Stat 5080/6080 Data Technologies Fall 2020

Homework Assignment 1 (09/27/2020)

Niranjan Poudel (A02304550)

Submitted on (10/21/2020)

50 (+ 4 EC) Points — Due Wednesday 10/21/2020 (via Canvas by 11:59pm)

(i) (30 Points + 4 EC Points) Assume we are interested in properties of the following statistical process: 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	1/4
	2	2	1/4
2	$(1,1)$	2	1/8
	$(1,2)$	3	1/8
	$(2,1)$	3	1/8
	$(2,2)$	4	1/8

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

Sum (d)	Probability (p)
1	1/4
2	3/8
3	1/4
4	1/8

Thus, it is

$$\begin{aligned} E(D_2) &= x_1p_1 + x_2p_2 + x_3p_3 + x_4p_4 \\ &= 1 * 1/4 + 2 * 3/8 + 3 * 1/4 + 4 * 1/8 \\ &= 9/4 \end{aligned}$$

and

$$\begin{aligned} Var(D_2) &= E(D_2^2) - (E(D_2))^2 \\ &= 1^2 * 1/4 + 2^2 * 3/8 + 3^2 * 1/4 + 4^2 * 1/8 - (9/4)^2 \\ &= 15/16 \end{aligned}$$

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.

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:

```r
set.seed(50)  # Setting seed before writing function
```

Function:

```r
SimulateNDie <- function(n = 2) {
  # Simulates a n sided die roll and computes the sum of
  # k dice rolls of the same n-sided die.
  #
  # Args:
  #   n: Number of sides for the die
  #
  # Returns
  #   The sum of k dice rolls of the same n-sided die
  #
  # Error handling
  if (n < 2) {
   stop("Argument n must be greater than or equal to 2.")
  }
  else {
    die <- c(1:n)
    value <- sample(die, 1)
    ksum <- sum(sample(die, value, replace = TRUE))
  }
  return(ksum)
}
```

Results:

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

## [1] 3

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

## [1] 1

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

## [1] 3
```

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

## [1] 12

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

## [1] 4

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

## [1] 3
```

(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:

```r
SimulateMExperiments = function(n, m) {
  # Simulates the m experiments for a n sided die to calculate
  #   the sum of k dice rolls of the same n-sided die.
  #
  # Args:
  #   n: Number of side of the die
  #   m: Number of simulations of the experiment
  #
  # Returns
  #   A vector of length m showing sum of k-dice rolls
  ksum <- c()
  for (i in 1:m) {
    die <- c(1:n)
    value <- sample(die, 1)
    ksum[i] <- sum(sample(die, value, replace = TRUE))
  }
  return(ksum)
}
```

Results:

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

## [1] 3

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

## [1] 3 2 1 3 3

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

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

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

## [1] 6

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

## [1]  5 12 14 10 12

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

##  [1] 2 4 2 4 6 8 5 9 7 7
```

(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:

```
exp <- SimulateMExperiments(2, 10000)  # Calling function
prop.table(table(exp))  # Can be viewed as probability as well

