DNA copy number segmentation strikes back

Pierre Neuvial

November 27, 2015

Introduction

This lab session deals with the problem of **segmenting** DNA copy number signals. See background slides on DNA copy number in cancers.

We will start with Gaussian simulations in order to begin by focus on programming issues. Only then will we work with real or realistic data.

Setup and data generation

```
set.seed(1) # for reproducibility

xlab <- "Position on chromosome"
ylab <- "DNA copy number"
```

Generating data using Gaussian simulations

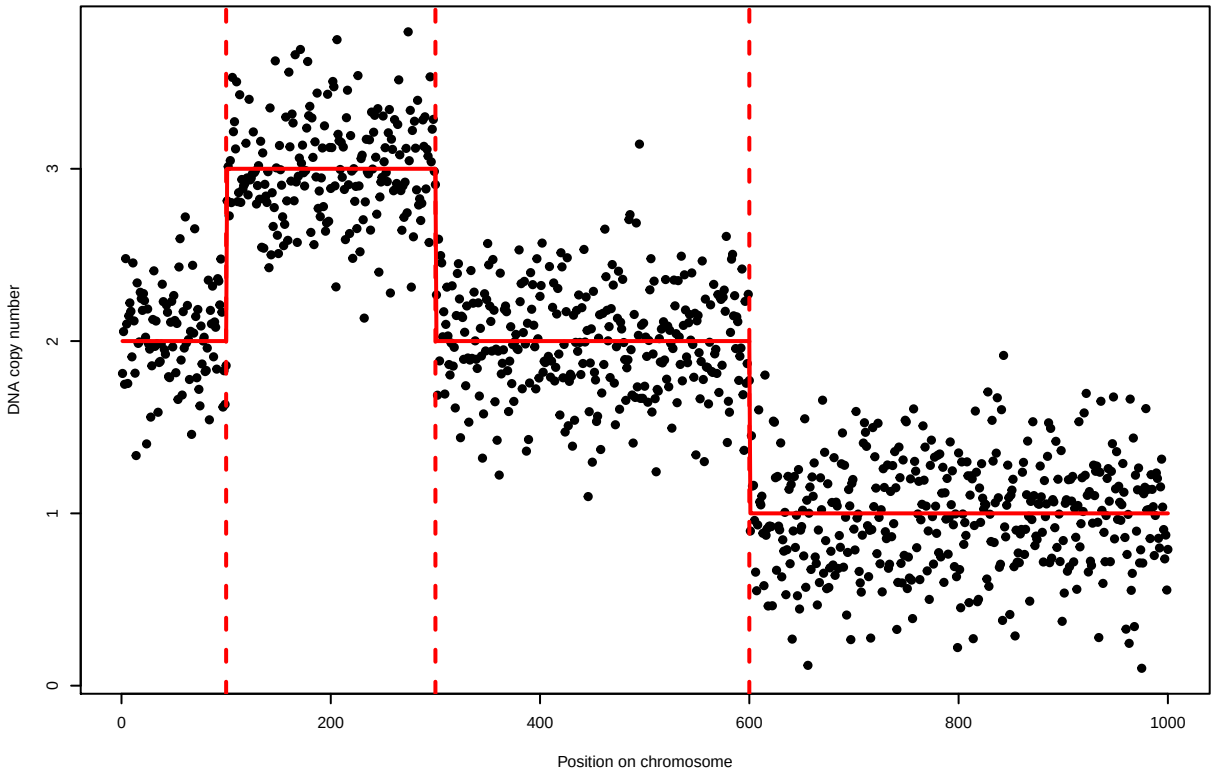
```
## truth
gamma <- rep(c(2, 3, 2, 1), times=c(100, 200, 300, 400))
## breakpoints
bkp <- which(diff(gamma)!=0)
bkp
```

```
## [1] 100 300 600
```

```
len <- length(gamma)

## signal + noise
y <- rnorm(len, mean=gamma, sd=0.3)
```

Plotting copy numbers along the genome

```
par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab)
lines(gamma, col=2, lwd=2)
abline(v=bkp, col=2, lwd=2, lty=2)
```



Naive approaches

Mixture models

- Use the 'mclust' package to fit a Gaussian mixture model

```
library("mclust")
```

