n = 5)
```

```
## [[1]]
## [1]  -9.966046  -9.784141 -11.369381 -10.518452 -11.500169
##
## [[2]]
## [1] -5.323874 -6.856984 -3.579201 -4.053493 -3.984711
##
## [[3]]
## [1]  1.05526543 -0.85415088 -0.79918530 -0.93997840  0.04292703
##
## [[4]]
## [1] 5.266964 5.665863 4.861122 4.235003 3.295460
##
## [[5]]
## [1]  9.236858  9.051440 11.040165 10.104060 10.215181
```

What happens with this map command is the map functions goes through the [-10,-5,0,5,10] sequence element by element and inputs whichever element into the rnorm function as the mean argument. The n = 5 argument in the map function then tells the rnorm function to generate 5 random numbers from a normal distribution abound the specified mean given by whichever element in the [-10,-5,0,5,10] sequence the map function is looking at.

(c) `map_dbl(seq(from = -10, to = 10, by = 5), rnorm, n = 5)`

```r
map_dbl(seq(from = -10, to = 10, by = 5), rnorm, n = 5)
```

```
## Error: Result 1 must be a single double, not a double vector of length 5
```

What happens with this map command is the map functions goes through the [-10,-5,0,5,10] sequence element by element and inputs whichever element into the rnorm function as the mean argument. The n = 5 argument in the map function then tells the rnorm function to generate 5 random numbers from a normal distribution around the specified mean given by whichever element in the [-10,-5,0,5,10] sequence the map function is looking at. The error arises with n = 5 as this map function can only generate one random double from a normal distribution around the specified mean at a time so as opposed to 25 random numbers (5 for each sequence element) only one can can be produced for each sequence element.

```
map_dbl(seq(from = -10, to = 10, by = 5), rnorm, n = 1)
```

```
## [1] -10.560960  -4.761163   1.734876   4.676212   9.894367
```

3. **Meeting** Haven and Kai plan to meet at the Coop Fountain. Both are impatient and won't wait more than 10 minutes for the other. However, they didn't specify a time, and simply agreed that they would meet between 8pm and 9pm.

```
n_sims <- 100
arrivals <- data.frame(
  kai = runif(n_sims, min = 0, max = 60),
  haven = runif(n_sims, min = 0, max = 60)
)
```

(a) Write code to estimate the probability of Haven and Kai meeting at the Coop. How likely is it that they will meet? [Hint: two of the early data verbs applied to the `arrivals` dataframe will serve you well here.]

```
n_sims <- 100
set.seed(6666)
arrivals <- data.frame(
  kai = runif(n_sims, min = 0, max = 60),
  haven = runif(n_sims, min = 0, max = 60)) %>%
  mutate(meet_or_no = ifelse(ifelse(kai-haven < 0 , -1*(kai-haven), kai-haven) <= 10, 1, 0)) %>%
  summarise(mean(meet_or_no))

arrivals
```

```
##   mean(meet_or_no)
## 1             0.33
```

(b) What if Kai is known to show up only in increments of 5 min (that is, they would never show up at 8:47pm, only 8:45pm or 8:50pm). Repeat the simulation to estimate the probability that they will meet. (Remember the `seq()` and `sample()` functions from class.)

```
n_sims <- 100
set.seed(4747)
arrivals_2 <- data.frame(
  kai = sample(seq(from = 0, to = 60, by = 5), n_sims, replace = TRUE),
  haven = runif(n_sims, min = 0, max = 60)) %>%
  mutate(meet_or_no = ifelse(ifelse(kai-haven < 0 , -1*(kai-haven), kai-haven) <= 10, 1, 0)) %>%
  summarise(mean(meet_or_no))

arrivals_2
```

```
##   mean(meet_or_no)
## 1             0.25
```

(c) How many simulations are needed to estimate the true probability? Let's simulate to find out how variable the simulations are.

i. First, re-run part (a) with a different `set.seed()` [did you remember to add `set.seed()` to the code which generates their arrival times? if you didn't, go back and do that now.] Report the two "meet" percentages. How different are your two meet percentages?

0.31 for set.seed(4747) 0.33 for set.seed(6666)

ii. Repeat the entire simulation 5000 times for each of three values of `sims = 100, 500, 1000`.

```
simulation_ii <- function(sims){
arrivals <- data.frame(
  kai = runif(sims, min = 0, max = 60),
  haven = runif(sims, min = 0, max = 60)) %>%
  mutate(meet_or_no = ifelse(ifelse(kai-haven < 0 , -1*(kai-haven), kai-haven) <= 10, 1, 0))
return(mean(arrivals$meet_or_no))}
```

```
frame_for_ii_100 <- data.frame(simulated_percents = replicate(5000,simulation_ii(100)))
```

```
frame_for_ii_500 <- data.frame(simulated_percents = replicate(5000,simulation_ii(500)))
```

```
frame_for_ii_1000 <- data.frame(simulated_percents = replicate(5000,simulation_ii(1000)))
```

iii. Write a few sentences describing the simulation results which are provided using the `skim_without_charts()` function (and gives mean, standard deviation, and percentiles).

