Aufgabe2

Max Melchior Lang

8/4/2021

Aufgabe 2

a)

```
### piece_wise_line_estimator function
###
### This function applies the function `line_estimator` piecewise on certain intervals
### defined by the breaks argument. The goal of the piecewise calculation is a better
### fit of the model.
### Arguments:
### data: A data frame.
### x_name: A character of length 1. The name of the variable in data that stores
###         information about the x-coordinates of the points.
### y_name: A character of length 1. The name of the variable in data that stores
###         information about the y-coordinates of the points.
### Returns:
###         - A named list of length n (depending on the no. of breakpoints)
###           - For each interval the list contains:
###             - The first entry (solution) contains the intercept and slope
###               (numeric vector of length 2).
###             - The second entry (prediction) contains the prediction
###               (numeric vector of same length as rows in the data set)
###             - The third entry (error) contains the sum of squared differences
###               (numeric vector of length 1)

source("line_estimator.R")
crash_test <- readRDS("crash-test.Rds")

piece_wise_line_estimator <- function(data, x_name, y_name, breaks = NULL){
  # Input Checks
  if(!is.numeric(breaks)){
    stop("The breaks argument has to be either a numeric vector or  single number")
  }
  if((length(breaks)==1) && (breaks<2)){
    stop("If you only entered a single number for breaks, it hast to be greater than or
         equal to 2")
  }
  nrows_data <- nrow(data)
  if (nrows_data < 3L) {
    stop("Not enough observations in data to estimate line.")
  }
  if(x_name == y_name) {
```

```
    stop("x_name and y_name must be different.")
  }

  if(!(x_name %in% colnames(data))) {
    stop(paste0(x_name, " is not a variable in 'data'."))
  }
  if(!(y_name %in% colnames(data))) {
    stop(paste0(y_name, " is not a variable in 'data'."))
  }
  stopifnot(is.data.frame(data))
  stopifnot(is.character(x_name) & length(x_name) == 1L)
  stopifnot(is.character(y_name) & length(y_name) == 1L)

  x <- data[[x_name]]
  if (!is.numeric(x)){
    stop(paste0("Variable ", x_name, " is not numeric."))
    }
  y <- data[[y_name]]
    if (!is.numeric(y)){
      stop(paste0("Variable ", y_name, " is not numeric."))
    }

# Cutting the x-variable in intervals
  x_cut <- cut(data[[x_name]], breaks= breaks)

  estimator_list <- list()

  for(i in 1:length(levels(x_cut))){
    data_intervall <- data[which(x_cut %in% levels(x_cut)[i]),]
    output_intervall <- line_estimator(data= data_intervall,
                                       x_name = x_name,
                                       y_name = y_name)

    estimator_list[[i]] <- output_intervall
    names(estimator_list)[i] <- levels(x_cut)[i]
  }
  return(estimator_list)
}
```

b)

