[redacted]

[redacted]

by

Scott Greenberg

Date: [redacted]

[redacted]

[redacted]

[redacted]

[redacted]

(i) (30 Points + 4 EC Points) [redacted]: First, a n–sided die ($n \geq 2$) is rolled once (see Figure 1 for examples of such dice). Suppose the outcome is k, where $k \in \{1, \ldots n\}$. Now, this same n–sided die is rolled k times. We are interested in the distributional properties of a random variable called D that represents the **sum** of these k dice rolls (but ignoring the outcome of the very first die roll in the calculation of the sum).

Figure 1: Examples of 4–, 6–, 8–, 10–, 12–, and 20–sided dice.

(a) (4 Points) To better understand this process, *manually* answer this question part: For $n = 2$, write down the 6 possible outcome sequences and their probabilities. They are not equally likely! Then aggregate these outcome probabilities to obtain the probabilities for the 4 possible sums. Keep the probabilities as fractions and not as decimals. Recall from an introductory statistics course how we calculate the mean and variance of a discrete probability distribution. Write down the formulas, then do the actual calculations. So, what are $E(D)$ and $Var(D)$ for $n = 2$?

Answer:

Table 1 shows the 6 possible outcomes for a 2–sided die. The probabilities for the sums (ranging from 1 to 4) have been aggregated in Table 2.

Table 1: Full table for $n = 2$.

1st Roll (k)	k Rolls	Sum (d)	Probability (p)
1	$\{1\}$	1	$\frac{1}{4}$
...	$\{2\}$	2	$\frac{1}{4}$
2	$\{1,1\}$	2	$\frac{1}{8}$
...	$\{1,2\}$	3	$\frac{1}{8}$
...	$\{2,1\}$	3	$\frac{1}{8}$
...	$\{2,2\}$	4	$\frac{1}{8}$

Table 2: Table for sums and probabilities for $n = 2$.

Sum (d)	Probability (p)
1	$\frac{1}{4}$
2	$\frac{3}{8}$
3	$\frac{1}{4}$
4	$\frac{1}{8}$

Thus, it is

$$\begin{aligned} E(D_2) &= d_1p_1 + d_2p_2 + d_3p_3 + d_4p_4 \\ &= 2.25 \end{aligned}$$

and

$$\begin{aligned} Var(D_2) &= E(D_2^2) - (E(D_2))^2 \\ &= 0.9375 \end{aligned}$$

If you do not want to use LaTeX to write up your results, write them up on paper and include a scan of this writeup to your final pdf.

(b) (4 Points) First reset the seed of your random number generator to the last 2 digits of your A-number. If you need any additional R packages, load them here.

```r
set.seed(10)
```

Write an R function that simulates a n–sided die ($n \geq 2$) and then determines the sum of k dice rolls of the same n–sided die. The return value of this function should be just this sum.

Include your R code and test your function as shown under Results.

Answer:

Function:

```r
SimulateNDie <- function(n = 2) {
  # Simulates a n-sided die (n ??? 2) and then
  # determines the sum of the result for that many dice rolls
  # of the same n-sided die
  #
  # Args:
  #  n: Number of faces of the die
  # Returns:
  # Sum of k rolls for a die with n faces
  # (where k is the value of 1 roll of a die with n faces)
  sum(sample(x = n, size = sample(x = n, size = 1, replace = TRUE), replace = TRUE))
}
```

Results:

```r
SimulateNDie() # n = 2 (between 1 and 4)
```

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

```r
SimulateNDie() # n = 2 (between 1 and 4)
```

```
## [1] 4
```

```r
SimulateNDie() # n = 2 (between 1 and 4)
```

```
## [1] 2
```

```r
SimulateNDie(4) # n = 4 (between 1 and 16)
```

```
## [1] 6
```

```r
SimulateNDie(4) # n = 4 (between 1 and 16)
```

```
## [1] 10
```

```r
SimulateNDie(4) # n = 4 (between 1 and 16)
```

```
## [1] 5
```

(c) (4 Points) Write an R function that simulates m of these two–staged dice rolls. For the following parts, work with $m = 10,000$.

