R Übungen (Programmierung für Naturwissenschaften)

June 2020

There are three parts to the Übungen. You have two weeks. Divide your time as you please. Parts 2 and 3 are more difficult, but there will be lots of help in the lectures and zoom sessions.
Bitte die Lösungen zu diesen Aufgaben bis zum 14.07.2020 um 18:00 Uhr an `pfn2@zbh.uni-hamburg.de` schicken.

1. Part 1 (Hello World in R)

The aim is to get a script running, fight with R syntax, make a plot and get ready for something more interesting. It is meant to be as easy as "hello world" in C lectures.

Do not ignore this part because it looks too easy. Make sure you know the syntax for the Klausur.

1.1. Setting up

In 2020 you are probably working on your own machine. Everything in this Übung should work with a basic R installation.

This means you know how to transfer a data file on to your private machine. This description assumes one is writing an R script, but you can get the same results if you use one of the graphical interfaces (Rcmdr, Rstudio, Jupyter, …).

1.2. Help

Within R, `?etwas` or `apropos('etwas')`

In general, you probably want a browser windows to be able to 'google R etwas'.

1.3. The mission

Summary:

- Write a script that can be executed with Rscript that will
  - Read a given data file with information in some columns
  - Plot a histogram of $\log_{10}$ of the distribution of values in a column
  - Put the plot output in a file
- Attach your script and the plot and mail them to the same address as for the other Übungen

1.3.1. Data

From a sequence search, one finds homologous sequences and ranks them according to similarity to the query sequence. The measure is an *e*-value (expectation value) which estimates the number of times you would see this much sequence similarity by chance. The values range from nearly zero to nearly 1. A very similar sequence has an *e*-value≈0. Some sequences are common and some are rare. We want to know if there are many close homologues or if the sequence is somewhat exotic. The values are not accurate, so we are only interested in the order of magnitude, so we work with $\log_{10}$ of the values.

There is example data gitlab (`R/uebung_simple/`)

Look at your Matrikelnummer. If it is an odd number (n mod 2 = 1) use the data file, `blast_clupea_harengus.out`. If it is an even number use the data file, use the file `blast_symphurus_orientalis.out`.

The files have come from a sequence search so they are typical of real data. There is much information that you do not want.

Have a look at the first few lines of the file (`less xxxx.out`).

- Note how many header lines should be skipped
- The third column is the only interesting one.

1.3.2. Steps

The aim is to write a script, but usually one starts working interactively (type `R` at the command line). When you are happy, save the commands in a script.

1.3.3. Read data

Get the data into a data frame. Look up the function for read.table() – either in a browser or by `?read.table`. You want to store the name of the file in a variable (`f <- '/blah/blah/xx.out'`) and then a line like `b_data <- read.table(file=f)`. You need to do a bit more.

You have to tell R how many lines should be ignored at the start. Add `skip=3` to the parameters. You also want to give reasonable names to the columns. Before reading the file, use the `c()` function to make a vector of names (like "id", "startend", "e_val"). If you call this vector `names`, you can put `col.name = names` into the arguments when reading the table. If this is confusing, it probably means you have not typed `?read.table`.

Having the data, check that it seems sensible. If your data is in "b_data", then `summary(b_data)` will give you summary statistics including the number of data points. From the linux command line, you can count the number of input lines with `wc`. This should agree with what R tells you.

1.3.4. A first plot

You can make a basic histogram with a line like `hist(b_data$e_val)`, or whatever name you chose. You will see immediately that this is not a good representation of the data. You should work with logarithms of the data, but $\log_{10}$, not natural logarithms (ln). We want 10^{-15} to become -15, not $\ln(10^{-15}) \approx -34$.

Look up the help for the log function and find how to set it to base 10.
Make a vector of the logarithms with one line by invoking the log function on the column from the data frame. Plot this data using `hist()`.

1.3.5. Nicer histogram

You want to take care of the axes and put an informative title on the plot. At this point, it becomes easier to put the commands in a file which you can edit. At first, we will just manually read the lines into R. Then will we check that we have a script that works with Rscript.

In your favourite editor, insert commands to read the data and make a histogram with defaults. If you put the commands into a file called `makehist.r`, then start R and from the command prompt, type `source ('makehist.r')`. Make sure this works and produces a histogram on your screen.

Improve the *x*-axis label. Before you histogram command, insert

```
xlab_text = expression(paste("log"["10"], italic(' e'), '-value'))
```

in your command file. In your histogram command, insert `xlab=xlab_text` into the parameters for the histogram command. From the R command line, use the `source` command to check that you have a better label.

