Prof. Dr. Steffen Wagner
Angewandte Statistik
Fachbereich II
Berliner Hochschule für Technik

Exercise 07: Control structures

Statistical Computing – WiSe 2022/23

Introductory exercises

`for` loops

Write a `for` loop to compute

$$\sum_{i=1}^{10} 0.5^i.$$

```
q <- 0.5
n <- 10

res <- 0
for(i in 1:n){
  res <- res + q^i
}
res
```

```
## [1] 0.9990234
```

Now move this `for` loop structure inside a function named `geomSeries` with the arguments `q` and `n` so that the function returns

$$f(q, n) = \sum_{i=1}^{n} q^i.$$

```
geomSeries <- function(q, n) {
  res <- 0
  for(i in 1:n){
    res <- res + q^i
  }
  res
}
```

Your function should return

```
geomSeries(0.5, 10)
```

```
## [1] 0.9990234
```

`while` loops

Suppose S_n is defined as

$$S_n = \sum_{i=1}^{n} 0.5^i.$$

Use a `while` loop to calculate S_n until

$$|S_n - S_{n-1}| < 10^{-6}.$$

```
q <- 0.5
tol <- 1e-6

res <- 0
i <- 1
term <- q^i
```

```r
while(abs(term) >= tol){
  # summation
  res <- res + term

  # increase of loop variable
  i <- i + 1
  # new addtive term
  term <- q^i
}
res
```

```
## [1] 0.9999981
```

Functions, default values and `if-else`-structures

Now alter the function `geomSeries` that it accepts the arguments `q`, `n` and `tol`. Use the following default values:

```r
geomSeries <- function(q, n = NULL, tol = 1e-6){
  # your code here
}
```

The wanted functionality of the modfied function `geomSeries` is as follows:

- if the argument `n` is used then `geomSeries` should return

$$f(q, n) = \sum_{i=1}^{n} q^i.$$

- if the argument `n` is not used (i.e. `is.null(n) == TRUE`), than `geomSeries` should return an approximation of the infinite sum

$$f(q) = \sum_{i=1}^{\infty} q^i$$

until $|S_n - S_{n-1}| < \text{`}tol\text{'}$.

```r
geomSeries <- function(q, n = NULL, tol = 1.e-6) {

  # Check if argument `n` is used or not
  if(!is.null(n)){
    # calculate for loop
    res <- 0
    for(i in 1:n){
      res <- res + q^i
    }
  } else {
    # calculate while loop
    res <- 0
    i <- 1
    term <- q^i
    while(abs(term) >= tol){
```

```r
      # summation
      res <- res + term

      # increase of loop variable
      i <- i + 1
      # new addtive term
      term <- q^i
    }
    res
  }
  return(res)
}
```

Have fun with programming in R! Your modified function `geomSeries` should work as follows:

```r
geomSeries(0.5, n = 3)
```

```
## [1] 0.875
```

```r
geomSeries(0.5, n = 5, tol = 1e-4)
```

```
## [1] 0.96875
```

```r
geomSeries(0.5, tol = 1e-4)
```

```
## [1] 0.9998779
```

The $\sqrt{N}$ law

The core function

Write a function `meanVarSdSe` that takes a numeric vector `x` as argument. The function should return a named vector that contains the mean, the variance, the standard deviation *sd* and the standard error *se* of `x`. The standard error is defined as

$$se(x) = \frac{sd(x)}{\sqrt{\#x}}, \tag{1}$$

where $\#x$ denotes the cardinality, i.e. the number of elements contained in `x`.

The code should have the following structure

and return a named vector according to

```
meanVarSdSe <- function(x){
  n    <- length(x)
  c(mean = mean(x),
    var = var(x),
    sd = sd(x),
    se = sd(x)/sqrt(n))
}
```

```
x <- 1:100
meanVarSdSe(x)
```

```
##       mean        var         sd         se
##  50.500000 841.666667  29.011492   2.901149
```

You can use the functions `mean`, `var`, `sd` and `length`. Check the help files for these functions for further arguments that can be used optionally.

