R Übung 1 (Programmierung für Naturwissenschaften)

July 2018

There is only time for one R Übung in 2018. You have the choice of two options

1. Do the equivalent of "hello world" in R – this is the first few pages of this file. This will get you the points for the Übung, but will leave you intellectually unsatisfied and full of guilt for having taken an easy route.
2. Read through the first few pages, then go to page 5 and see how easy it is to do non-linear least squares regression to an arbitrary function in R.

1. Introduction

The aim is to get a script running, fight with R syntax, make a plot and get ready for something more interesting in the second week. If you are very fluent in editing and scripting, you will finish quickly and can watch the world cup football.

2. Setting up

Adapt this as you please. If you have to install R packages, you probably want them in a sensible place. Do whatever you want on a private machine. On the machines in the uni, set up a file in your home directory `.Renviron` which contains a line like

```
R_LIBS=~/R
```

You can work on your own laptop, as long as you can produce the required plots and results. 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, …). Rcmdr runs on the machines in the uni.

3. Help

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

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

4. 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

4.1. 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 must work with $\log_{10}$ of the values.

There is raw data in `/home/torda/uebung_r1/data`. 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 it is 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.dat`).

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

4.2. 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.

4.2.1. 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. This is still not finished. 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 after accounting for skipped lines.

4.2.2. A first plot

You can make a basic histogram with a line like `hist(b_data$e_val)`, or whatever column name you chose above. 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()`.

4.2.3. 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 just flushes the output.
If your script is in `makehist.r`, then

```
Rscript makehist.r
```

should produce a file in png format.

4.2.4. 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 and go and watch the next football game.

R Übung 2 (Programmierung für Naturwissenschaften)

July 2018

1. Introduction

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" that nuclear magnetic resonance spectroscopists like to record. 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

$$y = \sin 2\pi\omega x \tag{1}$$

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

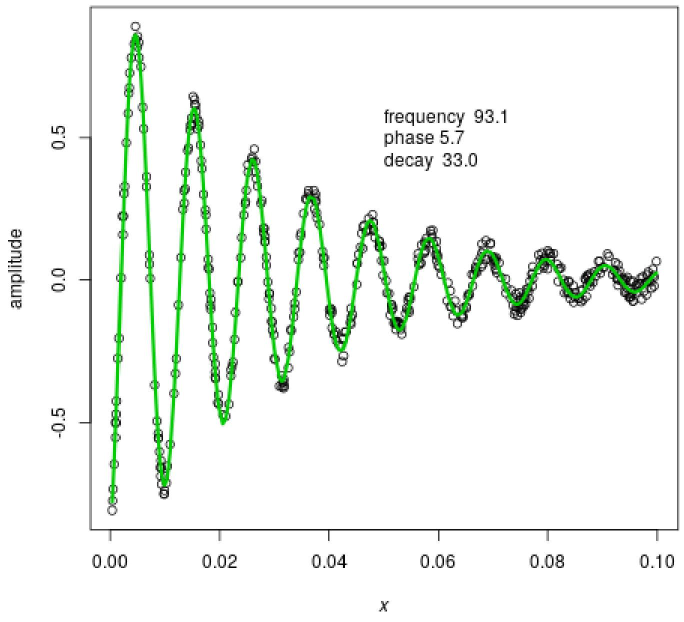
$$y = \sin(2\pi\omega x + \varphi) \tag{2}$$

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

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

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. Data set

You get your own dataset based on your Matrikelnummer. Running 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.

Run the program, 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	$[0{:}2\pi)$
ω	100 ± 10 %
β	30 ± 10 %

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

3. 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:

3.1. 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. It is probably easier to 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. 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 6, you should get a plot similar to the one on the first page.

3.2. 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 5. 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.

4. 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)