Case Study 1

AKSTA Statistical Computing

Konstantinos Vakalopoulos 12223236

2023-04-27

Ratio of Fibonacci numbers

Question 1

Write two different R functions which return the sequence $r_n = \frac{F_{n+1}}{F_n}$ where F_n is the nth Fibonacci number, once using `for` and once using `while`.

```
Fib_for <- function(n){
  #start the sequence with 1 and 1
  fibseq <- c(1,1)
  rseq <- c(1)
  if (n>2){
    for (i in 3:(n+1)){
      fibseq[i] <- fibseq[i - 1] + fibseq[i - 2]
      rseq[i-1] <- fibseq[i]/fibseq[i-1]
    }
  }
  return(rseq)
}
```

```
Fib_while <- function(n){
  #start the sequence with 1 and 1
  fibseq <- c(1,1)
  rseq <- c(1)
  i<-3
  if (n>2){
    while (i<=(n+1)){
      fibseq[i] <- fibseq[i - 1] + fibseq[i - 2]
      rseq[i-1] <- fibseq[i]/fibseq[i-1]
      i <- i + 1
    }
  }
  return(rseq)
}
```

The above functions Fib_for and Fib_while take as input an integer n and based on the Fibonacci sequence, the first two numbers are initialized and the sequence is calculated. Then, it is checked if the number n is greater than 2. If not, the sequence is 1, otherwise the while or the for loop (this depends on the function) is used until the the number of iterations are equal to n+1. Inside the loops, the Fibonacci number and the

sequence is calculated based on the equations: $F_n = F_{n-1} + F_{n-2}$ and $r_n = \frac{F_{n+1}}{F_n}$. Finally, for the while loop, the variable i is increased by one. This is not happening for the for loop because the increase is done automatically.

Question 2

Benchmark the two functions for $n = 100$ and $n = 1000$ (you can use package `microbenchmark` or package `bench` for this purpose). Which function is faster?

```
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.2.3
```

```
bench_for100 <- microbenchmark(Fib_for(100))
bench_while100 <- microbenchmark(Fib_while(100))

print(paste("The time for n=100 using the 'for' loop in seconds:",
            mean(bench_for100$time)/1e6,
            "+/-", round(sd(bench_for100$time)/1e6,6), "seconds"))
```

```
## [1] "The time for n=100 using the 'for' loop in seconds: 0.465237 +/- 3.412126 seconds"
```

```
print(paste("The time for n=100 using the 'while' loop in seconds:",
            mean(bench_while100$time)/1e6,
            "+/-", round(sd(bench_while100$time)/1e6,6), "seconds"))
```

```
## [1] "The time for n=100 using the 'while' loop in seconds: 0.24284 +/- 1.379369 seconds"
```

```
bench_for1000 <- microbenchmark(Fib_for(1000))
bench_while1000 <- microbenchmark(Fib_while(1000))

print(paste("The time for n=1000 using the 'for' loop in seconds:",
            mean(bench_for1000$time)/1e6,
            "+/-", round(sd(bench_for1000$time)/1e6,6), "seconds"))
```

```
## [1] "The time for n=1000 using the 'for' loop in seconds: 1.136152 +/- 0.705556 seconds"
```

```
print(paste("The time for n=1000 using the 'while' loop in seconds:",
            mean(bench_while1000$time)/1e6,
            "+/-", round(sd(bench_while1000$time)/1e6,6), "seconds"))
```

```
## [1] "The time for n=1000 using the 'while' loop in seconds: 1.313006 +/- 1.034175 seconds"
```

In this question, the library `microbenchmark` is used in order to compare the efficiency of the functions, the for and while loop from the previous question. First we benchmark the two functions for $n = 100$ and the average time is calculated. The same thing is done for $n = 1000$. According to the results, for $n = 100$ the results are similar. It is observed that the for loop is slightly faster that the while loop. However, if we take a look at the standard deviation, for the for loop the standard deviation is higher. For $n = 1000$, it can be seen that the for loop is faster that the while loop.