## exp
##      1      2      3      4
## 0.2532 0.3748 0.2487 0.1233

# mean and variance
mean(exp)  # Mean from simulation

## [1] 2.2421

var(exp)  # Variance from simulation

## [1] 0.9365812
```

Comment:

Place your answer here

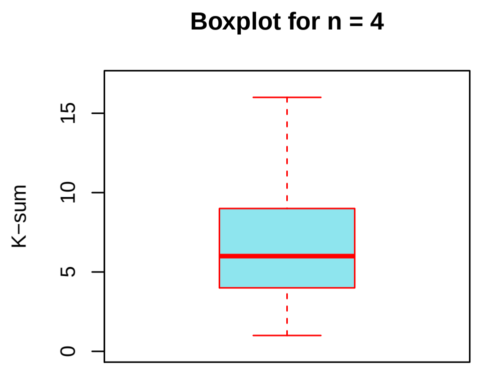
Looking at results from manual calculations (Table 2) and the results from the above simulations we can say that our simulation results and manual calculations are fairly equivalent (differences in the mean, variance and probability being less than 0.009 for all the observations). As we know these two values will very very rarely match, we can say we have got pretty good results and move forward. Furthermore, the real experiment will also rarely match the theoretical or simulation results but it will give us a good understanding of around what values should we anticipate.

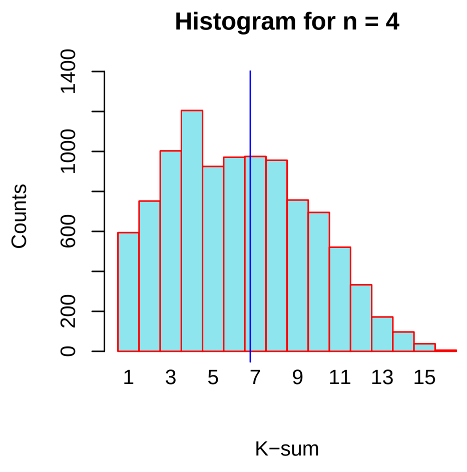
(e) (6 Points) Provide two different graphical summaries of your experiment for $n = 4$ and $n = 6$ (for $m = 10,000$) that show the sampling distributions of the sum of the dice rolls. Keep in mind that these are discrete distributions — so do not over–simplify your graphs. Fine–tune your final graphs 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.

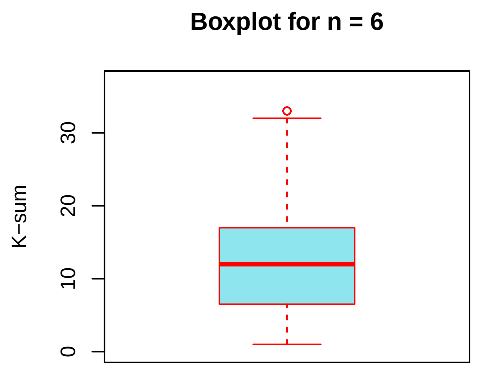
Include your R code and all resulting graphs. Also comment your results in 1 or 2 sentences.

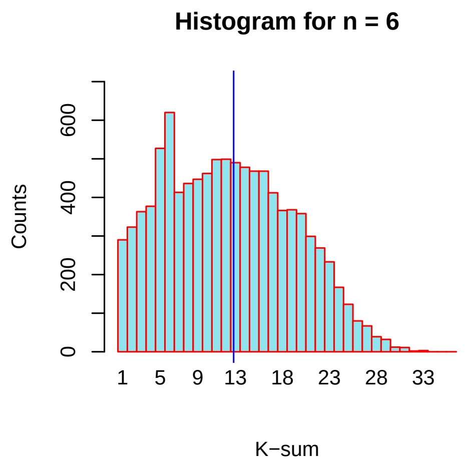
Answer:

```r
par(mfrow = c(2, 2))  # Four plots together
# First of all lets plot for n = 4
sim4 <- SimulateMExperiments(4, 10000)  # Calling function

# Plotting the box-plot initially
boxplot(sim4, main = "Boxplot for n = 4",
        ylim = c(0, 17), ylab = "K-sum",
        col = "cadetblue2", border = "red")
# Next we plot the histogram
# First lets assign histogram values to a list, makes it easy to work around
hst <- hist(sim4, breaks = c(0:16), plot = FALSE)
# Plotting histogram
plot(hst, xaxt = "n", xlab = "K-sum",
     ylab = "Counts", col = "cadetblue2", border = "red",
     ylim = c(0, 1350), main = "Histogram for n = 4")
axis(1, hst$mids, labels = c(1:16),  # For proper x-axis labeling
     padj = - 1.5, tick = FALSE)
abline(v = mean(sim4), col = "blue")  # Just to show the mean line