Include your R code and test your function as shown under Results.

Answer:

Function:

```
SimulateMExperiments <- function(n = 2, m = 10000) {
  # For a total of m times, simulates a n-sided die (n ??? 2) and then
  # determines the sum of the result for that many dice rolls
  # of the same n-sided die
  #
  # Args:
  #  n: Number of faces of the die
  #  m: number of times we perform the Experiment of
  #     rolling an n-sided die then summing the result that many dice rolls
  #     for the same n-sided die
  # Returns:
  # Sum of k rolls for a die with n faces
  # (where k is the value of 1 roll of a die with n faces)
  replicate(m, sum(sample(x = n, size = sample(x = n, size = 1, replace = TRUE),
                          replace = TRUE)))
}
```

Results:

```
SimulateMExperiments(2, 1) # n = 2 (between 1 and 4)

## [1] 3

SimulateMExperiments(2, 5) # n = 2 (between 1 and 4)

## [1] 2 3 1 3 1

SimulateMExperiments(2, 10) # n = 2 (between 1 and 4)

##  [1] 2 3 2 1 4 2 2 3 2 2

SimulateMExperiments(4, 1) # n = 4 (between 1 and 16)

## [1] 2

SimulateMExperiments(4, 5) # n = 4 (between 1 and 16)

## [1]  8  1  4  5 10

SimulateMExperiments(4, 10) # n = 4 (between 1 and 16)

##  [1] 10  4  4 10  4  3  1  8  2  6
```

(d) (4 Points) Run your function for $m = 10,000$ and $n = 2$. Obtain the empirical probabilities for $P(D = 1), P(D = 2), P(D = 3)$, and $P(D = 4)$. The *table* function in R makes this easy. Also calculate the mean and variance for your sample. Compare with your manual results from part (a). Do they match? Of course, they won"t be exactly the same, but are they close? If so, you are on the right track. If not, check your hand calculations and/or your computer code and try again. You are ready to continue with the remaining parts of this question when your simulation results and the results from your hand calculations are reasonably close.

Include your R code and your final numerical results. Also comment your results in 1 or 2 sentences.

Answer:

```
result.1d <- SimulateMExperiments(2, 10000)
#Empirical Probabilities
table(result.1d)/10000

## result.1d
##      1      2      3      4
## 0.2507 0.3763 0.2521 0.1209

#mean
mean(result.1d)

## [1] 2.2432

#variance
sd(result.1d)^2

## [1] 0.9273465
```

Comment:

My empirical probabilities are pretty close to the ones I calculated by hand in part (a).

The mean and variance are also really close to the ones I calculated by hand in part (a).

(e) (6 Points) Provide two different graphical summaries of your experiment for $n = 4$ and $n = 6$ (for $m = 10,000$) that show the sampling distributions of the sum of the dice rolls. Keep in mind that these are discrete distributions — so do not over–simplify your graphs. Fine–tune your final graphs and include meaningful labels, titles, etc. Include all 4 graphs in a single figure. As the possible sums for $n = 4$ and $n = 6$ widely differ, it is not recommended

to use a common scale for the axes here.