Show that you can put an arbitrary title on the plot. Put the number of rows and your name in a variable like

```
nr <- nrow(hdata)
myname <- "andrew"
info = paste(nr, " sequences\n", myname)
```

and add this to the histogram by adding `main=info` to the list of parameters to hist. Please put your name and not "andrew" in the title.

Check that this produces a nice histogram when you plot it out.

Finally, get the output into a file. Before the histogram function add,

```
png(file="somename.png")
```

After the histogram function, add

```
dev.off()
```

This flushes the output and closes the file.
If your script is in `makehist.r`, then

```
Rscript makehist.r
```

should produce a file in png format.

1.3.6. Finishing:

Check that your script works when you run Rscript.

Check your png file. Does it have your name on the plot and the number of sequences ? If yes, then mail the script and plot as attachments to the same address as for other Übungen.

2. Part 2 Data Fitting

The aim is to take a data set of x,y points and fit them to a decaying periodic function using R's built-in, non-linear fitting routines.

The world is full of harmonic oscillators, which ring and disappear over time. This could be a piano string vibrating or the "free induction decay" in nuclear magnetic resonance spectroscopy. If we have periodic motion, we are talking about $y = \sin(x)$ where x is our time. To avoid debates over units, we will write

$$y = \sin 2\pi\omega x \quad (1)$$

Where ω is the frequency in s^{-1}. You usually cannot control when you start recording, so one has to account for a phase shift (φ) and usually write

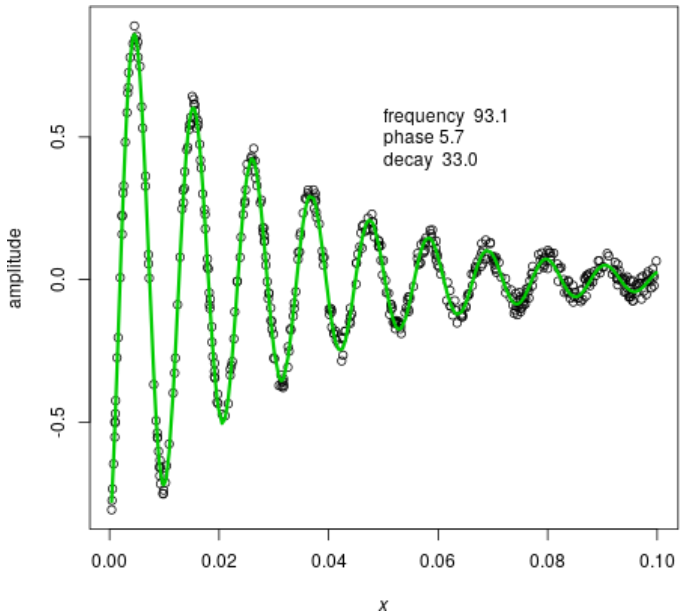
$$y = \sin(2\pi\omega x + \varphi) \quad (2)$$

Now we have to account for damping, so the final version is

$$y = e^{-\beta x} \sin(2\pi\omega x + \varphi) \quad (3)$$

Where β is a decay constant.

The job in a data fitting exercise is to read all the points and find values for ϕ, ω, and β. Given some data points, you should be able to make a plot like this one, with the data set marked by points and the fit to the data (prediction) shown by a continuous line.



2.1. Data set

Everybody should get their own data set based on their Matrikelnummer.

- If you can log in to the ZBH machine, unning the program
  `/home/torda/uebung_r2/sin_exp -o sinexp.dat 999`
  will make a few hundred data points and put them in a file called sinexp.dat. Do not use "999". Use your matrikelnummer.

- If you have a linux, windows or mac there are executables in gitlab under `R/uebung_fitting` (`linux_386/` `windows_386/` `darwin_386/`)

- If you do not want to run a binary, then you have to use a default set in gitlab in `R/uebung_fitting/default.dat`. Make a note in the assignment that you used the default values.

Run the program (sin_exp), have a quick look at the data file and in an x, y plot. The data are sorted, but not regularly spaced. There is a small amount of synthetic noise. There is a title line which should be used when reading the data.
Given your matrikelnummer, we can have the program print out your ϕ, ω, and β. This is only for checking the assigments that are sent in.

You do not know the exact parameter values until the fitting, but you have a rough idea of the ranges

ϕ	[0:2π)
ω	100 ± 10 %
β	30 ± 10 %

Use the ω and β values as starting points in the fitting.