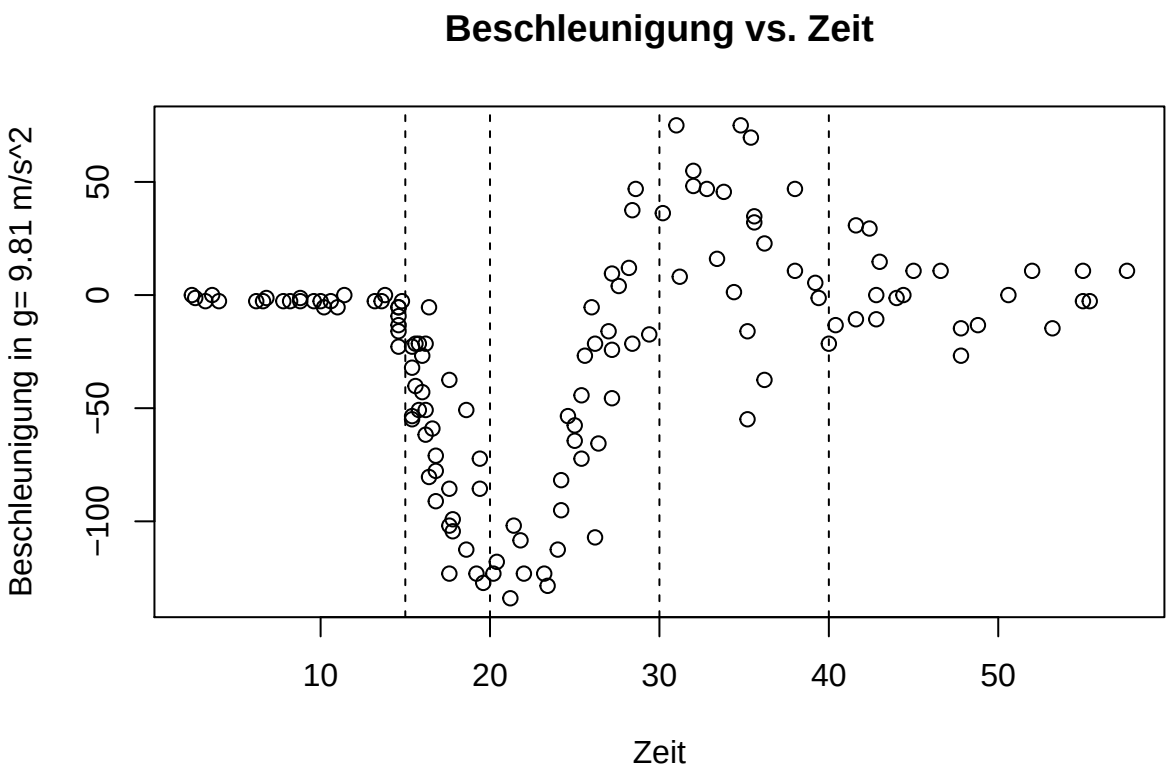
```
### plot_piecewise_line function
###
### This function plots/visualizes the output of the previous function
### `piece_wise_line_estimator`. The breaking points are visualized as dashed
### vertical lines, the model's "piecewise" estimates are presented by blue lines.
###
### Arguments:
### data: A data frame.
### x_name: A character of length 1. The name of the variable in data that stores
###         information about the x-coordinates of the points.
### y_name: A character of length 1. The name of the variable in data that stores
###         information about the y-coordinates of the points.
```

```r
### piecewise: A named list of length n (depending on the no. of breakpoints).
### Output object from `piece_wise_line_estimator`
### ...: further potential arguments passed to the plot function such as main, xlab etc.

plot_piecewise_line <- function(data, x_name, y_name, piecewise, ...){
  estimator_list <- piecewise
  # Extracting breaking points from list names with stringR
  breaks <- as.numeric(unlist(str_extract_all(as.character(names(piecewise)),
                                              "\\d+\\.*\\d*")))
  breaking_points <- breaks[seq(2, length(breaks), 2)]

  plot(data[[x_name]], data[[y_name]],...)

  for(i in 1:length(estimator_list)){
    abline(a= estimator_list[[i]][[1]][1,1], b= estimator_list[[i]][[1]][2,1],
           col= "blue")
  }
  for(j in 1:length(piecewise)-1){
    abline(v= breaking_points[j], lty= "dashed", col= "black")
  }

}
```

c)