Question 3

Plot the sequence for $n = 100$. For which n does it start to converge? What is this number?

```
library(ggplot2)
fib100 <- Fib_for(100)
df <- data.frame(x = c(1:100), y = fib100)
ggplot(df, aes(x=x,y=y))+
  geom_line()+
  labs(x = "Number of iterations", y = "Fibonacci Sequence")
```

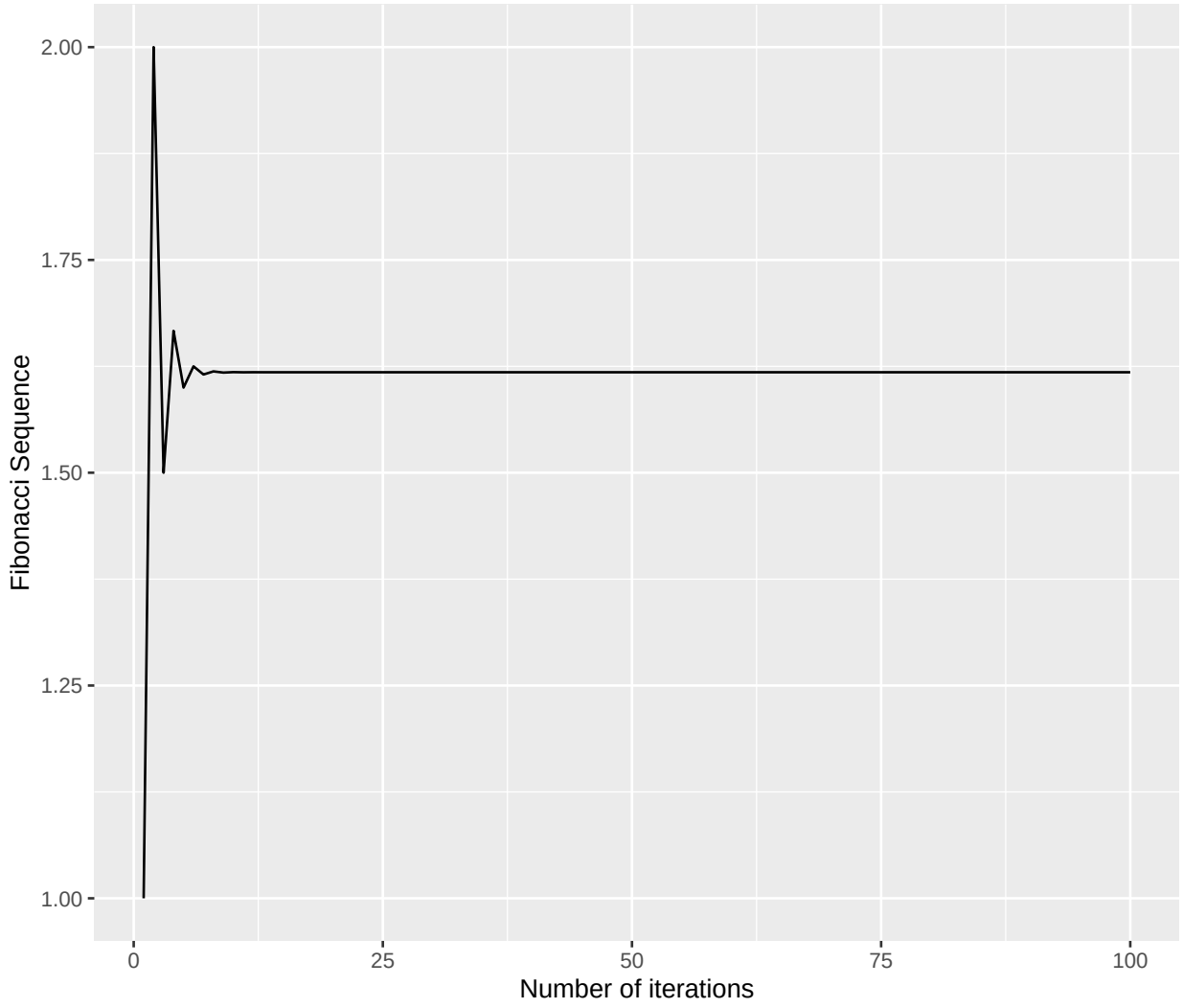

Figure 1: Line plot of the sequence

First, the library `ggplot` is introduced. Then, the function Fib_for is used for $n = 100$ and a data frame with two columns (x and y, where x is a sequence from 1 to 100 and the y is the sequence from the for loop) is used in order then to use the `ggplot` and to plot the sequence. According to the plot can be seen that the plot starts to converge and the value that this convergence is happening is 13.