2.2. Coding

You define a function which is passed to R's non-linear least squares routine, along with starting guesses for the parameters. It is easy to write this in less than 30 lines of simple R.

There are two parts:

2.2.1. The function for fitting

R's fitting routines want a function which acts on a vector and returns a vector of results. One should define a function whose signature might look like

```
sinexp <- function (x, phase, freq, decay).
```

`x` is a vector. The next three arguments are scalars. At the start of the function, define a vector to hold the result like `v = vector (mode='numeric')`. There are many strategies. An R enthusiast would code eq. 3 as a series of vector operations, but you can write an explicit loop like for `(a in x)` then calculate the function value for `a` and just

append to the result vector, `v = c(v, dd)`. Make sure the function ends with `return (v)`.

Your first attempt at a function will probably not work (everybody makes typing mistakes). Write a test that plots out the function. If you choose a range for x from 0 to 0.1 and the values in the table on page 7, you should get a plot similar to the one on the first page.

2.2.2. Fitting / non-linear least squares

Read the data and then try fitting:

```
d <- read.table ('/where/you/stored/the/data', header=TRUE)
```

will store a data table in `d`. Do say `header=TRUE` since the columns have names in the data file.

Check that you have a table with about 500 entries for x and y.
You can now do least squares regression with one function call, but first you should suffer for character-building reasons. Type

```
?nls
```

at an R prompt to see what you are doing. In your program, you will need a line like

```
nlmod <- nls (y~sinexp(x, phase, freq, decay), data = d,
 start = list(phase=3, freq=150, decay=30))
```

If your data is in `d` and your function was called `sinexp`, this will take the initial values from the start list and put the result into an R structure called `nlmod`. If this has worked, you can type `nlmod` at the command line and see its structure. You can get a better idea of the results by typing

```
coef(nlmod)
```

or putting this in your script. The `coef()` function knows how to get the fit coefficients from a regression object.

Finally, you should make a plot. This is most easily done in two steps.

```
plot (d$x, d$y, xlab = expression (italic(x)), ylab="amplitude")
```

will plot the x, y points from the data frame. If this is followed by

```
lines(d$x, predict(nlmod), col = 3, lwd = 3)
```

you will get a nice line joining points which are predicted by the model. The plot should look like the one page 6. To save your plot, add a line like

```
png(file='yourname_fit.png')
```

before the plot command and a `dev.off()` after the plot is finished.

2.3. Finishing

Send a mail message to the same address as for the other Übungen including

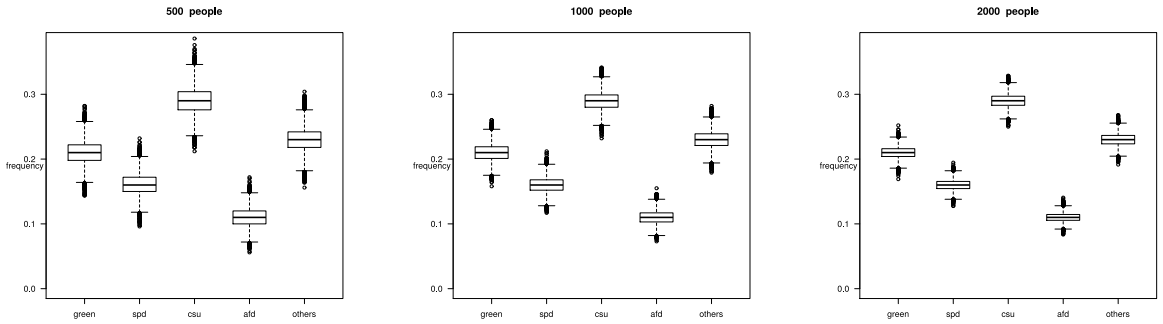
- Your matrikelnummer
- Your fit values for phase, frequency and decay rate
- An image of your plot (attachment)
- Your R-script that did the fitting (attachment)

3. Part 3 Statistical uncertainty

The aim is to use R's sampling command to estimate statistical error.

There are more than 10^7 voters in Germany, but the ZDF Politbarometer survey is based on less than 1300 people.[1] It would be better to survey more people, but it would also be more expensive. We can estimate the error with a simulation method and a small amount of R code.