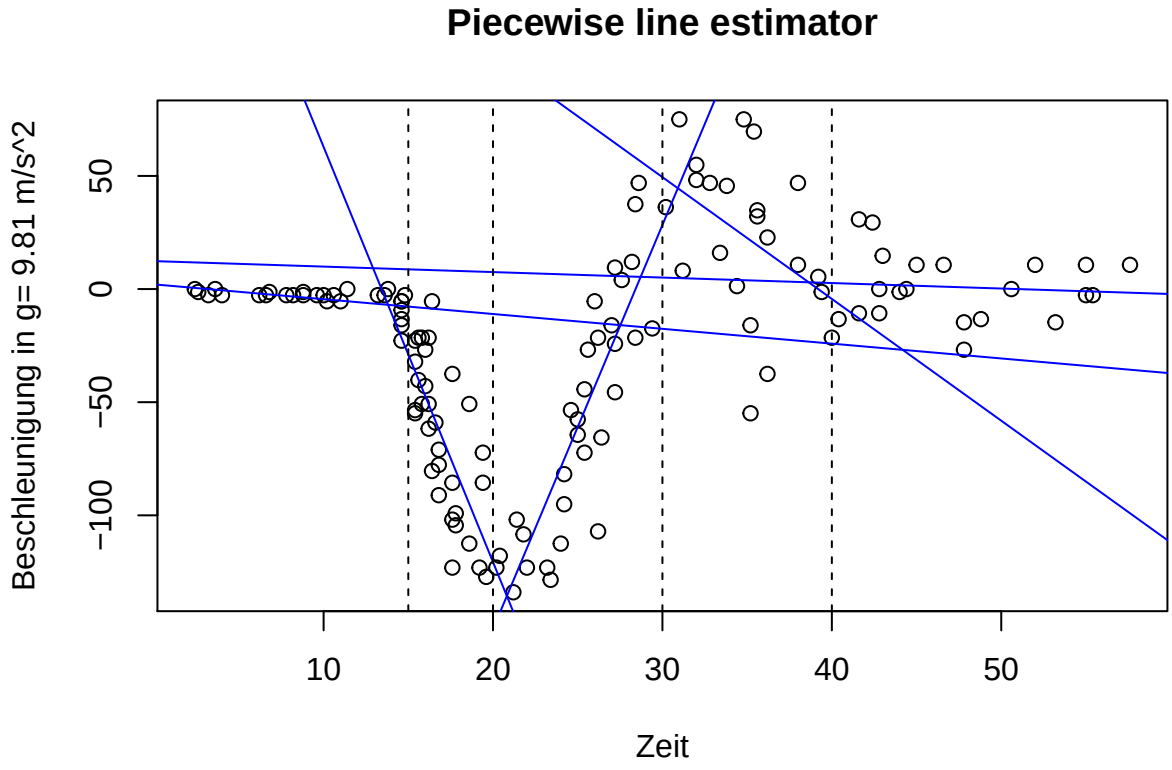
As seen in the visualization below the breaking points `c(0,15,20,30,40, Inf)` seem fairly reasonable.

```r
plot(crash_test[["zeit"]], crash_test[["beschleunigung"]], xlab= "Zeit",
     ylab= "Beschleunigung in g= 9.81 m/s^2", main= "Beschleunigung vs. Zeit")
for(j in 1:length(c(0,15,20,30,40, Inf))){
    abline(v= c(0,15,20,30,40, Inf)[j], lty= "dashed", col= "black")
  }
```



d)

```
breaks <- c(0,15,20,30,40, Inf)
piecewise <- piece_wise_line_estimator(crash_test, "zeit", "beschleunigung", breaks)
plot_piecewise_line(crash_test, "zeit", "beschleunigung", piecewise = piecewise,
                    xlab= "Zeit", ylab= "Beschleunigung in g= 9.81 m/s^2",
                    main= "Piecewise line estimator")
```



e)

```
### total_error function
###
### Given multiple (model) lines in the x/y coordinate system, this functions
### calculates
### the total error, which is the sum of errors for each piecewise fitted line.
### The error is defined as the squared difference between the line and the y values.
### Arguments:
### piecewise: A list: The out put of the `piece_wise_line_estimator`
### function.
### Returns: A numeric of length one: the computed mean.

total_error <- function(piecewise){
  if(!is.list(piecewise)){
    stop("The piecewise object has to be a list (output of
         piece_wise_line_estimator")
  }
  error_vector <- vector()
  for(i in 1:length(piecewise)){
    error_vector[i] <- piecewise[[i]][[3]]
  }
  return(sum(error_vector))
}
```

```
breaks <- c(0,15,20,30,40, Inf)

piecewise_lines <- piece_wise_line_estimator(crash_test,
                                             x_name = "zeit",
```

```
                                          y_name = "beschleunigung",
                                          breaks= breaks)
piecewise_line_error <- total_error(piecewise_lines)


global_line_error <- line_estimator(crash_test,
                                    x_name = "zeit",
                                    y_name = "beschleunigung")[[3]]

c("Piecewise Lien Error"= piecewise_line_error, "Global Line Error"= global_line_error)
```

```
## Piecewise Lien Error    Global Line Error
##             65392.77            281143.83
```

```
piecewise_line_error < global_line_error
```

```
## [1] TRUE
```

The total global error is therefore greater than the "piecewise" total error.