Look at the following code sequence. What result do you expect?

```
x <- c(NA, 1:100)
meanVarSdSe(x)
```

Now run the code. Explain the result. Extend the function definition of `meanVarSdSe` with the argument `...`, as is illustrated as follows:

```
meanVarSdSe <- function(x, ...){
  c(mean = mean(x, ...),
    var = var(x, ...),
    sd = sd(x, ...),
    se = sd(x, ...)/sqrt(sum(!is.na(x))))
}
```

so that the `na.rm = TRUE` argument can be passed optionally to the functions `mean`, `var` and `sd`. What is the correct value for $\#x$ in the case of missing values? Use `sum(!is.na(x))` as denominator in eq. (1). Read the help page for the function `is.na()`. The optimized function should return

```
meanVarSdSe( c(x, NA), na.rm = TRUE)
```

```
##       mean        var         sd         se
##  50.500000 841.666667  29.011492   2.901149
```

Convergence and Control Structures

The $\sqrt{N}$ Law: the precision of the sample average improves with the square root of the sample size N. Simulate 10^6 Poisson distributed random numbers with expectation value $\lambda = 100$ via

```
set.seed(1)  # why?
x <- rpois(n = 1e6, lambda = 100)
```

Write a loop (`repeat` or `while`) which calculates the mean, variance, standard deviation and standard error of the first N elements of `x` until 'se' ≤ 0.05. Start with $N = 2$ and multiply N after each iteration by a factor of 2. Store N and the calculated values for each iteration as rows in a matrix named `result`. Use the function `meanVarSdSe` developed in exercise,. Make sure that the column names of your result matrix match the following output. Your matrix should look like

```
tol <- 0.05
factor <- 2
```

```
result <- NULL
n <- 2
repeat{
  result  <- rbind(result, c(N=n, meanVarSdSe(x[1:n])))
  if(result[nrow(result), "se"] <= tol) break
  n <- n * factor
}
```

```
tail(result)  #  see ?tail for details
```

```
##             N      mean       var       sd         se
## [11,]  2048  99.90137 104.63414 10.22908 0.22603293
## [12,]  4096  99.97803 101.65813 10.08257 0.15754008
## [13,]  8192 100.02466  99.13747  9.95678 0.11000792
## [14,] 16384 100.00885 100.58669 10.02929 0.07835384
## [15,] 32768 100.04257 101.13827 10.05675 0.05555623
## [16,] 65536  99.97209 100.32065 10.01602 0.03912508
```

Question: Why is a `for` loop not optimal for the specific task?

Check: Are these values plausible?

- Extract the columns `se` and `N` from the `result` matrix and store in individual vectors named `se` and `N`.

  ```
  se <- result[ , "se"]
  N <- result[, "N"]
  ```

- Plot the standard error versus the sample size in a scatter plot. Make use of the additional argument `log = 'xy'`. Discuss the result.

```r
plot(N, se, log = "xy")
```

- Now perform a linear regression and model the logarithm of the standard error `se` as a linear function of `log(N)`. What coefficient $\widehat{b}_1$ do you expect according equ. (1)?

```r
lm01 <- lm(log(se) ~ log(N))
summary(lm01)
```

```
##
## Call:
## lm(formula = log(se) ~ log(N))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.17541 -0.05660  0.01691  0.02669  0.37975
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.26831    0.06804   33.34 9.71e-15 ***
## log(N)      -0.49841    0.01015  -49.10  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1297 on 14 degrees of freedom
## Multiple R-squared:  0.9942, Adjusted R-squared:  0.9938
## F-statistic:  2410 on 1 and 14 DF,  p-value: < 2.2e-16
```

Something's not working - why?

Lower the break condition to 'se' ≤ 0.01. Run the changed code. Make sure that you reinitialize `N` and `result` before running the loop. What happens? Compare `N` and `length(x)`. Which expression causes the error?