# Next we use similar coding for plotting n = 6.
sim6 <- SimulateMExperiments(6, 10000)

#Box-pot
boxplot(sim6, main = "Boxplot for n = 6",
        ylim = c(0, 37), ylab = "K-sum",
        col = "cadetblue2", border = "red")
# Next histogram similarly as for n = 4
hst <- hist(sim6, breaks = c(0:36), plot = FALSE)
plot(hst, xaxt = "n", xlab = "K-sum",
     ylab = "Counts", col = "cadetblue2", border = "red",
     ylim = c(0, 700), main = "Histogram for n = 6")
axis(1, hst$mids, labels = c(1:36),
     padj = - 1.5, tick = FALSE)
abline(v = mean(sim6), col = "blue")
```









```
table(sim6)   # Not asked but just to see the data

## sim6
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
## 290 323 363 377 527 620 413 436 447 462 498 499 490 478 468 468 412 366 368 358
##  21  22  23  24  25  26  27  28  29  30  31  32  33
## 299 269 233 167 123  80  67  39  32  12  11   2   3
```

Comment:

Above we can see the box-plot and histogram for 4 sided die and 6 sided die. Looking at the histogram for n = 4 we can see that the probability of getting the sum 4 is the highest whereas probability of getting 16 is the lowest. Also we can see the median is somewhere around 6 and the mean in the histogram is also little bit higher than 6. Similarly looking for plots of n = 6 we can see that the median and mean are somewhere around 12, the probability of getting the sum 6 is the highest and from our simulation we get very low values for sum greater than 30,some of them being zero. Box-plot says some of them are outlires(for n = 6) because even in 10,000 simulation the frequency of some outcomes is very low and even zero for some, which signifies that the probability for some of the higher sum value is very low.

(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:

```
# For n=4
mean(sim4)  # Mean from simulation

## [1] 6.2601

var(sim4)  # Variance from simulation

## [1] 10.70472

# 90 % confidence-interval from Normal approximation
c(mean(sim4) - qnorm(0.95) * sd(sim4) / sqrt(10000),
  mean(sim4) + qnorm(0.95) * sd(sim4) / sqrt(10000))

## [1] 6.206284 6.313916

# for n= 6
mean(sim6)  # Mean from simulation

## [1] 12.3102

var(sim6)  # variance from simulation

## [1] 45.44552

# 90 % confidence-interval from Normal approximation
c(mean(sim6) - qnorm(0.95) * sd(sim6) / sqrt(10000),
  mean(sim6) + qnorm(0.95) * sd(sim6) / sqrt(10000))