f) Bonus

The performance of this function is not anywhere near my standards, however it gets the job done. Slowly.

```
### optimize_breakpoints function
###
### This function finds the "best" breakpoints for the piecewise line estimator,
### such that the total error is minimized.
###
### Arguments:
### data: A data frame.
### x_name: A character of length 1. The name of the variable in data that stores
###         information about the x-coordinates of the points.
### y_name: A character of length 1. The name of the variable in data that stores
###         information about the y-coordinates of the points.
### min.Ilength: A numeric vector of length 1: The minimal length of the
### intervalls between the breaking points
### Returns: A list of length 3. The list contains the "best" breaking points, the
### minimal error and the model.

optimize_breakpoints <- function(data, x_name, y_name,min.Ilength= 6){
  possible <- as.data.frame(t(combn(data[[x_name]], 3)))
  possible <- possible %>%
    mutate(I.Length1= V1- min(data[[x_name]]),
           I.Length2= V2-V1,
           I.Length3= V3-V2,
           I.Length4= max(data[[x_name]])-V3)

  possible <- possible %>%
    dplyr::filter(I.Length1>min.Ilength) %>%
    dplyr::filter(I.Length2>min.Ilength) %>%
    dplyr::filter(I.Length3>min.Ilength) %>%
    dplyr::filter(I.Length4>min.Ilength)

  global_min <- Inf
  current_min <- vector()
```

```
  for(i in 1:(nrow(possible))){
    breaks <- c(min(data[[x_name]]),NA,NA,NA,max(data[[x_name]]))

    breaks[2] <- possible[i,][[1]]
    breaks[3] <- possible[i,][[2]]
    breaks[4] <- possible[i,][[3]]



    model <- piece_wise_line_estimator(data, x_name = x_name,
                                       y_name = y_name,
                                       breaks= breaks)



    current_min <- total_error(model)

    if(current_min < global_min){
      global_min <- current_min
      minimal_breaks <- breaks
      best_model <- model
    }
  }

output <- list("minimal_breaks"= minimal_breaks,
               "minimal_error"= global_min,
               "best_model"= model)

return(output)
}
```

Session Info

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] de_DE.UTF-8/de_DE.UTF-8/de_DE.UTF-8/C/de_DE.UTF-8/de_DE.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] forcats_0.5.1   stringr_1.4.0   dplyr_1.0.7     purrr_0.3.4
##  [5] readr_1.4.0     tidyr_1.1.3     tibble_3.1.1    ggplot2_3.3.3
```

```
##  [9] tidyverse_1.3.1 pammtools_0.5.7
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.6          lubridate_1.7.10    mvtnorm_1.1-1
##  [4] lattice_0.20-44     assertthat_0.2.1    digest_0.6.27
##  [7] foreach_1.5.1       utf8_1.2.1          R6_2.5.0
## [10] cellranger_1.1.0    backports_1.2.1     reprex_2.0.0
## [13] evaluate_0.14       highr_0.9           httr_1.4.2
## [16] pillar_1.6.0        rlang_0.4.11        lazyeval_0.2.2
## [19] readxl_1.3.1        rstudioapi_0.13     Matrix_1.3-3
## [22] checkmate_2.0.0     rmarkdown_2.8       splines_4.0.2
## [25] munsell_0.5.0       broom_0.7.6         compiler_4.0.2
## [28] numDeriv_2016.8-1.1 modelr_0.1.8        xfun_0.22
## [31] pkgconfig_2.0.3     mgcv_1.8-35         htmltools_0.5.1.1
## [34] tidyselect_1.1.1    prodlim_2019.11.13  codetools_0.2-18
## [37] fansi_0.4.2         withr_2.4.2         crayon_1.4.1
## [40] dbplyr_2.1.1        timereg_2.0.0       grid_4.0.2
## [43] nlme_3.1-152        jsonlite_1.7.2      gtable_0.3.0
## [46] lifecycle_1.0.0     DBI_1.1.1           magrittr_2.0.1
## [49] scales_1.1.1        cli_2.5.0           stringi_1.6.1
## [52] fs_1.5.0            xml2_1.3.2          ellipsis_0.3.2
## [55] generics_0.1.0      vctrs_0.3.8         Formula_1.2-4
## [58] lava_1.6.9          iterators_1.0.13    tools_4.0.2
## [61] glue_1.4.2          hms_1.0.0           pec_2020.11.17
## [64] survival_3.2-11     yaml_2.2.1          colorspace_2.0-1
## [67] rvest_1.0.0         knitr_1.33          haven_2.4.1
```