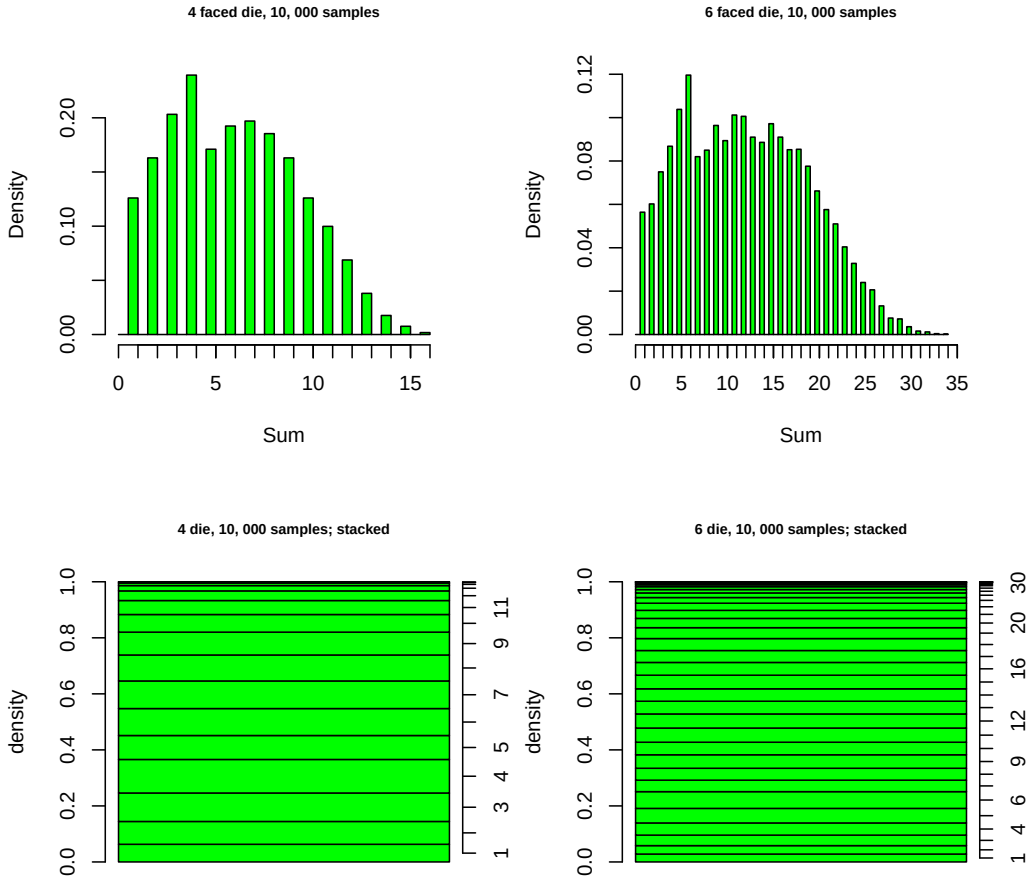
Include your R code and all resulting graphs. Also comment your results in 1 or 2 sentences.

Answer:

```
#Save the simulated data
experiment.n.4.1e <- SimulateMExperiments(4, 10000)
experiment.n.6.1e <- SimulateMExperiments(6, 10000)

#Setting up par to get 4 graphs on 1 result
par(mfrow = c(2, 2), cex.main = .75, mex = 1.05)

#Creating the graphs
hist(experiment.n.4.1e, breaks = (0:(max(experiment.n.4.1e)*2))/2, freq = FALSE,
     right = TRUE, col = "#00ff00", xlim = c(0, max(experiment.n.4.1e)+1),
     main = "4 faced die, 10, 000 samples",
     xlab = "Sum", include.lowest = FALSE)
axis(side = 1, at = 1:max(experiment.n.4.1e), label = FALSE)
hist(experiment.n.6.1e, breaks = (0:(max(experiment.n.6.1e)*2))/2, freq = FALSE,
     right = TRUE, col = "#00ff00", xlim = c(0, 36),
     main = "6 faced die, 10, 000 samples",
     xlab = "Sum", include.lowest = FALSE)
axis(side = 1, at = 1:35, label = FALSE)
barplot(as.matrix(unname(table(experiment.n.4.1e))/10000),
        main = "4 die, 10, 000 samples; stacked",
        col = "#00ff00", ylab = "density", ylim = c(0, 1), beside = FALSE)
cs.4.start <- cumsum(unname(table(experiment.n.4.1e))/10000)
cs.4.end <- ((unname(table(experiment.n.4.1e))/10000)/2)
axis(side = 4, at = cs.4.start-cs.4.end, , labels = 1:max(experiment.n.4.1e))
barplot(as.matrix(unname(table(experiment.n.6.1e))/10000),
        main = "6 die, 10, 000 samples; stacked",
        col = "#00ff00", ylab = "density", ylim = c(0, 1), beside = FALSE)
cs.6.start <- cumsum(unname(table(experiment.n.6.1e))/10000)
cs.6.end <- ((unname(table(experiment.n.6.1e))/10000)/2)
axis(side = 4, at = cs.6.start-cs.6.end,
     labels = 1:max(experiment.n.6.1e))
```




```
#reset par
par(mfrow = c(1, 1))
```

Comment:

We see that increasing the number of faces, produces a shorter but wider graph when we put the bars side by side. When we stack the bars, we see much more categories.