## [1] 12.19932 12.42108
```

(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:

Here is the quick summary of the results from questions above

i. The following table compares the result for n = 2 die from manual calculation and simluation.

Table 3: Table comapring manual and simulation results for n =2

Values	Manual	Simulation
Prob fo sum 1	0.25	0.2532
Prob fo sum 2	0.375	0.3748
Prob fo sum 3	0.25	0.2487
Prob fo sum 4	0.125	0.1233
Mean	2.25	2.421
Variance	0.9375	0.93658

We can see 10,000 simulation fairly approaching towards the actual values from the comparison of these values from manual calculation and simulation.

ii. The results in 1 (b) shows a simple simulation of experiments where for the die with n = 2 the sum values are in between 1 an 4 and for n = 4 the sums are between 1 and 16.

iii. The results in 1 (c) generally shows the experiment for the given m-number of values for 2 sided and 4 sided die and the results are in between 1 and 4, 1 and 16 for simultaneous dies.

iv. The results from 1 (d) are summarized in the table 3 compared with manual calculations.

v. In 1 (e) we plotted a box-plot and histogram for four sided die and 6 sided die and visualized the results of 10,000 simulations. From the plot the mean and median values for four sided die is somewhere around 6 and for six sided die the median and mean are some where around 12 (not clear from plots to be exact). It seems like the higher values of the sum have the least probability of occurrence as we can see from histogram and for six sided die some of the results were zero showing their extreme chances of occurrence (does not necessarily mean they will not occur in real life experiments).

vi. Table 4 below summaries the results from 1 (f) for mean and variances for four sided and six sided die along with confidence interval for the expected value (mean).

Table 4: Mean and variances for n = 4 and n = 6 from 10,000 simulations.

n	Mean	Variance	CI
4	6.2601	10.70472	6.206284, 6.313916
6	12.3102	45.44552	12.19931, 12.42108

We do not have the exact values of mean and variance from theoretical calculations up to this point so we cannot make a good judge of how good these values are. But we can see the confidence interval for 4 sided die is narrower than confidence interval for six sided die which makes perfect sense.

vii. **I am finding it hard to explain whether m = 10,000 is a good choice for simulation or not. If we look at the results of 2 sided die it seems to be doing fairly good and even with 4 sided and 6 sided the results seems fairs (we do not have good theoretical value to compare at this point). But, still some of the observation for 6 sided die is equal to zero i.e for sum greater than 33. A good choice of the number of simulation depends on the type of experiment we are conducting, what is the accuracy we want to gain from the experiment and so on. Again, increasing the number of simulations increases the cost (calculation time and may be better computers) so it totally depends on various factors. For this homework purpose i think m = 10,000 is enough to get an idea while having efficient computing time**

(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:

First of all for a single roll of die (n=4) the expected value is

$$\begin{aligned} E(X) &= 1/4 + 2/4 + 3/4 + 4/4 \\ &= 2.5 \end{aligned}$$

Based on our first roll we roll the die again so the expected values should be the average of all possible rolls.

$$\begin{aligned} E(D) &= 1/4(1 * 2.5 + 2 * 2.5.....4 * 2.5) \\ &= 1/4(2.5 + 5 + 7.5 + 10) \\ &= 6.25 \end{aligned}$$

Again for a single roll of die (n=6) the expected value is

$$\begin{aligned} E(X) &= 1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 \\ &= 3.5 \end{aligned}$$

Based on our first roll we roll the die again so the expected values should be the avergae of all possible rolls.

$$\begin{aligned} E(D) &= 1/6(1 * 3.5 + 2 * 3.5.....6 * 3.5) \\ &= 1/6(3.5 + 7 + 10.5 + 14 + 17.5 + 21) \\ &= 12.25 \end{aligned}$$

The expected value for a die (n=4) is 6.25 from theoretical results and from the simulation it is 6.2601.
The expected value for a die (n-6) is 12.25 from theoretical results and from the simulation it is 12.3102
So for a (n=6) the expected value has more deviation than for the 4 sided die.

(ii) (20 Points) In this question, you have to translate the R code from class that was used to estimate the growth of the US population (`L05_WorldPopulation.R`) into three R functions. **As always, show the R code and the results from your functions.**

(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:

```r
GetUSPop <- function(){
  # Reads the current US population from a Web-page:
  #   https://www.livepopulation.com/country/united-states.html
  #
  # Args:
  #   None (not required)
  #
  # Returns:
  #   Current US population in numbers
  clockHTML <-
    readLines("https://www.livepopulation.com/country/united-states.html")

  unlink("https://www.livepopulation.com/country/united-states.html")

  popLineNum <- grep("<b class=\"current-population\">", clockHTML)

  popLine <- clockHTML[popLineNum]

  text1 <- "\t\t\t<p class=\"text-18 text-bold c-666 text-center\""

  text2 <-  "><b class=\"current-population\">|</b></p>"

  text <- paste0(text1, text2)

  popText <- gsub(text, "", popLine)

  pop <- as.numeric(gsub(",", "", popText))

  return(pop)
}
```

Results:

```r
GetUSPop()

## [1] 329999359

Sys.sleep(10) # wait for 10sec

GetUSPop()

## [1] 329999360

Sys.sleep(10) # wait for 10sec

GetUSPop()