You could use a plot to convey the uncertainty. On the left are the results of surveying 500 people. Each time we repeat the survey, we get a different result. Our uncertainty depends on how different the results are. The line in the middle of each box marks the median. For the greens, it is 21%. The boxes show the 25 and 75% quartiles, so half the estimates like within the box. This is also known as the interquartile range. The whiskers show 1 ½ times the interquartile range and the dots mark a few outliers. The next two plots show the same data, but after sampling 1000 and 2000 people. The median values hardly change, but the spread of data is much smaller.



At the end of the exercise, you should be able to make a plot like this, for an arbitrary n people, but a neat table would also be fine.

We use R's sampling method to generate the data based on probabilities that we invent. The greens get 21%, the AFD 11% and so on. We then simulate 500 people by drawing 500 random votes, but with the correct probabilities. R's `sample()` function does this in one step.

[1] www.forschungsgruppe.de/Umfragen/Politbarometer/Methodik/

This would give us one set of survey results, but the statistical question is different. If we repeat the survey, the results will be different. This difference determines the uncertainty. This means there will be two parameters. First, we have `n_people`, the number of people in the survey. Second, we have to repeat the survey `n_sampling` times. This should simply be a large enough number (10^5) that it gives us a good estimate of the error due to `n_people`.

3.1. Summary

Code up the following probabilities

green	spd	csu	afd	others
0.20	0.15	0.39	0.09	$1 - \sum_{i \in \text{others}} p_i$

So, "others" takes the leftover probabilities.

For 500, 1000 and 2000 people

survey 20000 times

Print results for each size of survey

Optional: make a nice plot

3.2. Overview of code

- Set up constants
  - Names of parties
  - True probabilities of parties
  - Number of samplings for a given number of people
- a function for sampling a given number of people
- perform sampling and collect median results for each party, estimate the uncertainty or plot it

3.3. Coding

You can structure the code as you want, but here are some guidelines. The code used to make the plots above and print out the results took less than 50 lines of R.

3.3.1. Settings constants

It is easiest to store the probabilities for the parties in an array. Something like

```
true_prob <- c(0.21, 0.16, 0.29, 0.11 )
```

works, but then you must write another line to set up the probability of "others". You can also store the party names in a vector, `parties <- c('green', 'spd', 'csu', 'afd', 'others')`.

3.3.2. Sampling

Read the documentation for `sample()` and think about something like

```
s <- sample (1:nparty, n, replace=TRUE, true_prob)
```

The first argument says you want a vector of size `nparty`. The next says how many times to repeat the sampling.

When we perform sampling from a collection, we can sample without replacement. Every time we take something from the collection, it disappears. Alternatively, you can sample with replacement. You draw an object, count it and put it back. In our case, the population is so big (10^7) that our sampling has no effect, so we say, `replace=TRUE`. The last argument is a vector of probabilities.

If you write a line like this, have a look at the contents of `s`. It should be a vector of length `n_people`. Each entry is how a person voted. You have to take this vector and store the number of people who voted green, spd, …

3.3.3. Repeating the sampling

You can do one survey using the steps above. Now write a loop to sample 20 000 times. Put all the results into a matrix or data frame. The easiest approach is to use a line like

```
ms <- replicate (n_sampling, onesamp(n_people, true_prob))
```

where `onesamp()` is a function you have written to do one survey over `n_people`. The `replicate()` call will call your function `n_sample` times and append the results to a matrix, which happens to be called `ms` here.

If you have got this far, you have all the information you need. Imagine you have build a matrix where each row has the results for a party. You can give names to the rows, like, `rownames(ms) <- parties`. This would let you get all the result for a party in a structure like `ms['name',]`. If you would like to get the 95 % confidence limits, you would take this vector and write something like,

```
cat(quantile(ms['name',], c(0.025)), quantile(ms['name',],c(0.975)))
```

This gives you the data between 2.5% and 97.5%. In other words, you can be sure (95%

confidence) that the real result lies here. In a nice presentation, we have a result that looks like,

```
Data collected from  500  people and with  20000  resampling
 party expcted median     (95% limits)
 green  0.210   0.210    0.174 - 0.246
   spd  0.160   0.160    0.128 - 0.194
   csu  0.290   0.290    0.250 - 0.330
   afd  0.110   0.110    0.082 - 0.138
others  0.230   0.230    0.194 - 0.266
```

It says clearly that with a sample of 500 people, we estimate the green vote to be between 17.4 % and 24.6% with 95% certainty.

You must repeat the calculation for 500, 1000 and 2000 people.

3.4. Finishing

Send a mail message to the same address as for the other Übungen including

- Your name and matrikelnummer
- Your R-script that did the fitting (attachment)
- The output in a form that resembles the table in the box above.