Data Analytics: Methoden und Programmierung

Assignment 1: computational statistics

```
source(file="assignment1_functionLib.R")
```

This assignment serves two purposes. First, it will train your R programming skills, in particular writing your own functions. Second, it serves as an introduction to computational statistics (or: understanding statistics by simulation). The idea is to study well-known problems by simulation.

Feel free to work in groups (maximum of 4 students per group). Each group must submit at least one R-file containing well-documented functions and test cases to test the functions. You may use two files (one for the functions and another one for the testcases), but this is not necessary. Write your answers and explanations as comments into the R-File. You may submit an Rmd-file instead of an R-File, but this not necessary.

A note on grading: There will be a written exam (50%), the in-class quizzes (10% for both classes; the Data Science and the Programming class) and two home assignments (each 15%). The groups will be asked to present their home assignments in class. There will be a single final grade calculated from the sum of all achieved points. In total, you can earn a maximum of 100 points (=100%).

This home assignment has 15 points (=15%) and is due Sunday, May 30, 6PM

1 Estimation (3 Points)

Let Y denote a discrete random variable describing rolling a usual die with six sides.

1.1 Expectation and variance

Assume that each side is observed with equal probability. Compute (on a sheet of paper) the expectation and the variance of the random variable Y. Insert your calculations and the answer as a comment into the R-File.

1.1.1 Expectation and variance by simulation

Now produce some code to simulate rolling the die. The computer should roll the die a few thousand times (e.g. 10000) and estimate the expectation and the variance from this simulation data. Write a function by completing this skeleton.

```
sim_die <-  function(n=10000){


  #expectation/mean
  e <-
  #variance
  v <-
  return(c(e,v))
}
```

1.2 Variance of the mean

Assume that you roll the die 20 times and then compute the average ($\bar{Y} = \frac{1}{n}\sum_i Y_i$) of the 20 values. This describes a new experiment and the average ($\bar{Y}$) is a new random variable. Compute (on a sheet of paper) the expectation and the variance of $\bar{Y}$

1.2.1 By simulation

Write a function that computes the average of 20 rolls several thousand times and estimates the expectation and the variance of $\bar{Y}$

2 Hypothesis testing (12 Points)

A population is described by a random variable X and you would like to perform a hypothesis test regarding the expectation of the population. For example, the random variable could describe the distribution of returns from a particular investment. A possible interesting hypothesis is –for example– whether the expected return is larger than 2 percent. Letting μ_X denote the unknown expected value, you wish to test $H_0 : \mu_X \leq 2$ against $H_1 : \mu_X > \mu_0 = 2$. For simplicity, consider only this one-sided hypothesis testing problem in the following.

Let X_i, $i \in [1, n]$ denote random draws from the population. Assume that the population of returns is characterized by a Normal distribution. Thus, X_i are iid random variables following a Normal distribution with expectation μ_X and variance σ_X^2.

In the following, you are asked to write several functions for carrying out this hypothesis testing problem.

2.1 A single t-statistic

Write a function computing a single t-statistic. This function has two arguments:

1) `x`: vector having the realizations x_i.
2) `mu0`: the value according to the null hypothesis

```
#generate a vector of random variables
x <- rnorm(1000,mean=10,sd=3)

#compute a single t statistic
tvalue <- my.tstat(x,mu0=8) #note H_0 is false here
tvalue
```

```
## [1] 21.67301
```

2.2 Generating a matrix of random variables

Write a function `my.genSampleMatrix` that generates a matrix of random variables. This function should generate several thousand times a sample of size n. The arguments are:

1) `nReps`: the number of replications. This value will govern the precision of your simulation and is typically equal to several thousand.
2) `nSample`: the sample size of each sample ($=n$)
3) `mu`: the true expected value of the population
4) `sigma`: the true σ of the population

The function should create a matrix where *each row* is a particular sample. The number of rows is equal to `nReps`.

```
#generate a matrix of random variables
my.samMatrix <- my.genSampleMatrix(nReps=2, nSample=5, mu=2, sigma=2)
my.samMatrix
```

```
##            [,1]     [,2]     [,3]      [,4]     [,5]
## [1,]  0.9344014 1.360619 1.322071 2.5589608 1.151102
## [2,] -0.2159358 5.358126 3.423067 0.6043203 6.037797
```

2.3 Vector of t-statistics

Write a function that accepts the matrix of random variables as an input and computes a vector of t-statistics. That is, it should compute the t-statistic for each sample in the matrix. This function should make use of your previously defined function that computes the t-statistic for a single vector.