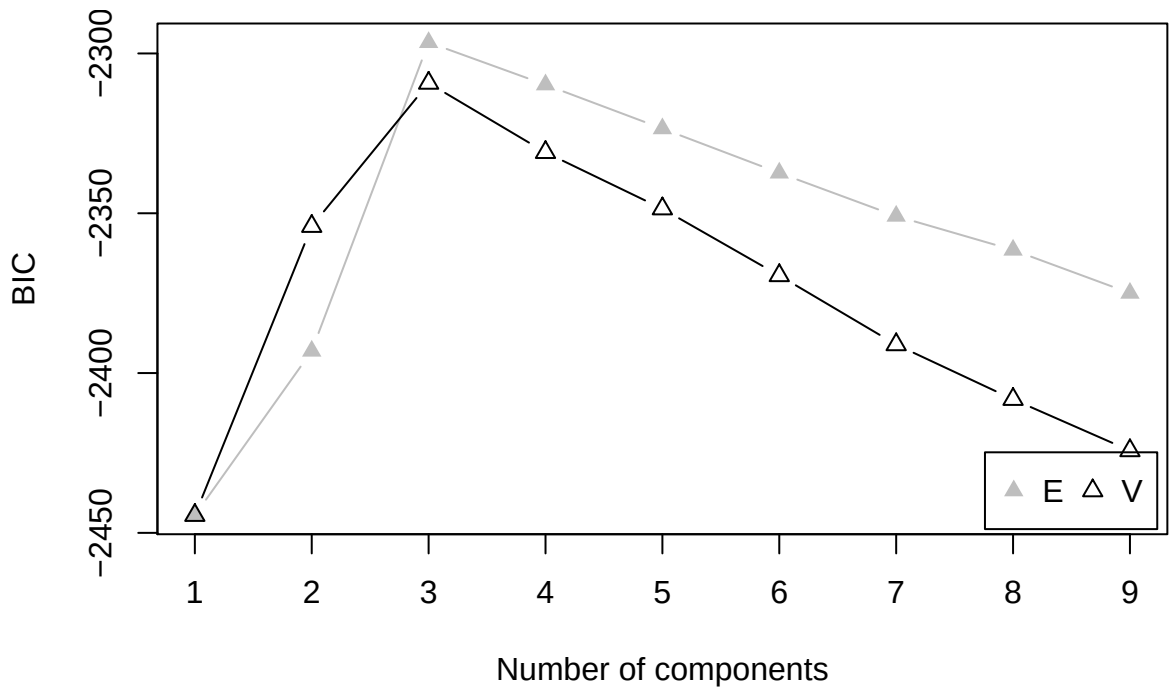
```
## Package 'mclust' version 5.1
## Type 'citation("mclust")' for citing this R package in publications.
```

```
res <- Mclust(y)
summary(res)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust E (univariate, equal variance) model with 3 components:
##
##  log.likelihood    n df       BIC       ICL
##       -1127.549 1000  6 -2296.545 -2451.651
##
## Clustering table:
##   1   2   3
## 390 412 198
```

- How many classes does the model suggest to choose and why ?

```
plot(res$BIC)
```

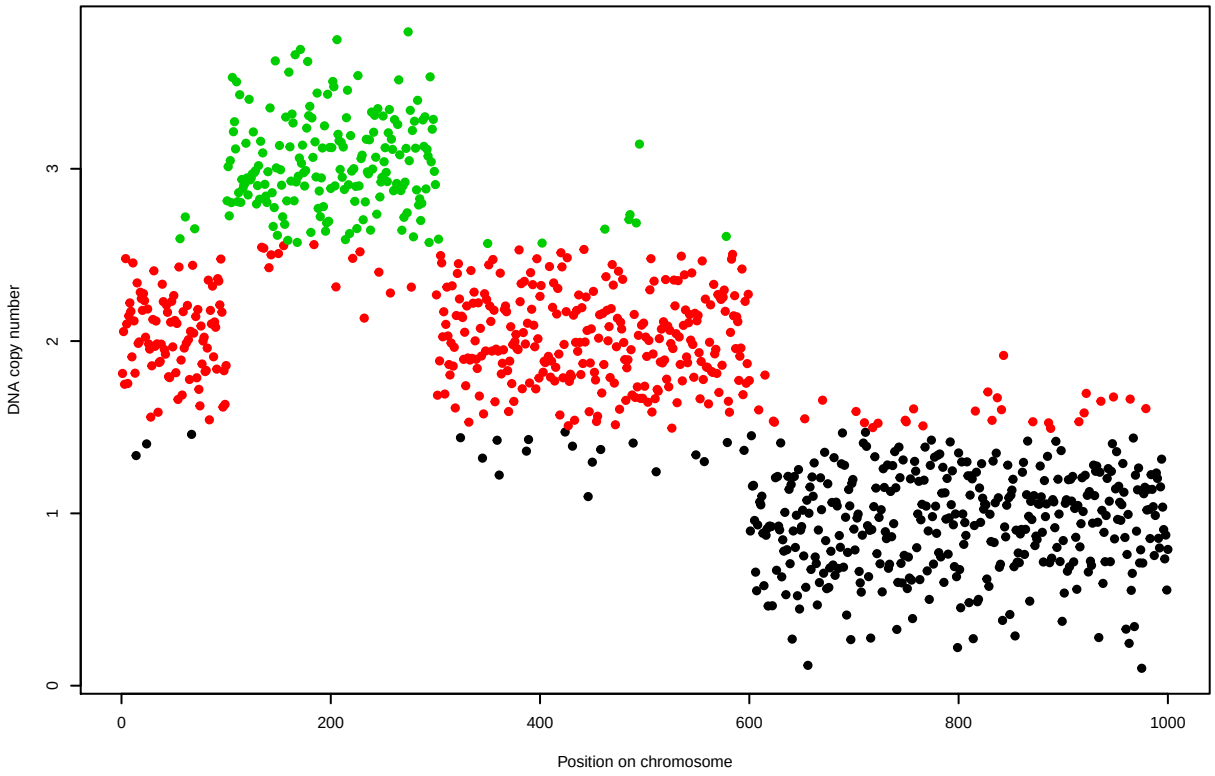


- Plot the resulting segmentation

```
cls <- res$classification
table(cls)
```

```
## cls
##   1   2   3
## 390 412 198
```

```
par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab, col=cls)
```