(f) (2 Points) What are $E(D)$ and $Var(D)$, based on your m simulation runs for $n = 4$ and $n = 6$? And what is a 90% confidence interval for the mean of the sums, based on a Normal approximation and your m simulation runs?

Include your R code and your final results.

Answer:

```r
#Expectation for 4 faces
mean(experiment.n.4.1e)

## [1] 6.2122

#Variance for 4 faces
sd(experiment.n.4.1e)^2

## [1] 10.90246

#Confidence interval for the mean of the sums with a 4-faced die.
#Based on a Normal Approximation and 10000 simulation runs.
c(mean(experiment.n.4.1e)+(qt(.05, df = 9999)*(sd(experiment.n.4.1e)/(10000^.5))),
  mean(experiment.n.4.1e)+(qt(.95, df = 9999)*(sd(experiment.n.4.1e)/(10000^.5))))

## [1] 6.157884 6.266516

#Expectation for 6 faces
mean(experiment.n.6.1e)

## [1] 12.3103

#Variance for 6 faces
sd(experiment.n.6.1e)^2

## [1] 45.70178

#Confidence interval for the mean of the sums with a 6-faced die.
#Based on a Normal Approximation and 10000 simulation runs.
c(mean(experiment.n.6.1e)+(qt(.05, df = 9999)*(sd(experiment.n.6.1e)/(10000^.5))),
  mean(experiment.n.6.1e)+(qt(.95, df = 9999)*(sd(experiment.n.6.1e)/(10000^.5))))

## [1] 12.19909 12.42151
```

(g) (6 Points) Summarize your results. Do you think $m = 10,000$ is a good choice for the simulations, or would it be (highly) beneficial to increase m? Justify your answer. Knowing the exact results from the EC part below is not required to answer this question part.

Answer:

For $n = 4$, the least likely sum is 16 with a probability of $\frac{1}{1024}$. With 10000 experiments, we aren"t really cutting it close, as the probability of never getting a sum of 16 is a little larger than 0.00005 or $\frac{1}{20000}$.

For $n = 6$, the least likely sum is 36 with a probability of $\frac{1}{279936}$. With 10000 experiments, we will most likely not see a sum of 36, as the probability of never getting a sum of 36 is a little larger than 0.95 or $\frac{19}{20}$.

I believe it would not be detrimental if we had a simulation where one of the possibilities did not occur if we were to only look at Expectation and

Variance. However, we would see that it would become highly beneficial to increase our value of m if we were to analyze the sample space of our simulations.

(h) (4 EC Points) Find a solution based on Mathematical Statistics for $E(D)$ and $Var(D)$ for $n = 4$ and $n = 6$ and write it up. Moment generating functions, characteristic functions, or conditional expectations might help to answer this question. Show your work (and not only your final result). So, how close do your simulation–based results get to these theoretical results?

Answer:

Looking at this game we can define the first dice roll as a random variable named N. We can also describe the subsequent dice rolls as a sequence of random variables $X_1, ..., X_N$ (We name the last random variable of this sequence X_N because the number of dice rolls we take is determined by random variable N). We can create a a Moment Generating Function which we can define as $M_N(t) = Ee^{tx} = \Sigma_{i=1}^{n} e^{ti}/n$ based on the definition of Moment Generating Functions and Expectation. Because the subsequent dice rolls are also rolling an n sided die, we can say they also have the same MGF. Theorem 4.6.8 from Dr Symanzik's lecture notes for Stat 6710 from 2010 says: *Let* $X_1, ..., X_N$*be iid rv's with common mgf* $M_X(t)$*. Let N be a discrete rv on the non−negative integers with mgf* $M_N(t)$*. Let N be independent of the* $X_i's$*. Define* $S_N = \Sigma_{i=1}^{N} X_i$*, The mgf of* S_N *is* $M_{S_N} = M_N(lnM_X(t))$. Based on this theorem We can say that this game is represented by S_N. We can substitute our formula for $M_X(t)$ in this equation to get $M_{S_N}(t) = M_N(ln(\Sigma_{i=1}^{n} e^{ti}/n))$. We can also substitute in $M_N(t)$ here to get $M_{S_N}(t) = \Sigma_{j=1}^{n} e^{(ln(\Sigma_{i=1}^{n} e^{ti}/n))j}/n$. This can then be simplified down to get $M_{S_N}(t) = \frac{1}{n}\Sigma_{j=1}^{n}(\frac{1}{n}\Sigma_{i=1}^{n} e^{ti})^j$. We can use this to find $E(S_N)$ for n=4 and n=6 by using Theorem 3.2.2. Also from the same notes by Dr Symanzik and then use the variance shortcut $Var(S_N) = E(S_N^2) - E(S_N)^2$ by using the same theorem to calculate $E(S_N^2)$.

(ii) (20 Points) [redacted]

(a) (4 Points) Write a function called `GetUSPop` that reads in the current US population and returns the number as an integer. No need to report intermediate results, so remove all unnecessary code from `L05_WorldPopulation.R`. Test your function three times. There are no arguments to this function, but apparently, each consecutive call should result in a slightly higher number.

Answer:

Function:

```
GetUSPop <- function() {
  # Gets the current US population
  # Args:
  #  None
  # Returns:
  #  Returns the current US population as an integer
  html <- "https://www.livepopulation.com/country/united-states.html"
  clock.html <- readLines(html, warn = FALSE)
  pop.line.num <- grep("<b class=\"current-population\">", clock.html)
  pop.line <- clock.html[pop.line.num]
  p1 <- c("\t\t\t<p class=\"text-18 text-bold c-666 text-center\">")
  p2 <- c("<b class=\"current-population\">|</b></p>")
  pop.text <- gsub(paste0(p1, p2),
                   "", pop.line)
  pop <- as.numeric(gsub(",", "", pop.text))
  pop
}
```

Results:

```
GetUSPop()

## [1] 330015794

Sys.sleep(10) # wait for 10sec

GetUSPop()

## [1] 330015794

Sys.sleep(10) # wait for 10sec

GetUSPop()

## [1] 330015795
```

(b) (6 Points) Write a function called `EstimateUSPopGrowthOnce` that provides one estimate of the current annual growth of the US population. This function **must** call your previous `GetUSPop` function twice.

This function should have one argument called `delay` that specifies the delay between the two calls of the `GetUSPop` function. Units for `delay` should be seconds. Set a default of 10 sec for `delay`. Do the necessary adjustments in your R code to allow for any `delay` $\geq$ 10 sec. If `delay` is $<$ 10 sec, return NA as the time interval will be too short to obtain a meaningful estimate of the growth. Even for a 10 sec `delay`, you may sometimes get a growth of 0.

According to `https://www.npr.org/2019/12/31/792737851/u-s-population-growth-in-2019-is-slowest-in-a-century`, the US population grew by *1, 552, 022* in 2019. The estimate for 2020 should be similar or slightly smaller due to the Coronavirus pandemic. If your results widely differ, check that you correctly adjusted the growth calculation. Test your function six times with the default delay and delays of 5 sec, 10 sec, 30 sec, 60 sec, and 120 sec. Write a general comment, based on what you have noticed. Do not report specific numbers as these may differ slightly from run to run.

