Practical Exercise 3 – Solution

Niels Richard Hansen

September 11, 2017

From previous exercises

```
myBreaks <- function(x, h = 5) {
  x <- sort(x)
  breaks <- xb <- x[1]
  k <- 1
  for(i in seq_along(x)[-1]) {
    if (k < h) {
      k <- k + 1
    } else {
      if (xb < x[i - 1] && x[i - 1] < x[i]) {
        xb <- x[i - 1]
        breaks <- c(breaks, xb)
        k <- 1
      }
    }
  }
{{  last <- length(breaks)
  if(k == min(h, length(x) - 1)) last <- last + 1
  breaks[last] <- x[length(x)] }}
  breaks
}
```

```
infrared <- read.table("../../datasets/infrared.dat", header = TRUE)
F12 <- infrared$F12
myHist <- function(h, ...)
  hist(log(F12), function(x) myBreaks(x, h), ...)
```

Problem 3.1

```
myHist <- function(h, ...) {
  tmp <- hist(log(F12), function(x) myBreaks(x, h), plot = FALSE, ...)
  class(tmp) <- "myHistogram"
  tmp
}
```

And then we try it out

```
myHist(40)
```

```
## $breaks
##  [1] -2.99573227 -1.77195684 -1.38629436 -1.20397280 -1.04982212
##  [6] -0.89159812 -0.73396918 -0.59783700 -0.46203546 -0.32850407
## [11] -0.10536052  0.03922071  0.25464222  0.60431597  7.93088943
##
## $counts
```

```
##  [1] 40 49 47 45 41 41 41 47 42 45 41 43 40 66
##
## $density
##  [1] 0.05204735 0.20231545 0.41048774 0.46484421 0.41262149 0.41417916
##  [7] 0.47958262 0.55110394 0.50084837 0.32112087 0.45155671 0.31784820
## [13] 0.18215342 0.01434443
##
## $mids
##  [1] -2.3838446 -1.5791256 -1.2951336 -1.1268975 -0.9707101 -0.8127836
##  [7] -0.6659031 -0.5299362 -0.3952698 -0.2169323 -0.0330699  0.1469315
## [13]  0.4294791  4.2676027
##
## $xname
## [1] "log(F12)"
##
## $equidist
## [1] FALSE
##
## attr(,"class")
## [1] "myHistogram"
```

Next we write the print method.

```
print.myHistogram <- function(x)
  cat(length(x$counts))
```

```
myHist(40)
```

```
## 14
```

Note that R (the graphics and base packages, to be specific) implements generic `plot`, `print` and `summary` functions. To implement a method for such generic functions, all you need is to implement a function called `print.myHistogram`, say, following the naming convention `f.classname` for the method for class `classname` for generic function `f`. Also note that you don't need to test in `print.myHistogram` whether its argument is of class myHistogram, because the method is only called for objects of this class. Finally, you will never explicitly call `print.myHistogram`, but you will call `print` with an argument of class myHistogram, and the so-called *dispatch mechanism* in R will then call `print.myHistogram`.

Problem 3.2

Note that

```
plot(myHist(40))
```

```
## Error in xy.coords(x, y, xlabel, ylabel, log): 'x' is a list, but does not have components 'x' and 'y
```

gives an error. The error message is cryptic.

One could imagine that the call should still produce a plot of the histogram, but it doesn't. Since we have modified the class label, what happens is that R does no know that it should use `plot.histogram`, and it calls `plot.default`. This function cannot find suitable `x` and `y` components and complains.

There is a very simple way of making our class "inherit" the histogram class.

```
myHist <- function(h, ...) {
  tmp <- hist(log(F12), function(x) myBreaks(x, h), plot = FALSE, ...)
  class(tmp) <- c("myHistogram", "histogram")
```