The arguments are:

1) `sampleMatrix`: the matrix containing the samples.
2) `mu0`: the value according to the Null hypothesis.

```
my.samMatrix
```

```
##            [,1]     [,2]     [,3]      [,4]     [,5]
## [1,]  0.9344014 1.360619 1.322071 2.5589608 1.151102
## [2,] -0.2159358 5.358126 3.423067 0.6043203 6.037797
```

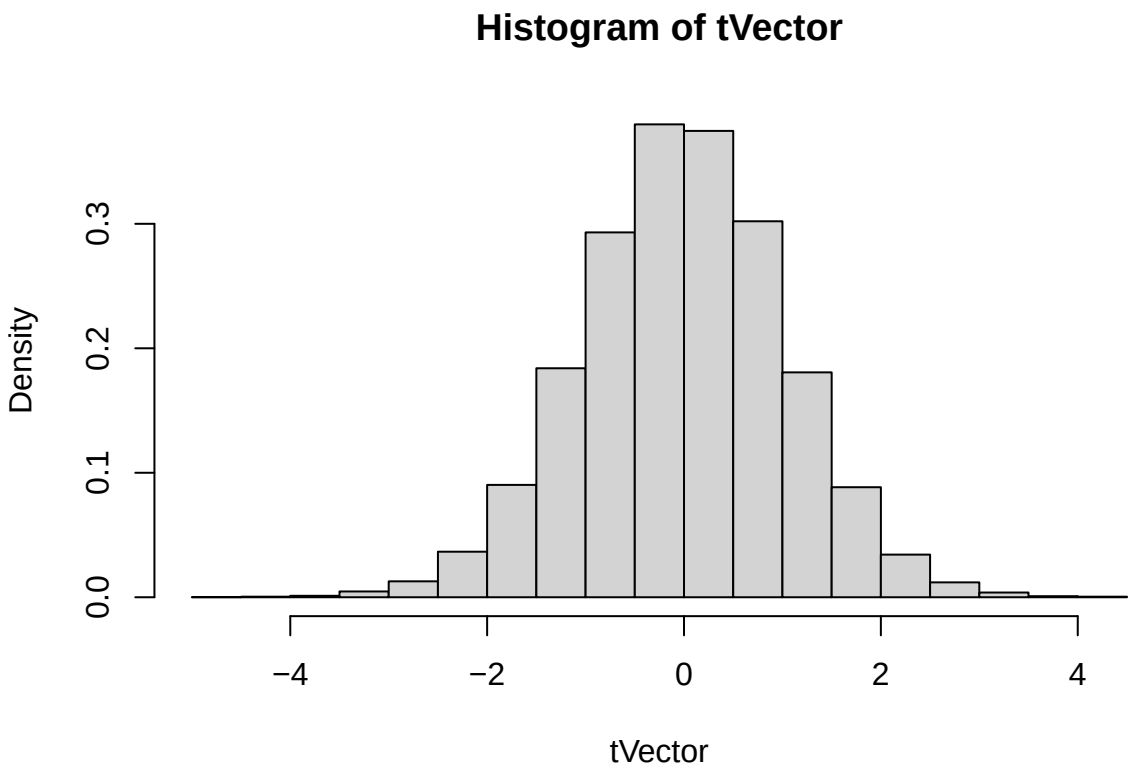
```
#compute vector of t stats
tVector <- my.compute.tVector(sampleMatrix=my.samMatrix, mu0=2)
tVector
```

```
## [1] -1.8851975  0.8359817
```

2.3.1 Further example: the same function with a larger number of repetitions.

```
my.samMatrix <- my.genSampleMatrix(nReps=40000, nSample=30, mu=2, sigma=2)

#compute vector of t stats
tVector <- my.compute.tVector(sampleMatrix=my.samMatrix, mu0=2)
hist(tVector, probability = TRUE)
```



2.4 The decision problem

Until now, you can generate a matrix holding the samples and you can compute the t-statistic for each sample in the matrix. Now write a function `my.computeTestDecisions` that carries out the decision problem. It takes the matrix holding the samples as an input, computes the t-statistics and checks whether the t-statistics are larger than a critical value.