The golden ratio

Two positive numbers x and y with $x > y$ are said to be in the golden ratio if the ratio between the larger number and the smaller number is the same as the ratio between their sum and the larger number:

$$\frac{x}{y} = \frac{x+y}{x}$$

The golden ratio $\Phi = \frac{x}{y}$ can be computed as a solution to $\Phi^2 - \Phi - 1 = 0$ and is

$$\Phi = \frac{\sqrt{5}+1}{2}$$

Using R, show that the golden ratio satisfies the Fibonacci-like relationship (up to $n = 1000$):

$$\Phi^{n+1} = \Phi^n + \Phi^{n-1}$$

Use once == and once all.equal to compare the two sides of the equation. What do you observe? If there are any differences, what can be the reason?

```
n <- c(1:1000)
FI <- (sqrt(5)+1)/2
left_part <- sapply(n, function(x){FI^(x+1)})
right_part <- sapply(n, function(x){FI^x+FI^(x-1)})

left_part == right_part
```

```
##    [1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
##   [13] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
##   [25]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##   [37] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##   [49]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
##   [61]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##   [73]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
##   [85] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##   [97] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [109]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [121]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
##  [133]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
##  [145]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
##  [157]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [169]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
##  [181] FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [193]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##  [205]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
##  [217] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
##  [229] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [241]  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [253] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [265]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
##  [277]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [289] FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [301] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
##  [313] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [325] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
```

```
##  [337] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##  [349] FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [361]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [373] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [385]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [397]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [409] FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE
##  [421]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [433]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
##  [445] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE
##  [457] FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [469]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [481]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
##  [493]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
##  [505]  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE
##  [517]  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE
##  [529] FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE
##  [541]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
##  [553] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [565] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
##  [577]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##  [589]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [601] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE
##  [613]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
##  [625]  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
##  [637] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE
##  [649]  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
##  [661]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [673]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [685]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
##  [697]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [709] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
##  [721]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [733] FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
##  [745] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE
##  [757] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [769] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
##  [781] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [793]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [805] FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [817] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
##  [829] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [841]  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
##  [853] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [865] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
##  [877]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
##  [889]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##  [901]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [913]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [925]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
##  [937]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [949] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
##  [961]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE
##  [973]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

```
##  [985]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [997]  TRUE FALSE FALSE  TRUE
```

```
all.equal(left_part, right_part)
```

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

First, the n is initialized up to 1000 and the golden ratio is calculated and assigned to the variable FI. Afterwards, the left and the right part of the equation $\Phi^{n+1} = \Phi^n + \Phi^{n-1}$ are calculated using the supply function. The supply function is most optimal way to calculate the two sides of the equation. Finally, the results are compared using once the == and once the function all.equal. Using the operator ==, it can be seen that some of the values are not equal because the result is FALSE. However, when the all.equal is used, the final result is TRUE, which means both sides of the equation are equal. The reason why the results are different, is that the operator == is used to test if the left-hand side is equal to the right-hand side. It returns a logical value of TRUE if they are equal and FALSE if they are not. On the other hand, the function all.equal compares the two objects for near-equality and returns TRUE if they are equal within a tolerance level.

Game of craps

The game of craps is played as follows: First, you roll two six-sided dice; let x be the sum of the dice on the first roll. If $x = 7$ or $x = 11$ you win, otherwise you keep rolling until either you get x again, in which case you also win, or until you get a 7 or 11, in which case you lose. Write a program in R to simulate a game of craps. Explain the steps of your program. If the code is not explained, no points will be earned.