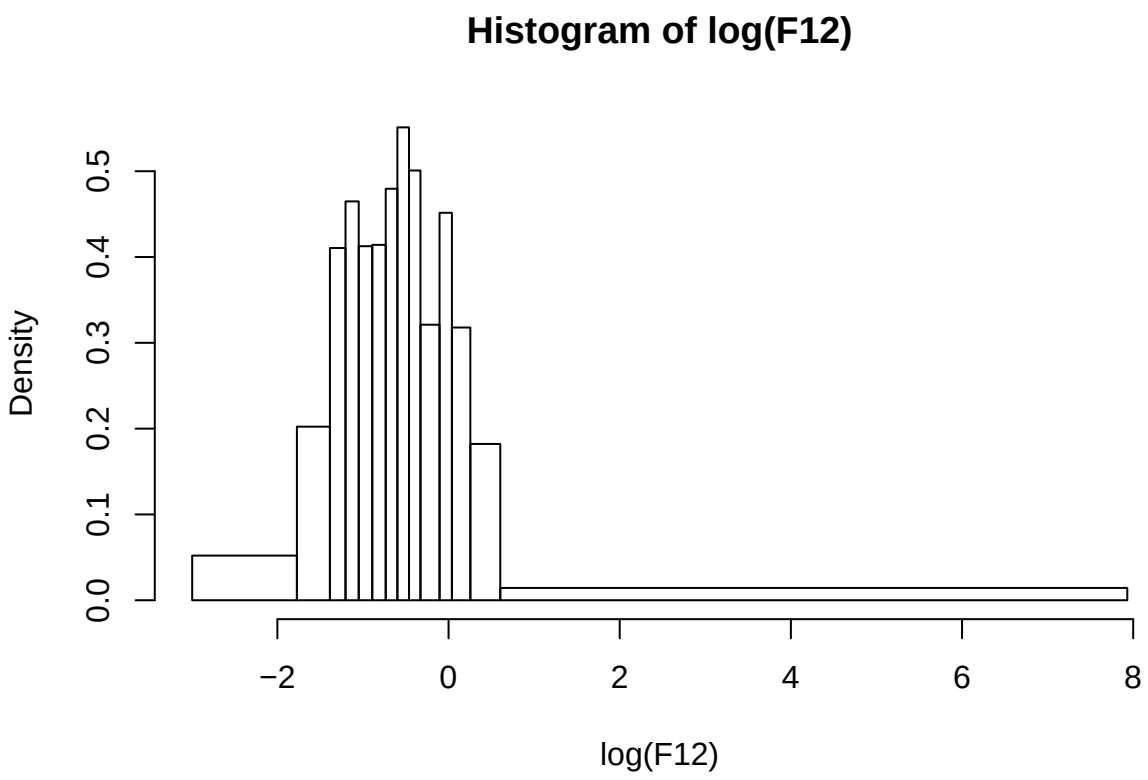
```
  tmp
}
```

And now the result is printed using our method and plotted using the method for objects of class histogram.

```
myHist(40)
```

```
## 14
```

```
plot(myHist(40))
```



Problem 3.3

```
summary.myHistogram <- function(x)
  as.data.frame(x[c("mids", "counts")])
```

```
summary(myHist(40))
```

```
##          mids counts
## 1  -2.3838446     40
## 2  -1.5791256     49
## 3  -1.2951336     47
## 4  -1.1268975     45
## 5  -0.9707101     41
## 6  -0.8127836     41
## 7  -0.6659031     41
## 8  -0.5299362     47
## 9  -0.3952698     42
## 10 -0.2169323     45
## 11 -0.0330699     41
```

```
## 12  0.1469315     43
## 13  0.4294791     40
## 14  4.2676027     66
```

