Experimental Design and Data Analysis: Assignment 1

This assignment consists of becoming acquainted with the basics of `R`, and solving 3 exercises. You do not have to report on the first part, but submit solutions for the exercises as a single pdf-file. This should contain the R-code you used, and answers to the questions posed (including relevant plots if needed).

1 Introduction to `R`

You can download and install a version of `R` on your computer from web-page `http://www.r-project.org`. You may also want to use `RStudio`, a useful IDE for R, downloadable from `www.rstudio.com`.

`R` can be used interactively in the `R Console window`, which will open by default when you start the program. You may prefer to work from a `script window` (choose `file -> Open script`). This can be used as text editor in which a sequence of commands, or even a full program, can be written, and run. The contents of the script can be saved (select the script window and choose `file -> Save`), and reopened later.

The default file name extension for script files (suggested by `R` when you save a script) is `.R`, but script files are just plain text files, and can be edited also in your favourite text editor. You run parts of the text in a script window by selecting the text and typing CONTROL-R or choosing `edit -> run` from the menu (after selecting the script window).

In `file -> Change dir`, visible when the `R Console window` is selected, you can also choose a default directory (= folder in your windows system). `R` will look in that directory for data files and save files to that directory.

The steps below give a quick introduction to `R`, which should be sufficient for assignment 1. Try out the commands one by one and study the results. After each command specified below, hit the *enter* key! If you don't like typing the commands, you can find them listed in the file `assign1.R`. However, run the lines one by one to see the outcomes and read the comments below!

The `R` prompt is the >-sign. If you type a command that is not syntactically complete and type *enter*, then `R` will show a + to ask for more. You can type commands over several lines by grouping them within braces { and }. You can repeat previous commands by hitting the ↑ button several times in the `R Console window`.

Simple object manipulation

```
> x=3
```

The object `x` is created and *assigned* the value 3 using the assignment symbol `=`. (`R` fanatics use `<-` instead.)

```
> x
> x==3; x==4
```

The double `==` asks whether left and right side are equal. Two commands separated by ';', are executed in "batch mode".

```
> y=c(x,6)
```

The `concatenation function c` concatenates the objects `x` and 6, and the resulting object is called `y`.

```
> y
> 1:10
> seq(0,10,length=40)
> rep(2,5)
> rep(1:2,5)
> y=c(1.4,1.6,7.5,3.1,9.5,1.0,3.8,2)
> length(y)
```

`y` is a `vector` and has a `length`.

```
> y[1]
> y[c(1,4,5)]
> y[-1]
> y<4
> y[y<4]
```

Plotting and getting help

```
> boxplot(y)
> help(boxplot)
> z=c(2.7,4.3,9.5,1.4,5.5,7.2); boxplot(y,z)
> xaxis=seq(0,10,length=30)
> xaxis
> sin(xaxis)
```

A function, such as the sine, applied to a vector creates a vector obtained by applying the function to every coordinate.

```
> plot(xaxis,sin(xaxis))
> plot(xaxis,sin(xaxis),type="l",xlab="x-axis",ylab="y-axis")
```

So `plot(x,y)` just plots the pairs of coordinates `(x[i],y[i])` as points in a coordinate plane. The `type="l"` command tells `R` to connect the points by line segments. The `type`-option can have other values as well (see help documentation on `plot`) to get different plots. For smooth curves you need a fine grid of points.

```
> xaxis=seq(0,10,length=300)
> plot(xaxis,sin(xaxis),type="l")
```

Incidentally, to save a plot you can choose `file -> save as ->` after selecting the plot window.

Arithmetic

```
> 2*3-7
> 2^3
> y^3
```

Just as with the sine function, functions are applied coordinate-wise to vectors.

```
> 4*(3:9)
```

Data handling

```
> mean(y)
```

The sample mean of the elements of the object `y` is computed.

```
> var(y)
> sum((y-mean(y))^2)/(length(y)-1)
```

To understand this command, argue 'from the inside'. `y-mean(y)` is the vector where from each component of `y` the mean of the whole vector is subtracted. Also the square is taken component-wise, so `(y-mean(y))^2` is a vector of the same length as `y`. The function `sum` sums the elements of this vector. Dividing by `length(y)-1` gives the sample variance of the elements of `y`.

```
> sort(y)
> sample(y)
> sample(y)
```

The elements of `y` are sorted in ascending and random order, respectively.

```
> median(y)
```

The median of a sample is any number such that half of the data are to the left (or equal to) this value and half of the data are to the right (or equal) to this value.