The arguments are:

1) `my.samMatrix`: the matrix containing the samples.
2) `threshold`: the critical value. After choosing a (significance) level of α we can use a table showing the t-distribution to read off this value.
3) `mu0`: the value according to the Null hypothesis.
4) `decision.vect`: an indicator (Boolean) variable. The default value is `FALSE` which means that `my.computeTestDecisions()` should return the fraction of samples where the t-statistic is larger than the critical value. For `decision.vect` equal to `TRUE` the function should return the vector of decisions (a vector of Booleans indicating whether the t-statistic is larger than the critical value.)

```
#compute rejection rate / estimate the probability of rejecting the null

my.samMatrix <- my.genSampleMatrix(nReps=2, nSample=20, mu=2, sigma=3)

#use a significance level of 10 percent. recall this is one-sided testing.
rejectValue <- qt(0.9,df=19)
rejectValue
```

```
## [1] 1.327728
```

```
reject <- my.computeTestDecisions(my.samMatrix, threshold = rejectValue, mu0=2, decision.vect=TRUE)
reject
```

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

2.4.1 Further example: the same function with a larger number of repetitions.

This time we do not specify the argument `decision.vect`. Hence it takes the default value of `FALSE` and returns the fraction of samples where the t-statistic is larger than the critical value. We know that this fraction should be equal to our (significance) level α if the hypothesized μ_0 is equal to the true μ_X. By increasing the number of repetitions (`nReps`) we may improve the precision of the simulation.

```
my.samMatrix <- my.genSampleMatrix(nReps=20000, nSample=20, mu=2, sigma=3)
reject <- my.computeTestDecisions(my.samMatrix, threshold = rejectValue, mu0=2)
reject
```

```
## [1] 0.09975
```

2.5 Putting it all together.

Now a write a wrapper function that computes the fraction of samples where the t-statistic is larger than the critical value for *different values* of μ, σ etc. The idea is that we use only this wrapper function to study the properties of the t-test. In particular, we would like to estimate by simulation the probability to reject the null hypothesis (the power of a test) for various values of the unknown population expectation μ.

The arguments are:

1) `mu`: (see above) the true expected value of the population
2) `nReps`: see above
3) `nSample`: see above
4) `threshold`: see above
5) `sigma`: (see above) the true σ of the population
6) `mu0`: (see above) the value corresponding to the null hypothesis H_0

```
#now use the powervalue function to compute the probability of rejecting the null for various values

#same as above. H_0 is true. Result should be equal to significance level
#the precision can be increased by choosing a large nReps-value
my.powerValue(nReps=10000,nSample=20,threshold = rejectValue, mu=2,sigma=2, mu0=2)
```

```
## [1] 0.1019
```

```
#now H_0 is false
my.powerValue(nReps=10000,nSample=20,threshold = rejectValue, mu=2.4,sigma=2, mu0=2)
```

```
## [1] 0.3452
```

```
#now H_0 is false
my.powerValue(nReps=10000,nSample=20,threshold = rejectValue, mu=2.6,sigma=2, mu0=2)
```

```
## [1] 0.5071
```

2.5.1 Plotting

Now we are done. We can compute the power of the t-test for various values of μ etc.

Use your wrapper function to simulate the power of the t-test for the following problem: $H_0 : \mu_X \leq 2$ against $H_1 : \mu_X > \mu_0 = 2$ with sample size $n = 20$, $\sigma = 2$ and significance level $\alpha = 0.1$. Number of replications is 10000.

Compute the power of the t-test $\Pi(\mu_X)$ for $\mu_X = 2.0, 2.1, 2.2, ...4.0$ and plot the results:

```
plot(y=powerVect,x=muVect,xlab=expression(mu[X]),ylab=expression(Pi(mu[X]) ), type="b" )
```