```
library(skimr)
```

```
skim_without_charts(frame_for_ii_100)
```

Table 1: Data summary

Name	frame_for_ii_100
Number of rows	5000
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
simulated_percents	0	1	0.3	0.05	0.14	0.27	0.31	0.34	0.48

The 100 meeting simulation repeated 5000 times gives us an average meeting probability of 30.5% a standard deviation of 4.55% and the 0th percentile is 15%, the 25th percentile is 27%, the 50th percentile is 30% the 75th is 34% and the 100th is 46%

```
skim_without_charts(frame_for_ii_500)
```

Table 3: Data summary

Name	frame_for_ii_500
Number of rows	5000
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
simulated_percents	0	1	0.31	0.02	0.23	0.29	0.3	0.32	0.39

The 500 meeting simulation repeated 5000 times gives us an average meeting probability of 30.6% a standard deviation of 2.06% and the 0th percentile is 22.2%, the 25th percentile is 29.2%, the 50th percentile is 30.6% the 75th is 32% and the 100th is 37.8%

```
skim_without_charts(frame_for_ii_1000)
```

Table 5: Data summary

Name	frame_for_ii_1000
Number of rows	5000
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
simulated_percents	0	1	0.31	0.01	0.25	0.3	0.3	0.32	0.37

The 1000 meeting simulation repeated 5000 times gives us an average meeting probability of 30.5% a standard deviation of 1.48% and the 0th percentile is 25.6%, the 25th percentile is 29.5%, the 50th percentile is 30.5% the 75th is 31.6% and the 100th is 35.3%

The percentiles get closer together as the number of meetings increases, the sd decreases and the mean approaches 30.5%.

(d) Graph the `coop_results` output to compare the probability estimates under the three different options for `sims`. The graph should indicate that across the three different options for `sims`, one aspect of the output is very similar and one aspect of the output is very different. Include a caption for your graph (use 'fig.cap = "here is my caption" inside the curly brackets of the R chunk).

Here is some code which does most of part (c).

```r
coop_sim <- function(sims = 50) {
  kai <- runif(sims, min = 0, max = 60) # generate sims number of arrivals
  haven <- runif(sims, min = 0, max = 60) # generate sims number of arrivals
  return(
    data.frame(
      num_sims = sims,  # keep track of sims
      meet_prop = simulation_ii(sims) # calculate a single number = proportion of times they met
    )
  )
}

coop_sim(sims = 50)
```

Notice how we're mapping twice! (And each time uses `map_dfr()` because we want the output to be a data.frame.)

```r
reps <- 5000
coop_results <- seq(1, reps, 1) %>%         # integers 1 through reps to repeat process
  map_dfr(~map_dfr(c(100, 500, 1000), coop_sim))  # first the values you are testing in your function
                                              # then the function of interest
library(skimr)
coop_results %>%  # good idea to View(coop_results)!
  group_by(num_sims) %>%  # hold the data by the groups defined by 100, 500, 1000
  skim_without_charts(meet_prop)  # the variable for meeting
```

4. **type I errors** One of the technical conditions for the t-test is that the data start off as normally distributed. Using data which are exponentially distributed with a mean of 200 (`rexp(, rate = 1/200)`), use a simulation to assess whether the empirical type 1 error rate (for exponential data) is correct for the set level of significance (e.g., 0.05).

Repeat the entire analysis for two samples each of size 5 to 50 by units of 5 `seq(from = 5, to = 50, by=5)`).

The steps of the simulation should be:

(a) (done for you) generate two datasets from the same population with the same sample size.
(b) (done for you) perform a `t.test()` to calculate a p-value. Note, you wouldn't expect to reject the null hypothesis because the two groups of data come from the same population! (i.e., the null hypothesis is true).

```
t_test_pval <- function(n_obs){
  x1 <- rep(c("group1", "group2"), each = n_obs)
  y <-  rexp(2*n_obs, rate = 1/200)
  t.test(y ~ x1) %>%
    tidy() %>%
    select(estimate, p.value) %>%
    mutate(n_obs = as.factor(n_obs))  # for graphing, better if not a "number"
}
```

(c) use the `t_test_pval()` function to repeat the process many many times (how many? you'll likely need a few thousand reps) over the range of specified sample sizes. (use `map_df()` twice just exactly like in the Coop meeting example.)