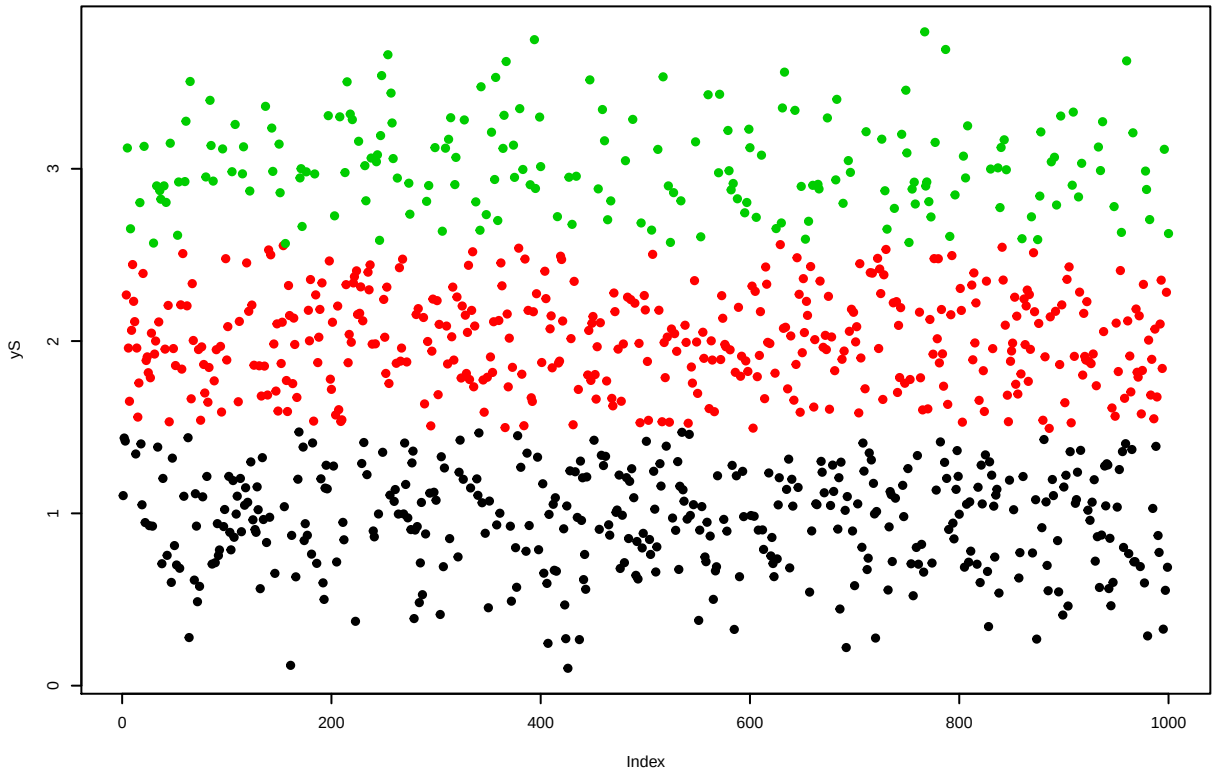
- What are the limitations of this approach?

The model does not take advantage of the piece-wise constant structure of the signal! The results of the model would be *exactly the same* if the data were shuffled (which would make no biological sense). This is illustrated by the code below, where the results of the mixture model are the same as above, whereas the original signal has no biological meaning.

```
yS <- sample(y)
resS <- Mclust(yS)
summary(resS)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust E (univariate, equal variance) model with 3 components:
##
##  log.likelihood    n df       BIC       ICL
##       -1127.549 1000  6 -2296.545 -2451.651
##
## Clustering table:
##   1   2   3
## 390 412 198
```

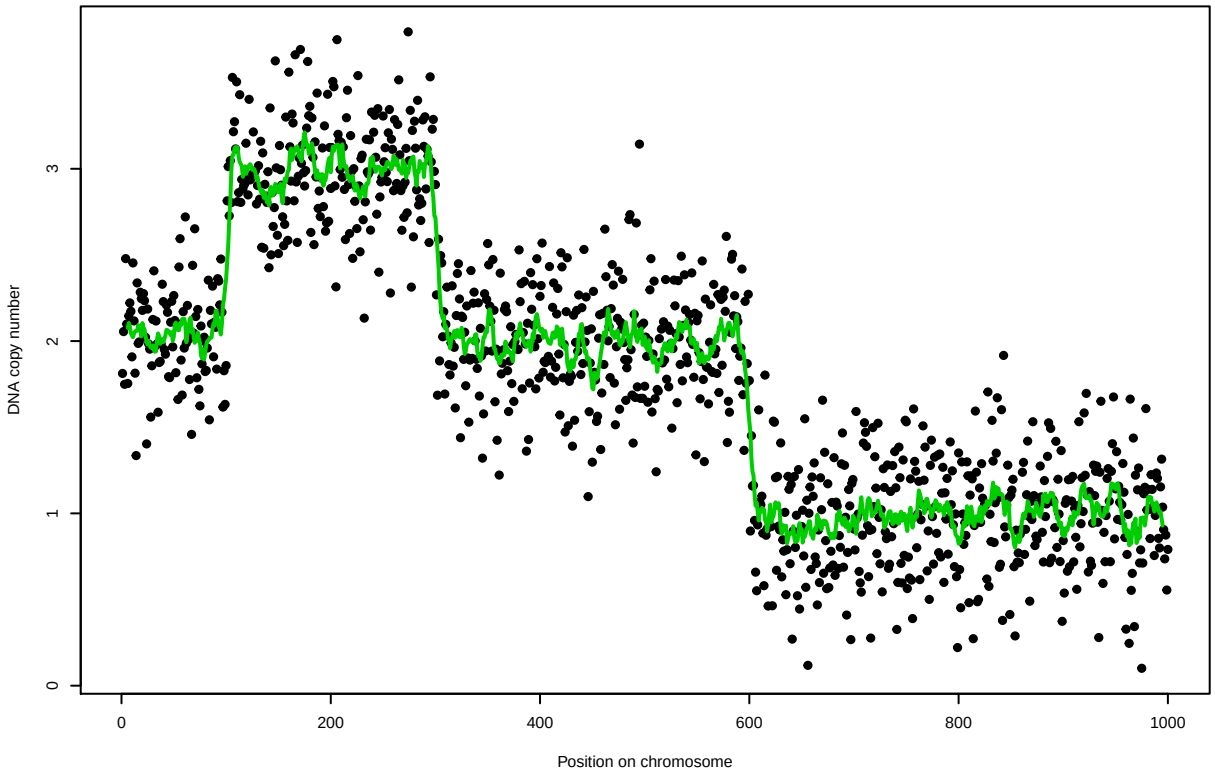
```
par(pch=19, cex=0.5)
plot(yS, col=resS$classification)
```