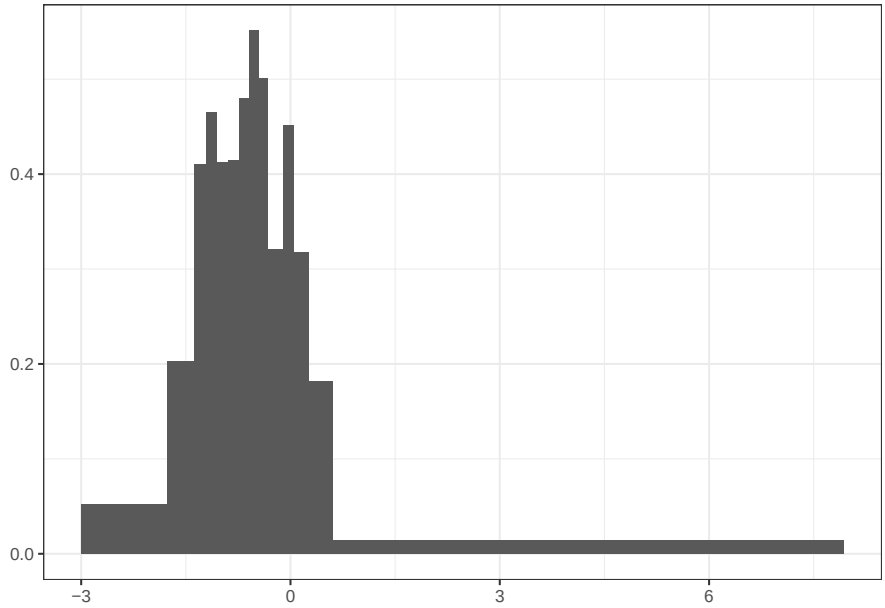
Note that in the implementation above, the entries in the list are refered to by names. This makes the implementation robust to internal changes in the number of components in the object, and is good practice. It is even better practice to use accessor functions provided by the programmer for the class. This is not widely used in R with S3 classes, but some examples include the functions `coefficients` and `residuals`, which are used together with objects of class lm or glm, say.

Problem 3.4

The plot method uses `geom_rect` from the package ggplot2 to plot the bars.

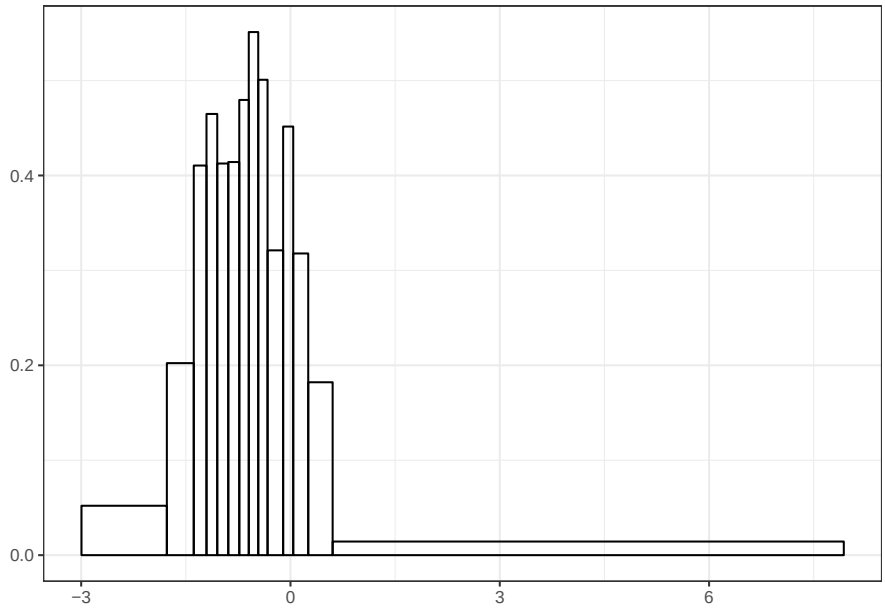
```
library(ggplot2)
plot.myHistogram <- function(x, plot = TRUE, ...) {
  tmp <- data.frame(
    breaksLeft = x$breaks[-length(x$breaks)],
    breaksRight = x$breaks[-1],
    density = x$density
  )
  p <- geom_rect(data = tmp,
                 aes(xmin = breaksLeft, xmax = breaksRight,
                     ymin = 0, ymax = density), ...)
  if (plot)
    p <- ggplot() + p
  p
}
```

```
plot(myHist(40))
```



The method implements that all additional arguments are passed on to `geom_rect`, which allows us to change the colors of the lines and the fill etc.

```
plot(myHist(40), color = "black", fill = NA)
```



We can also make the histogram semitransparent and overplot it with another one for a different value of h.

```
plot(myHist(40), fill = "red", alpha = 0.4) +
plot(myHist(20), plot = FALSE, fill = "blue", alpha = 0.4)
```