```
frame_4_c <- seq(1, 5000, 1) %>%
  map_dfr(~map_dfr(seq(from = 5, to = 50, by=5), t_test_pval))
```

(d) with the calculated p-values, wrangle your data to find out how often you reject (feel free to use a 0.05 level of significance).

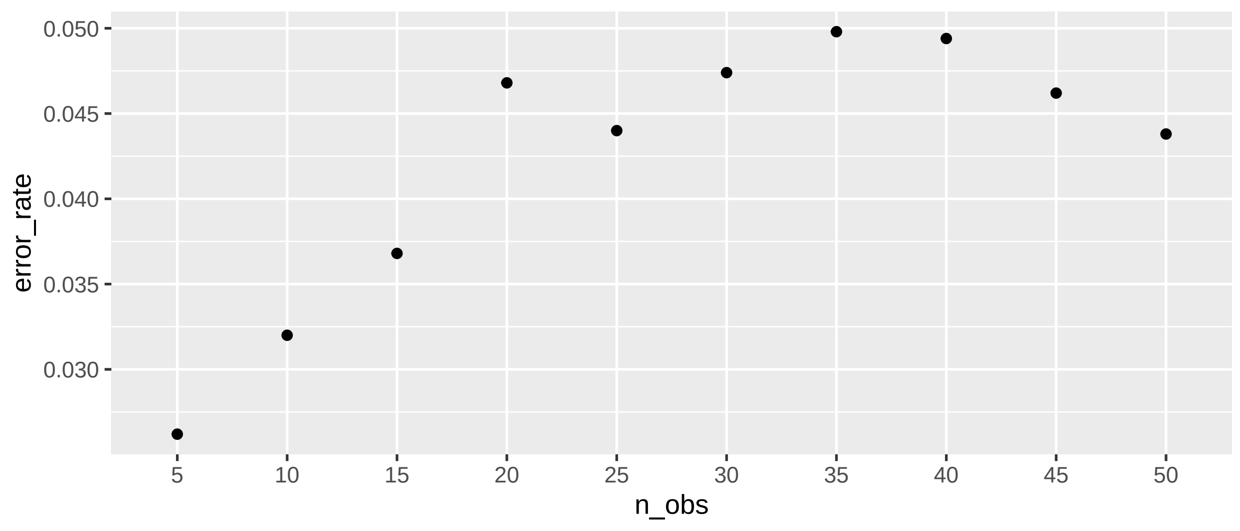
```
frame_4_c <- seq(1, 5000, 1) %>%
  map_dfr(~map_dfr(seq(from = 5, to = 50, by=5), t_test_pval))

frame_4_c %>%
  mutate(rejection_bool = ifelse(p.value <= .05,1, 0 )) %>%
  summarise(frequency_of_rejection = mean(rejection_bool))
```

```
## # A tibble: 1 x 1
##   frequency_of_rejection
##                    <dbl>
## 1                 0.0422
```

(e) plot the empirical type I error rate as a function of sample size

```
frame_4_c %>%
  mutate(rejection_bool = ifelse(p.value <= .05,1, 0 )) %>%
  group_by(n_obs) %>%
  summarise(error_rate = mean(rejection_bool)) %>%
  ggplot(mapping = aes(x = n_obs, y = error_rate)) +
  geom_point()
```



(f) Comment on whether the t-test is robust to the assumption of underlying normal data. [If you are curious, you can also investigate the change in mean of the exponential distribution – more or less skewed. The skewness also affects the appropriateness of the p-value calculation.]

What does it mean for a method to be robust to normality? See Brian Caffo discuss it here: https://www.youtube.com/watch?v=12sjyHGLP2k. Also, check out the course notes: http://st47s.com/Math154/Notes/sims.html#technical-conditions

The t-test is robust to the assumption of underlying normal data as well as to non-normal data with no outliers because of the fact that we have asymptotically correct inferences for both the mean meeting probability as well as for the type 1 error rate as shown in part e.

Below is the function to use to map. The function includes **both** the data generation and the p-value calculation. [n.b., previously we created data first and mapped it into a function. Creating data outside the function is useful to do when we want to keep the data and use it for some reason, maybe to compute residuals, for example. Here, we don't record any of the data because we just want the p-value.]

(a) and (b)

```
t_test_pval <- function(n_obs){
  x1 <- rep(c("group1", "group2"), each = n_obs)
  y <-  rexp(2*n_obs, rate = 1/200)
  t.test(y ~ x1) %>%
    tidy() %>%
    select(estimate, p.value) %>%
    mutate(n_obs = as.factor(n_obs))  # for graphing, better if not a "number"
}
```