Moving averages

- Calculate a moving average with +/- 5 neighbors. By convention we will consider that the moving average is not defined ('NA' in R) for the 5 first and 5 last data points.

```
h <- 5
z <- rep(NA, len)

idxs <- seq(from=h+1, to=len-h)

for (ii in idxs) {
    jj <- seq(from=ii-h, to=ii+h)
    z[ii] <- mean(y[jj])
}

par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab)
lines(z, col=3, lwd=2)
```



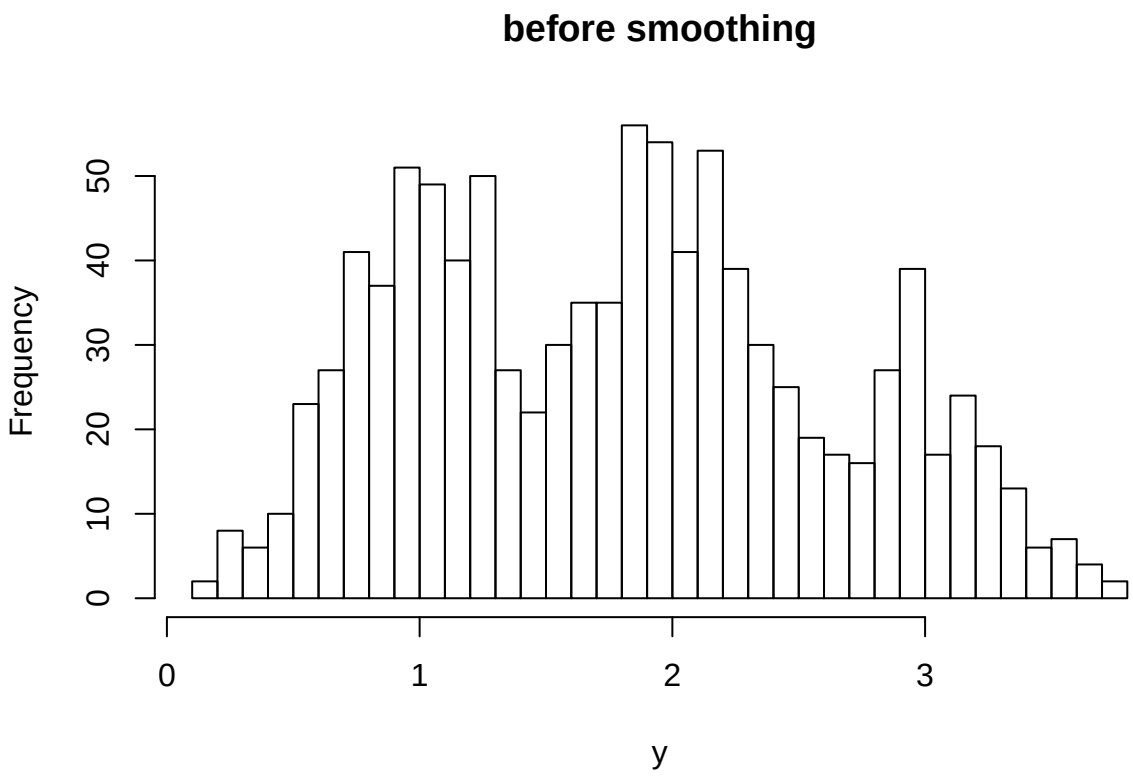
Another possible convention for the first 5 and last 5 data points:

```
h <- 5
z <- rep(NA, len)

idxs <- seq(from=1, to=len)

for (ii in idxs) {
    jj <- seq(from=max(ii-h, 1), to=min(ii+h, len))
    z[ii] <- mean(y[jj])
}
```