## [1] 329999360
```

(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){
  # Computes one estimate of the current annual growth
  #   of the US population reading the current population from
  #   https://www.livepopulation.com/country/united-states.html
  #
  # Args:
  #   delay: Time in seconds between two population readings
  #
  # Returns:
  #   The estimate of current annual growth of US population
  if(delay < 10) {
    warning("Time interval is too short to obtain meaningful estimate")
    return(NA)
  } else {
      first <- GetUSPop()  # Calling previous function
      Sys.sleep(delay)  # Default value is in seconds
      second <- GetUSPop()  # Calling function again
      diff <- second - first  # Difference between two calls
      growth <- diff * (31536000 / delay)  # Annual growth
      return(growth)
  }
}
```

Results:

```
EstimateUSPopGrowthOnce() # default

## [1] 3153600

EstimateUSPopGrowthOnce(5) # too short

## [1] NA

EstimateUSPopGrowthOnce(10)

## [1] 0

EstimateUSPopGrowthOnce(30)

## [1] 2102400

EstimateUSPopGrowthOnce(60)

## [1] 1051200

EstimateUSPopGrowthOnce(120)

## [1] 1576800
```

Comment:
Using`https://www.livepopulation.com/country/united-states.html` ,we estimated the population growth in us by capturing current population at certain time intervals. It looks like the time for a single population growth is greater than 10 seconds so sometime we might capture a single population growth using 10 seconds and sometime may be not. The population growth captured withing 10 seconds interval would either be zero or 1/10 th of current population. As we increase the interval we start to get better results and somewhat around last year growth of 1,552,022. But again the interval we select affects the growth we get and also might also depend on when we execute the code.

(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:

```r
EstimateUSPopGrowthNTimes <- function(N = 1, inbetween = 10,
                                      delay = 10, returnValue = "All") {
  # Computes N estimates of the US population growth
  #
  # Args:
  #   N: Number of estimates to be calculated.
  #   inbetween: Time between consecutive estimates of population growth.
  #   delay: Time in seconds between two population readings from webpage.
  #   returnValue: One of 'Mean', "Median' or 'All"  based on what value
  #                is to be returned.
  #
  # Returns:
  #   Either of mean, median or all of N estimates based on the argument.
  #   returnValue.
  vec <- vector()  # Empty vector
  for (i in 1:N) {
    vec[i] <- EstimateUSPopGrowthOnce(delay)  # Estimates growth
    Sys.sleep(inbetween)  # Time inbetween estimates
  }
  if (returnValue == "Median") {
    return(median(vec))
  } else if (returnValue == "Mean") {
    return(mean(vec))
  } else if (returnValue == "All") {
    return(sort(vec))
  # Error handling
  } else {
    text1 <- "Error: The returnValue argument should contain"
    text2 <- "one of either 'Mean', 'Median' or 'All'"
    text <- paste(text1, text2)
    cat(text)
    return(NA)
  }
}
```

<u>Results:</u>

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

EstimateUSPopGrowthNTimes()

## [1] 0

EstimateUSPopGrowthNTimes(5)

## [1]       0 3153600 3153600 3153600 3153600

EstimateUSPopGrowthNTimes(5, returnValue = "All")

## [1]       0       0 3153600 3153600 3153600

EstimateUSPopGrowthNTimes(5, returnValue = "Median")

## [1] 0

EstimateUSPopGrowthNTimes(5, returnValue = "Mean")

## [1] 3153600

EstimateUSPopGrowthNTimes(5, returnValue = "NotValid")

## Error: The returnValue argument should contain one of either 'Mean', 'Median' or 'All'
## [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")
```

Comment:

Place your answer here

The calculation from the shorter observations are not asmuch useful because as mentioned earlier with 10 second delays the estimates might not be that well. Again while returning vector we will observe zero and 1/10 th of the current population, though mean may differ.

The N estimate for the estimated population growth from the longer calculation or from the vector assigned to FinalGrowthEstimate is 1.314×10^6, 1.314×10^6, 1.314×10^6, 1.314×10^6, 1.314×10^6, 1.314×10^6, 1.314×10^6, 1.5768×10^6, 1.5768×10^6, 1.5768×10^6 sorted in ascending order and the mean is 1.39284×10^6.

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.