Simulating data

A *random number generator* simulates samples from standard distributions (normal, uniform, chisquare, binomial, etc.).

```
> x=rnorm(100)
```

The object `x` contains a randomly generated sample of size 100 from the standard normal distribution. Every time you re-evaluate this command, `x` will have another value.

```
> hist(x,prob=TRUE)
```

A histogram of the data contained in `x`.

```
> x=rbinom(1,30,0.5)
```

A single draw from the binomial distribution with parameters 30 and 0.5, i.e. the number of heads when you (fairly) throw a (fair) coin 30 times.

Loops

As we have seen calculations in `R` are "vectorized": a procedure is automatically applied to every coordinate of a vector. If desired, an explicit loop can be programmed with a `for` statement.

```
> for (i in 1:10) print(i)
```

Not very exciting. A more typical use is as follows.

```
> m=numeric(500)
> for (i in 1:500) m[i]=mean(rexp(25))
> hist(m)
```

You have first set up an empty numerical vector of length 500, and next filled this with the averages of 500 independent random samples of size 25 from the exponential distribution (0.5 times a chisquare with 2 degrees of freedom). The histogram looks normal. (It should, by the Central Limit Theorem.)

```
> hist(m,prob=TRUE)
> u=seq(min(m),max(m),length=100)
> lines(u,dnorm(u,mean(m),sqrt(var(m))))
```

These commands rescale the vertical axis of the histogram so that the area is 1, and add a best fitting normal density curve.

```
> hist(rexp(500))
```

This is equivalent to what you get if you replace 25 by 1 in the preceding. This histograms does not look normal!

Reading data from files

The function `read.table` is used to read data from text files. The file should contain one line for every "individual" and will typically have a first line with the names of the variables. For instance the first 6 lines of the file `LINREG1.R` are:

```
 run  VO2max
250  60.3
325  57.2
420  55.4
410  51.4
436  52.5
> data=read.table(file="LINREG1.R",header=TRUE)
> data
> dim(data)
```

The `read.table` command creates the `data.frame` with the name `data`, which has 18 rows and 2 columns. (Of course the file `LINREG1.R` was placed in the `R`-directory.)

```
> data$run
```

The dollar sign followed by a name is used to select the column with this name. An alternative is to use *subscripts*.

```
> data[,1]
> data[1:5,1]
```

Data types

Basic data types in `R` include `numeric` for numbers, `character` for letters, and `factor` for categories of nominal variables. The latter type is used for variables that encode class membership of individuals.

```
> labels=1:10
> labels
> sum(labels)
```

The vector `1:10` consists of the numbers 1 to 10 and labels is therefore of type `numeric`. If we would like to use them as labels of a categorical variable, for instance for subjects 1 to 10, then it is better to coerce it to type `factor`.

```
> labels=as.factor(labels)
> labels
> sum(labels)
```

Although the difference is not visible, internally `labels` is now of type `factor` and numerical functions do not apply to it.

2 Homework

Histograms and QQ-plots

Histograms and QQ-plots are methods to gain insight in the "distribution" of data or of a population. In this exercise we shall be interested in histograms and QQ-plots of random samples taken from a given population. Many statistical procedures assume that data follows a normal distribution. Histograms and QQ-plots give an indication of the plausibility of this assumption for a given data set.

Histograms are excellent, but tend to be unstable if applied to small samples. For instance, a histogram of a small number of randomly chosen individuals from a normal population may well look asymmetric or have outliers. QQ-plots are a better method to verify whether data can be assumed to be sampled from a given distribution.

Recall that the α-*quantile* of a population distribution is the number ξ_α such that a fraction α of the population is smaller than ξ_α.

```
> qnorm(0.95); qnorm(0.975)
```

Two quantiles of the normal distribution often used in testing theory.

Given a random sample $X_1, \dots, X_n$ of size n from a population, we may expect the ith smallest value in the sample to be close to the i/nth quantile of the population distribution, since i/n is the fraction of sample values smaller than the ith smallest. A *QQ-plot* is essentially a plot of the ordered sample values

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$$

versus the population quantiles

$$\xi_{1/n} \leq \xi_{2/n} \leq \cdots \leq \xi_{n/n}.$$

The only difference is that one uses the quantiles at $i/(n+1)$, or another slight adaptation, rather than at i/n.

A QQ-plot of a sample against its own distribution will show a straight line through the origin with slope 1, apart from some random aberrations. If the sample has been shifted to a different mean value and scaled to a different variance (e.g. a sample that is normal with some mean and variance compared to the standard normal distribution), then the QQ-plot will still show a straight line, but with a different intercept and slope.