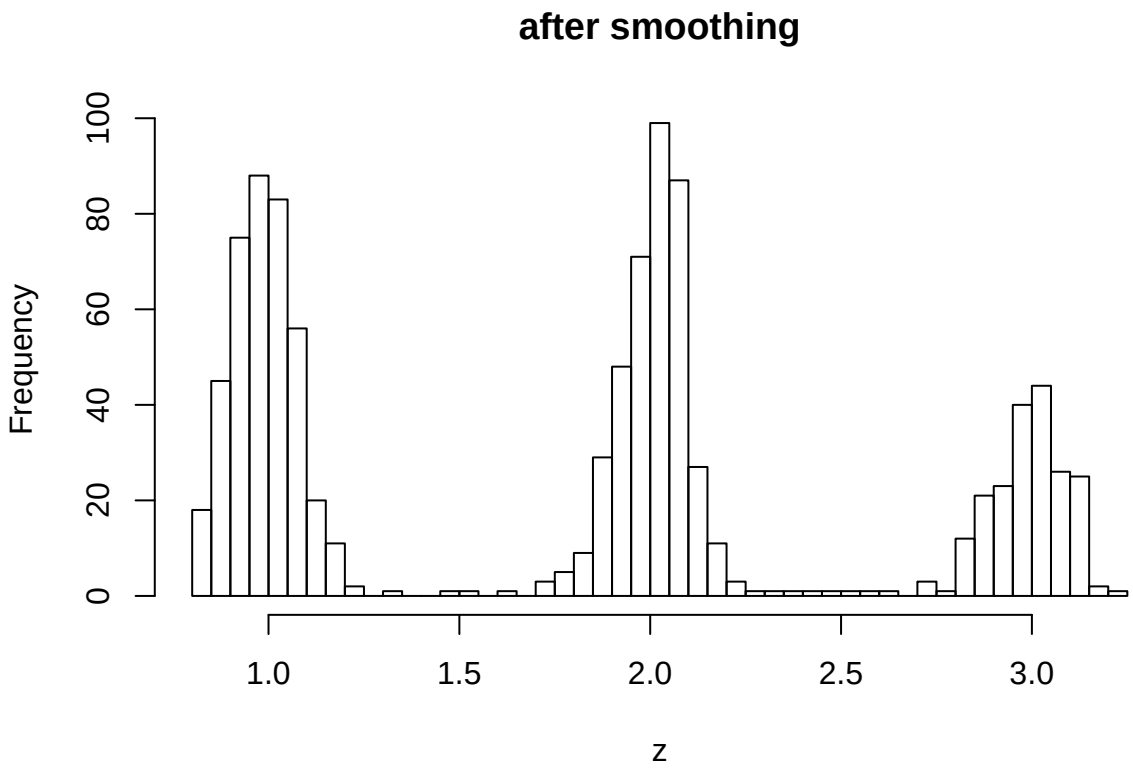
One advantage of this convention is that there are no missing values in 'z'.

- How can we use this smoothing to produce a segmentation?

```
hist(y, breaks=50, main="before smoothing")
```




```
hist(z, breaks=50, main="after smoothing")
```