Answer:

Function:

```r
EstimateUSPopGrowthOnce <- function(delay = 10) {
  # Provides one estimate of the current annual growth of the US population.
  #
  # Args:
  #  delay: Time in seconds function waits.
  #         Larger values are typically more accurate.
  #         Value must be greater than or equal to 10.
  #         Default value is 10 seconds.
  #
  # Returns:
  #  Provides one estimate of the current annual growth of the US population,
  #  if the given delay is at least 10 seconds. Otherwise returns NA
  if (delay >= 10) {
      pop.start <- GetUSPop()
      Sys.sleep(delay)
      pop.start2 <- GetUSPop()
      60*24*366*60*(pop.start2-pop.start)/delay
  }
  else{
    NA
  }
```

```
}
```

Results:

```
EstimateUSPopGrowthOnce() # default

## [1] 0

EstimateUSPopGrowthOnce(5) # too short

## [1] NA

EstimateUSPopGrowthOnce(10)

## [1] 3162240

EstimateUSPopGrowthOnce(30)

## [1] 1054080

EstimateUSPopGrowthOnce(60)

## [1] 1581120

EstimateUSPopGrowthOnce(120)

## [1] 1317600
```

Comment:

As the delay increases, the Estimate of Population Growth converges to a number that is a little below 1, 552, 022 (The population growth in 2019).

(c) (10 Points) Write a function called `EstimateUSPopGrowthNTimes` that provides N estimates of the current growth of the US population. This function **must** call your previous `EstimateUSPopGrowthOnce` function.

This function should have four arguments called `N`, `inbetween`, `delay`, and `returnValue`. Set the default value to 1 for `N`, to 10 for `inbetween`, to 10 for `delay`, and set `returnValue` to "All"".

`N` indicates how many estimates of the US population growth should be obtained. `inbetween` indicates the time in between consecutive estimates of the US population growth. This should be identical for all consecutive estimates. `delay` is needed for your `EstimateUSPopGrowthOnce` function.

`returnValue` specifies what should be returned from this function: the "Median"", the "Mean"", or "All"". In case of "All"", the estimated growths should be **sorted** from smallest to largest. If the user assigns anything else to this argument, report an error and return NA.

According to `https://www.npr.org/2019/12/31/792737851/u-s-population-growth-in-2019-is-slowest-in-a-century`, the US population grew by *1, 552, 022* in 2019. The estimate for 2020 should be similar or slightly smaller due to the Coronavirus pandemic. If your results widely differ, check that you correctly adjusted the growth calculation. Test your function with the settings below. Write a general comment, based on what you have noticed. Do not report specific numbers as these may differ slightly from run to run.

Answer:

Function:

```
EstimateUSPopGrowthNTimes <- function(N = 1, inbetween = 10,
                                      delay = 10, returnValue = "All") {
  # Provides N estimates of the current growth of the US population.
  #
  # Args:
  #  N: How many estimates of the US population growth should be obtained.
  #     Default value is 1.
  #  inbetween: The time in between consecutive
  #             estimates of the US population growth,
  #             identical for all consecutive estimates.
  #             Default value is 10.
  #  delay: Time in seconds each estimate function waits.
  #         Larger values are typically more accurate.
  #         Value must be greater than or equal to 10.
  #         Default value is 10.
  #  returnValue: Whether the function returns the mean, or median of
  #               the estimates of US population growth, or to return
  #               all of the estimates. Only accepts values of
  #               "Mean", "Median", or "All"; otherwise, function
  #               returns NA. Default value is "All".
  # Returns:
  #  Returns what was requested through returnValue.
  if (returnValue == "All" ||
      returnValue == "Mean" ||
      returnValue == "Median") {
    ki = 1
    pop.growths = c()
    while (ki<= N) {
      pop.growths <- append(pop.growths, EstimateUSPopGrowthOnce(delay))
      ki = ki+1
      Sys.sleep(inbetween)
    }
    if (returnValue == "All") {
      return (sort(pop.growths, decreasing = FALSE))
    }
```

```
    if (returnValue == "Mean") {
      return (mean(pop.growths))
    }
    if (returnValue == "Median") {
      return (median(pop.growths))
    }
  }
  else{
    NA
  }
}
```