```
roll <- sample(1:6, 2, replace = TRUE)
x <- sum(roll)
print(paste("You roll: ", x))
```

```
## [1] "You roll:  5"
```

```
if (x==7|x==11){
  print("You won the game of craps!")
}else{
  print("Keep rolling the dice")
  repeat{
    roll <- sample(1:6, 2, replace = TRUE)
    x_temp <- sum(roll)
    print(paste("You roll: ", x_temp))
    if (x==x_temp){
      print("You won the game of craps!")
      break
    }else if (x_temp==7|x_temp==11){
      print("You lost the game of craps!")
      break
    }else{
      print("Keep rolling the dice")
    }
  }
}
```

```
## [1] "Keep rolling the dice"
## [1] "You roll:  7"
## [1] "You lost the game of craps!"
```

First, for the dice roll, the function sample is used from 1 to 6 two times with replacement and then the results are added together (variable x). Afterwards, it is checked if the sum is equal to 7 or 11. If so, the message "You won the game of craps!" is printed to screen of the participant, otherwise the message "Keep rolling the dice" is printed. To keep rolling the dice the repeat loop is used, again the sample function is used and then we sum the results of the dice (variable x_temp). Consequently, the sum is checked. If the sum (x_temp) is equal to the sum of the dice on the first roll (x), a message "You won the game of craps!" is printed and we exit from the loop. If the sum (x_temp) is equal to 7 or 11, a message "You lost the game of craps!" is printed and we exit again from the loop. Finally, if none of the previous cases happen, we do not exit the repeat loop and we keep rolling the dice until we win or lose.

Readable and efficient code

Read over the code below and perform the following:

- Wrap it into a function `foobar0` which has arguments `x` and `z` and which returns the vector `x` at the end of the following code.
- Rewrite this into a function `foobar` which is easier to read, by reducing repetitive code. E.g. `foobar` might call a function to check the input, and another function to perform the three lines of computation. Also, use a function such as `paste` to create the error messages.
- Check that the two versions produce the same output using the function `all.equal`.

Part 1

```
foobar0 <- function(x,z){
  if (sum(x >= .001) < 1) {
    stop("step 1 requires 1 observation(s) with value >= .001")
  }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- sin(r) + .01
  if (sum(x >= .002) < 2) {
    stop("step 2 requires 2 observation(s) with value >= .002")
  }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 2 * sin(r) + .02
  if (sum(x >= .003) < 3) {
    stop("step 3 requires 3 observation(s) with value >= .003")
  }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 3 * sin(r) + .03
  if (sum(x >= .004) < 4) {
    stop("step 4 requires 4 observation(s) with value >= .004")
  }
```

```
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 4 * sin(r) + .04
  return(x)
}
```

```
set.seed(1)
x <- rnorm(100)
z <- rnorm(100)
foobar0_result <- foobar0(x,z)
```

First, the set.seed is used in order to get the same values for x and z. Then, the function foobar0 is created and take as input the variables x and z. Inside the function foobar0 there is the code given from the exercise. The result of the foobar0 is assigned to the variable foobar0_result.

Part 2

```
input <- function(x,threshold,step){
  if (sum(x >= threshold) < step) {
    paste("step", step, "requires", step, "observation(s) with value >= ", threshold)
  }
}
```

```
computation <- function(x,z,mul,add){
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- mul * sin(r) + add
  return(x)
}
```

```
foobar <- function(x,z){
  input(x,0.001,1)
  x <- computation(x,z,1,0.01)

  input(x,0.002,2)
  x <- computation(x,z,2,0.02)

  input(x,0.003,3)
  x <- computation(x,z,3,0.03)

  input(x,0.004,4)
  x <- computation(x,z,4,0.04)
  return(x)
}
```

```
set.seed(1)
x <- rnorm(100)
z <- rnorm(100)
foobar_result <- foobar(x,z)
```

Again, the set.seed is used in order to get the same values for x and z. Then, the function foobar is created and take as input the variables x and z. Inside the function foobar there are two new functions. The first function is the so called "input" function which takes as input the variable x, the step of the procedure, which is also the minimum number of observations, and a threshold value. The "input" function had been created in order to create the error messages using the function paste. The second function is the "computation" function. This function is responsible for the three lines of computation, which are the linear regression model, the residuals and the calculation of x. The function takes as input the initial x or the modified x, the z, a variable mul, which takes the values 1, 2, 3 and 4, and a variable add, which takes the values 0.01, 0.02, 0.03 and 0.04.

Part 3

```
all.equal(foobar0_result,foobar_result)
```

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

Finally, the function all.equal is used in order to check that the two versions (foobar0_result and foobar_result) produce the same output using the function. According to the output, the result is the same.