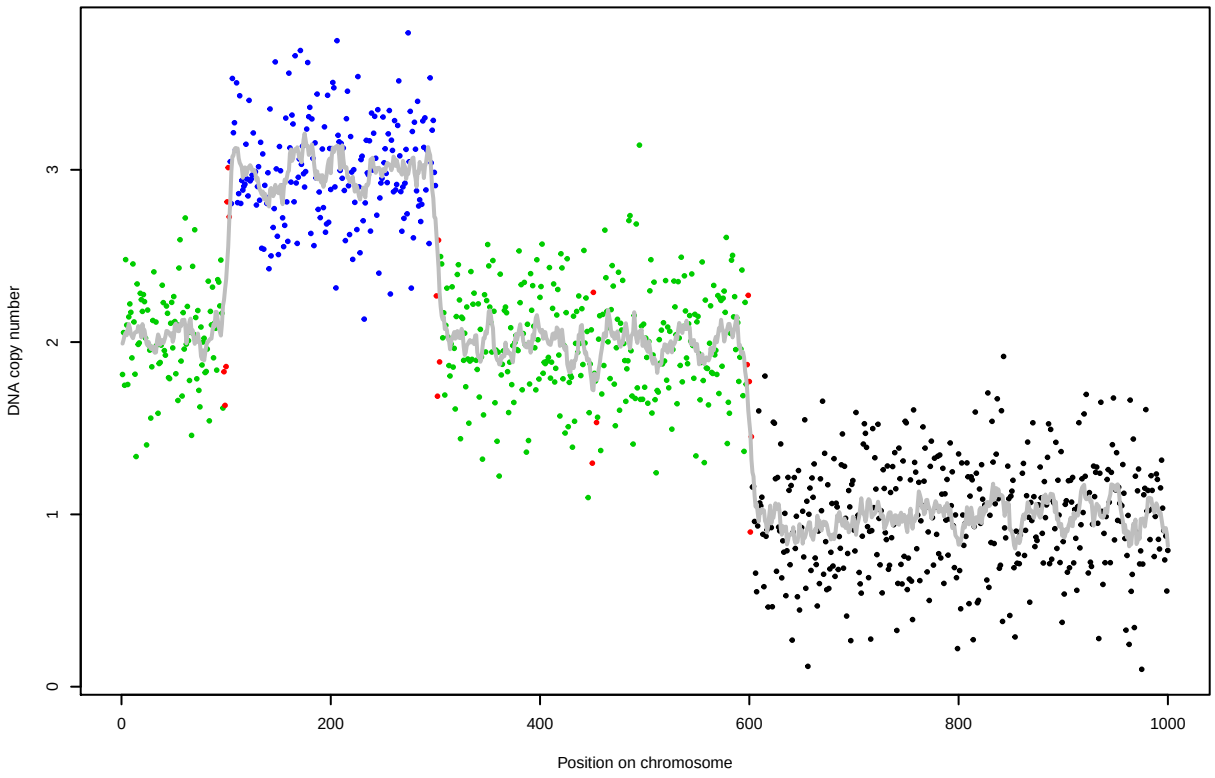


The biological states are much better separated after smoothing by moving average. One possibility is to use a Gaussian mixture model on these smoothed data.

```
res <- Mclust(z)
summary(res)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust V (univariate, unequal variance) model with 4 components:
##
##  log.likelihood    n df       BIC       ICL
##        -71.7573 1000 11 -219.4999 -262.2723
##
## Clustering table:
##   1   2   3   4
## 398  18 387 197
```

```
cls <- res$classification
par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab, col=cls, pch=19, cex=0.5)
lines(z, col="gray", lwd=2)
```



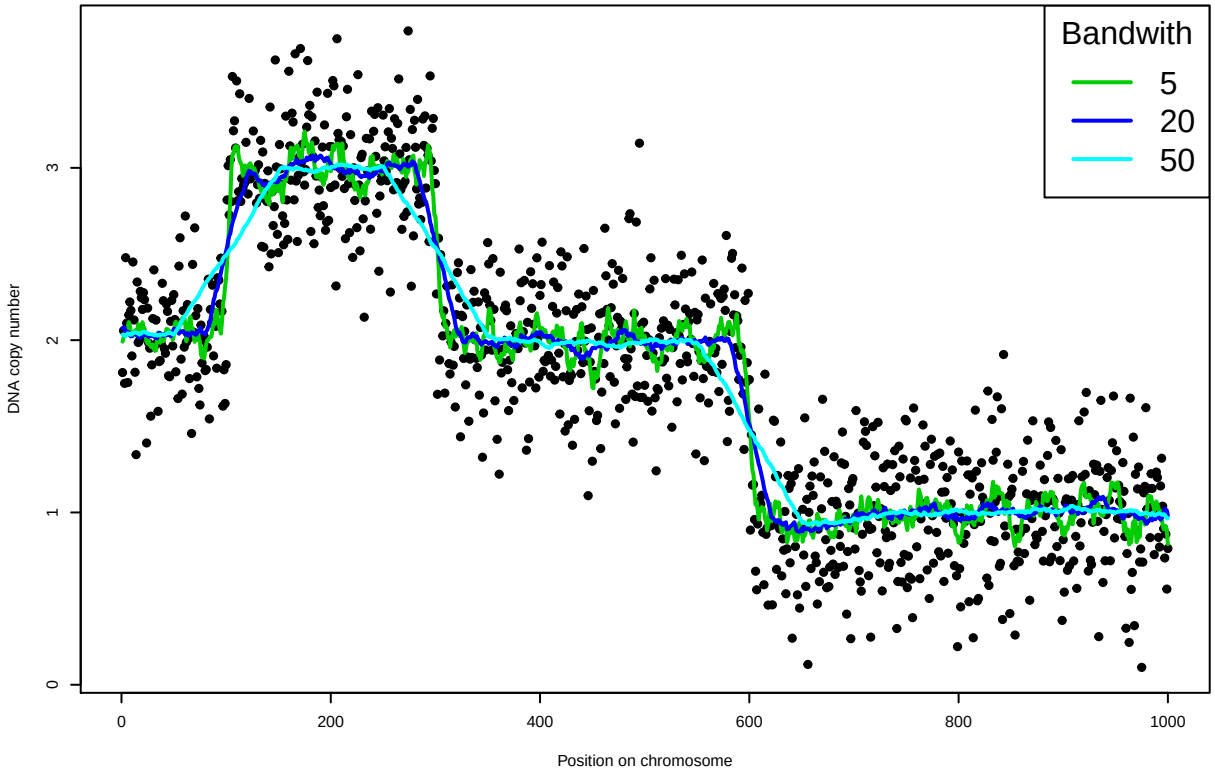
Interestingly, in this particular simulation run the model found 4 classes while the true number of biological states is 3. The 4th class (red points below) mainly consists of points close to the true breakpoints: for these points, the smoothed value is *intermediate between* two biological states. This is a (bad) side effect of the moving average.

- Illustrate the influence of the "bandwidth" parameter 'h'

In order to avoid duplicating the code for calculating a moving average, we can incorporate it into a *function*:

```
movav <- function(y, h) {
    len <- length(y)
    z <- rep(NA, len)
    idxs <- seq(from=1, to=len)

    for (ii in idxs) {
        jj <- seq(from=max(ii-h, 1), to=min(ii+h, len))
        z[ii] <- mean(y[jj])
    }
    z
}
```

```
z5 <- movav(y, 5)
z20 <- movav(y, 20)
z50 <- movav(y, 50)

par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab)
lines(z5, col=3, lwd=2)
lines(z20, col=4, lwd=2)
lines(z50, col=5, lwd=2)
legend("topright", title="Bandwith", legend=c(5, 20, 50), col=3:5, lty=1, lwd=2, cex=2)
```