We can make a QQ-plot relative to any population distribution, but here we shall plot against the standard normal distribution only, with the function `qqnorm`. Thus a plot using `qqnorm` of a sample from a normal distribution will show approximately a straight line, and a deviation from a line indicates that the sample was not taken from a normal population.

```
> x=rnorm(30)
> par(mfrow=c(1,2))
> hist(x)
> qqnorm(x)
```

A histogram and QQ-plot of a sample of size 30 from the normal distribution. The QQ-plot should look linear. (The command `par(mfrow=c(1,2))` instructs `R` to place the two plots in 1 row of 2 columns. It remains in force until `mfrow` is changed, for instance back to normal with `par(mfrow=c(1,1))`.)

```
> hist(10*x+3)
> qqnorm(10*x+3)
```

What has changed?

```
> x=rnorm(10)
> hist(x)
> qqnorm(x)
```

A histogram and QQ-plot of a sample of size 10 from the normal distribution. The histogram looks bad; the QQ-plot better.

```
> x=runif(100)
> hist(x)
> qqnorm(x)
```

A plot of the ordered values of a random sample from the uniform distribution relative to the quantiles of the normal distribution. (Sampling from the uniform distribution is equivalent to choosing an "arbitrary" point from the numbers between 0 and 1. This is independently repeated 100 times.)

```
> x=rchisq(30,5)
> hist(x)
> qqnorm(x)
```

The function `rchisq` samples from yet another population distribution, the *chisquare distribution*, in this case with 5 *degrees of freedom*. It suffices to know that it is far from normal, as you can see.

EXERCISE 1
If you place the file `assign1.RData` in your `R`-directory and type

```
> load(file="assign1.RData")
```

you will have data vectors `x1, x2,..., x5` in your `R` working directory. Make a histogram and a QQ-plot for each of them, and decide which ones could have been sampled from a normal distribution. Experiment by simulating some normal samples of similar sizes and looking at their QQ-plots, before you make up your mind!

Two-sample t-test

In this exercise we simulate possible outcomes of the two-sample t-test for comparing, say, heights of men and women. We use simulated data, as follows.

```
> m=30
> n=30
> mu=180
> nu=175
> sd=10
> x=rnorm(m,mu,sd)
> y=rnorm(n,nu,sd)
> t.test(x,y,var.equal=TRUE)
> t.test(x,y,var.equal=TRUE)[[3]]
```

The first 5 lines set up the parameters of the problem: the numbers of men and women sampled from the populations of all men and women (`m` and `n`), the means of the populations of all men and women (`mu` and `nu`), and the standard deviations of these populations (`sd`; for simplicity we take these to be equal for men and women). After fixing these parameters we sample (independently) from the populations and assign the heights of the sampled men and women to the vectors `x` and `y`, respectively. Finally, we perform the t-test. The `t.test` command produces a small report on the test. This includes the p-value for testing the null hypothesis that the population means of the two populations are equal. It also gives a confidence interval (= interval estimate) for the difference of the population means. The report is actually an R-object of type `list`, with the p-value as its third component. The latter number is extracted in the last line.

We shall study the p-value and the *power function* of the t-test. The latter function gives, for every given set of parameters (m, n, μ, ν, sd), the probability that the t-test rejects the null hypothesis. We can approximate it by repeatedly simulating data and computing the fraction of times that the t-test rejects the null hypothesis.

```
> B=1000
> p=numeric(B)
> for (b in 1:B) {x=rnorm(m,mu,sd)
>                  y=rnorm(n,nu,sd)
>                  p[b]=t.test(x,y,var.equal=TRUE)[[3]]}
> power=mean(p<0.05)
```

This script assumes that we reject the null hypothesis if the p-value is smaller than 0.05. Thus the test is performed at confidence level 95 %.

EXERCISE 2

1. Set `mu=nu=180`, `m=n=30` and `sd=10`. Repeat the script 1000 times, and record the 1000 p-values. How many p-values are smaller than 5%? How many are smaller than 10%? What is the distribution of the p-values (make a histogram)?
2. Set `mu=nu=180`, `m=n=30` and `sd=1`. Answer the same questions.
3. Set `mu=180, nu=175`, `m=n=30` and `sd=6`. Answer the same questions.
4. Explain your findings.

EXERCISE 3

1. Set `mu=180`, `m=n=30` and `sd=5`. Calculate the power of the t-test for every value of `nu` in the grid `seq(175,185,by=0.1)`. Plot the power as a function of `nu`.
2. Set `mu=180`, `m=n=100` and `sd=5`. Repeat the preceding exercise. Add the plot to the preceding plot.
3. Set `mu=180`, `m=n=30` and `sd=100`. Repeat the preceding exercise.
4. Explain your findings.