Problem Set 5

Andreas Bender, Philipp Kopper, Sven Lorenz

30 November 2021

Resources

Read chapter 21 in R4DS. You can skip the exercises.

Keyboard Shortcuts

Try incorporating the following shortcuts into your workflow for this weeks exercise sheet:

Shortcut	What does it do?
`CTRl + Shift + N`	opens new R-Script file
`F1`	Go to function help page (cursor over function)
`F2`	Get function code in new tab
`CTRL + Shift + F`	Search for keyword in all files in your directory

Exercise 1: Mean Function

In problemset 4, problem 3 you have written a function that calculates the mean (potentially accounting for missing values).

a) Rewrite the `my_mean` function without using the `sum` function. Use a `for` loop instead to determine the required quantities.

b) Compare the runtime of your function from a) with the `mean` function available directly from `R`. To do so, apply both function to the vector generated below (a random sample from the standard normal distribution) 10000 times and track the time it takes to do so. *Hint*: `replicate` and `system.time` (as illustrated below, `system.time` will track how long it takes the code within the curly brackets to execute.).

```
# sample 100 observations from standard normal
x <- rnorm(100)
# track time it takes to calculate the mean of the vector (once)
system.time(
  {
    mean(x)
  }
)
```

```
##    user  system elapsed
##   0.001   0.000   0.000
```

Exercise 2: Normalisation

In Statistics it can be helpful to normalise values. For example, a 2 cm difference in heights cannot be compared to a 2 kg difference in weight. However, after normalising both values they are more comparable because the resulting normalised values rather reflect the relative difference of a value from its mean.

Normalisation of a value (x) works like this:

$$x_n = \frac{(x - \bar{x})}{S_x}$$

with $\bar{x}$ being the mean of x and S_x being the standard deviation.

a) Write a function that takes a numeric vector as input and normalises it. It should return the normalised numeric vector. Name the function `normalisation`. Make sure that you check your input and that you document the function. Make also sure that missing values are appropriately handled.

b) Test your function. Load the vector `heights` from the Moodle page. Compute the mean and standard deviation outside your function and save those values. Compare these values with the mean and standard deviation of your normalised vector.

c) Use a for-loop to apply your new normalisation function to all numeric columns of the `Boston` data set in the `MASS` package. Note: Read the description of the data set to determine which columns are truly numeric.

d) Apply the same procedure to the `iris` data set.

e) Write a general function that performs the procedure from c) and d). Name it `normalise_df`. Test it using the `iris` data set.

Exercise 3: Newton's Method

In school you have learned that there is a numerical approach to root finding (dt.: "Nullstellenfindung"): Newton's method. Using Newton's method one uses the local derivative of a function to approach a root iteratively. A single iteration of Newton's method is described by:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

with x_t being the current proposed value for the root. For a refresher on Newton's method, we recommend this video.

a) We want to find the roots of the function $f(x) = 5\sin(x) + 0.001x^4 - 0.5x - 3$. For Newton's method to work we need to find the derivative for this function as well. Compute the derivate of $f(x)$ symbolically (by pen and paper). Store those functions ($f(x)$ and $f'(x)$) in `R` and give them the names `func` and `derivative`.

b) Plot the function `func` for a (sufficiently large) sequence of values between - 5 and 7. Hint: Use the `seq` function.

c) We start the root finding with Newton's method at $x_{t=0} = -2$. What is the closest root to this starting point? Argue.

d) Conduct a single iteration of Newton's method starting at $x_{t=0} = -2$. What is your new x-value, x_1? What is your $f(x_1)$ at this value of x?

e) Now write a for-loop to repeat this single iteration 100 times. Did you find a root?

f) Instead of iterating over a fixed amount of times one could rather write a while-loop that stops iterating if `func(x)` is reasonably close to zero. Apply Newton's method starting at $x_0 = -2$ using a while-loop that stops iterating if `func(x)` is less than 0.01 away from 0.

g) Wrap your solution from f) in a function. Name it `newton` and give it appropriate inputs. To make sure that your function does not run forever, automatically break the function after 1000 iterations. You should return a list that gives you the x-value and the associated value of $f(x)$. Test your function: It should match your result from f)

Session Info

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.1.2    magrittr_2.0.1    fastmap_1.1.0     cli_3.1.0.9000
##  [5] tools_4.1.2       htmltools_0.5.2   yaml_2.2.1        stringi_1.7.6
##  [9] rmarkdown_2.11    knitr_1.36        stringr_1.4.0     xfun_0.28
## [13] digest_0.6.28     rlang_0.99.0.9001 evaluate_0.14
```