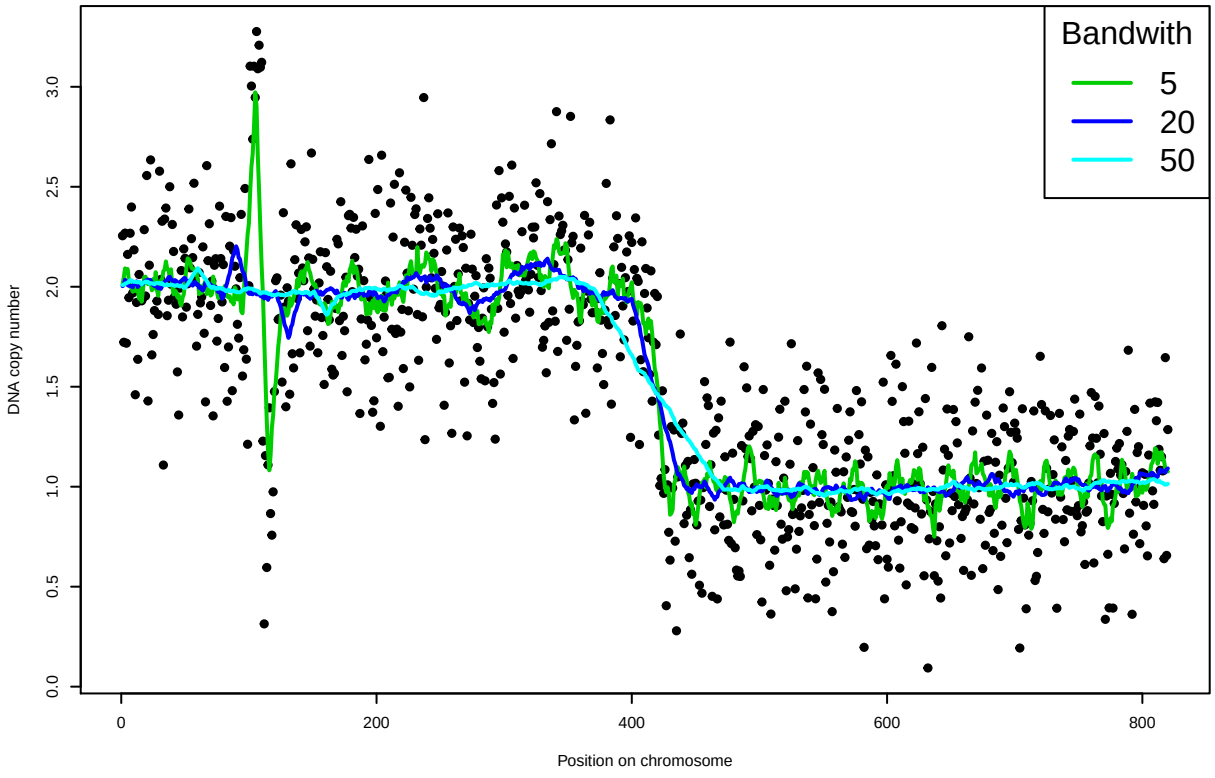


- Can you give an example of signal where the moving average approach is likely to fail?

We consider a signal with short regions:

```
## truth
gamma <- rep(c(2, 3, 1, 2, 1), times=c(100, 10, 10, 300, 400))
## breakpoints
bkp <- which(diff(gamma)!=0)

len <- length(gamma)

## signal + noise
y <- rnorm(len, mean=gamma, sd=0.3)
```

```
z5 <- movav(y, 5)
z20 <- movav(y, 20)
z50 <- movav(y, 50)

par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab)
lines(z5, col=3, lwd=2)
lines(z20, col=4, lwd=2)
lines(z50, col=5, lwd=2)
legend("topright", title="Bandwith", legend=c(5, 20, 50), col=3:5, lty=1, lwd=2, cex=2)
```