Results:

```
# Omit some of these when initially testing your code.

EstimateUSPopGrowthNTimes()

## [1] 0

EstimateUSPopGrowthNTimes(5)

## [1]       0 3162240 3162240 3162240 3162240

EstimateUSPopGrowthNTimes(5, returnValue = "All")

## [1] 3162240 3162240 3162240 3162240 3162240

EstimateUSPopGrowthNTimes(5, returnValue = "Median")

## [1] 3162240

EstimateUSPopGrowthNTimes(5, returnValue = "Mean")

## [1] 0

EstimateUSPopGrowthNTimes(5, returnValue = "NotValid")

## [1] NA

# Use the following to create a meaningful vector.
# However, this will take 10 x (180 + 120) sec, i.e., about 50 min!
# While testing your code, work with smaller numbers !!!
# Only run this once prior to submission. Make sure that you do this
# at least 60min before the submission deadline !!!
# Convert back to the shorter times if this fails and you get close
# to the submission deadline (only -1 point if run with shorter times).

#FinalGrowthEstimate <- EstimateUSPopGrowthNTimes(10,
#                                                 inbetween = 180,
#                                                 delay = 120,
#                                                 returnValue = "All")
FinalGrowthEstimate

## Error in eval(expr, envir, enclos):  object 'FinalGrowthEstimate' not found
```

Comment:

For these we hover around values around 1,552,022 (Growth rate for 2019). With the final growth estimate, there are very little unique values as it seems that it repeats a bit.

The vector is 999999, 1000000, 1000001.

The mean is 10^6 and the standard deviation is 1.

General Instructions

(i) Create a single pdf document, using R Markdown, Sweave, or knitr. When you take this course at the 6000–level, you have to use LaTeX in combination with Sweave or knitr. You only have to submit this one document to Canvas.

(ii) Include a title page that contains your name, your A–number, the number of the assignment, the submission date, and any other relevant information.

(iii) Start your answers to each main question on a new page (continuing with the next part of a question on the same page is fine). Clearly label each question and question part. Your answer to question (i) should start on page 2!

(iv) Show your R code and resulting graph(s) [if any] for each question part!

(v) Before you submit your homework, check that you follow all recommendations from Google"s R Style Guide (see `http://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html`). Moreover, make sure that your R code is consistent, i.e., that you use the same type of assignments and the same type of quotes throughout your entire homework.

(vi) Give credit to external sources, such as stackoverflow or help pages. Be specific and include the full URL where you found the help (or from which help page you got the information). Consider R code from such sources as "legacy code or third–party code"" that does not have to be adjusted to Google"s R Style (even though it would be nice, in particular if you only used a brief code segment).

(vii) **Not following the general instructions outlined above will result in point deductions!**

(viii) For general questions related to this homework, please use the corresponding discussion board in Canvas! I will try to reply as quickly as possible. Moreover, if one of you knows an answer, please post it. It is fine to refer to web pages and R commands, but do not provide the exact R command with all required arguments or which of the suggestions from a stackoverflow web page eventually worked for you! This will be the task for each individual student!

(ix) Submit your single pdf file via Canvas by the submission deadline. Late submissions will result in point deductions as outlined on the syllabus.

1 Citations and References

- (vriesmeys2019, title = How to Add Titles and Axis Labels to a Plot in R, url = https://www.dummies.com/programming/r/how-to-add-titles-and-axis-labels-to-a-plot-in-r/, journal = dummies, author = Vries, Andrie de and Meys, Joris, year = 2019, month = Nov)

- (nidhibiet2020, title = Remove names or dimnames from an Object in R Programming - unname() Function, url = https://www.geeksforgeeks.org/remove-names-or-dimnames-from-an-object-in-r-programming-unname-function/, journal = GeeksforGeeks, author = nidhibiet, year = 2020, month = Jun,)

- (41317943, TITLE = How to plot a CDF functon from PDF in R, AUTHOR = Martin Schmelzer (https://stackoverflow.com/users/1777111/martin-schmelzer), HOWPUBLISHED = Stack Overflow NOTE = URL:https://stackoverflow.com/a/41317943 (version: 2020-10-24) EPRINT = https://stackoverflow.com/a/41317943 URL = https://stackoverflow.com/a/41317943)

- (Bevans, url = https://www.scribbr.com/statistics/confidence-interval/ journal = Scribbr author = Bevans, Rebecca)

- (Symanzik, Title = Math 6710 Lecture Notes Fall 2010, Author = Dr Symanzik)