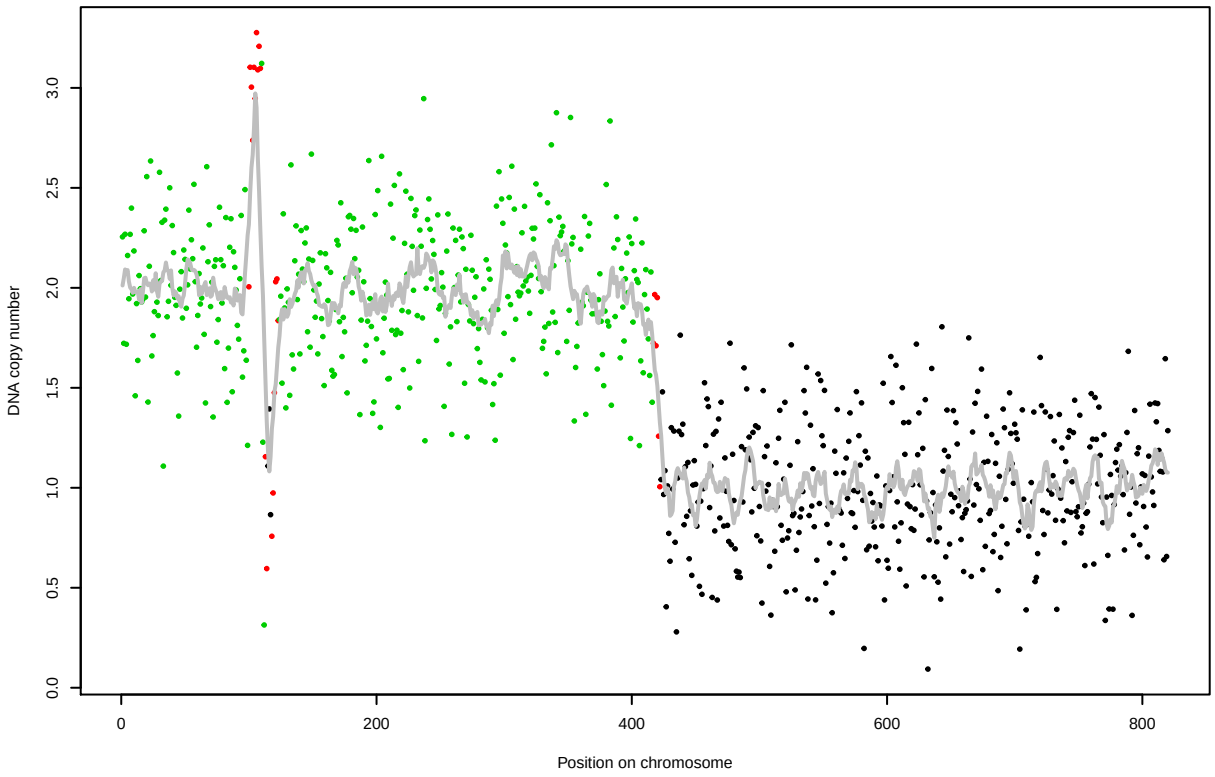



```
res <- Mclust(z5)
summary(res)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
```

```
##
## Mclust V (univariate, unequal variance) model with 3 components:
##
##  log.likelihood   n df      BIC      ICL
##        102.2468 820  8 150.8192 103.2327
##
## Clustering table:
##   1   2   3
## 401  24 395
```

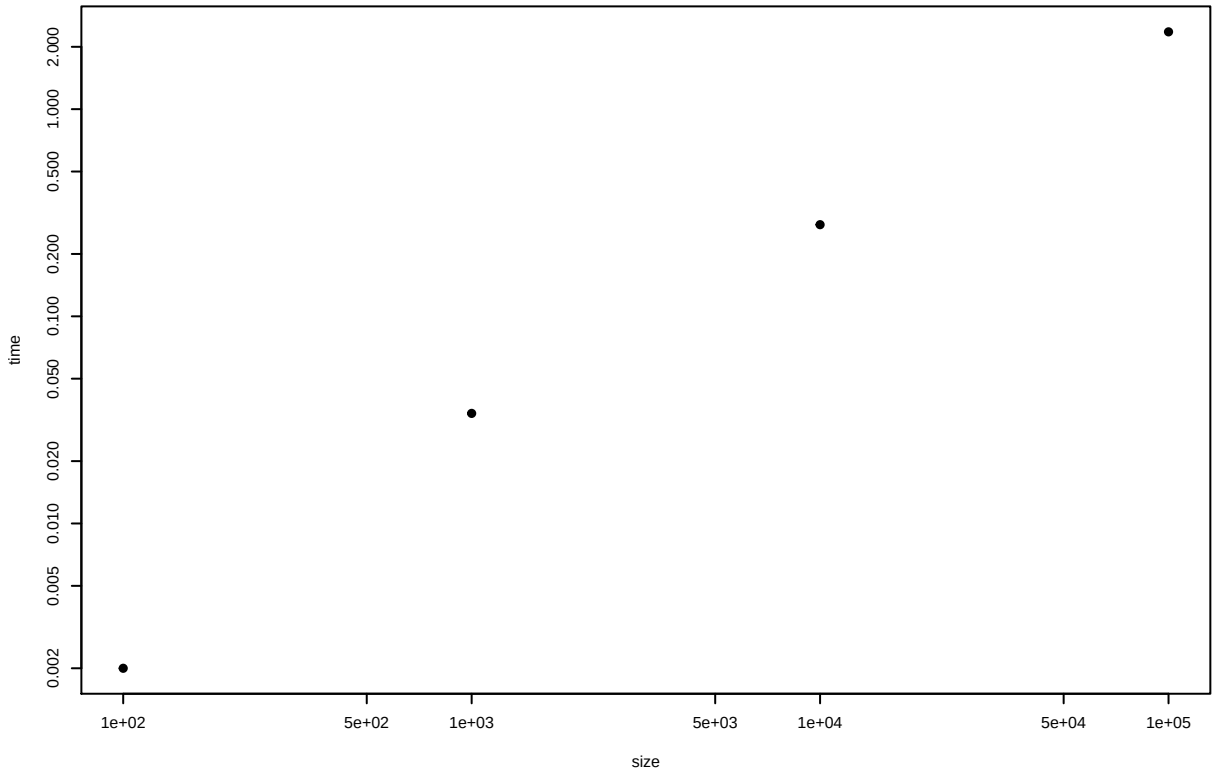
```
cls <- res$classification

par(pch=19, cex=0.5)
plot(y, xlab=xlab, ylab=ylab, col=cls, pch=19, cex=0.5)
lines(z5, col="gray", lwd=2)
```



- What is the empirical time complexity of your implementation as a function of the signal length? (and why should we care ?)

As the moving average loops over all signal points, and makes a fixed number (2h+1) of operations for each point, the complexity of this implementation is *linera* in the signal size. This can also be checked empirically:

A small function to time the segmentation algorithm:



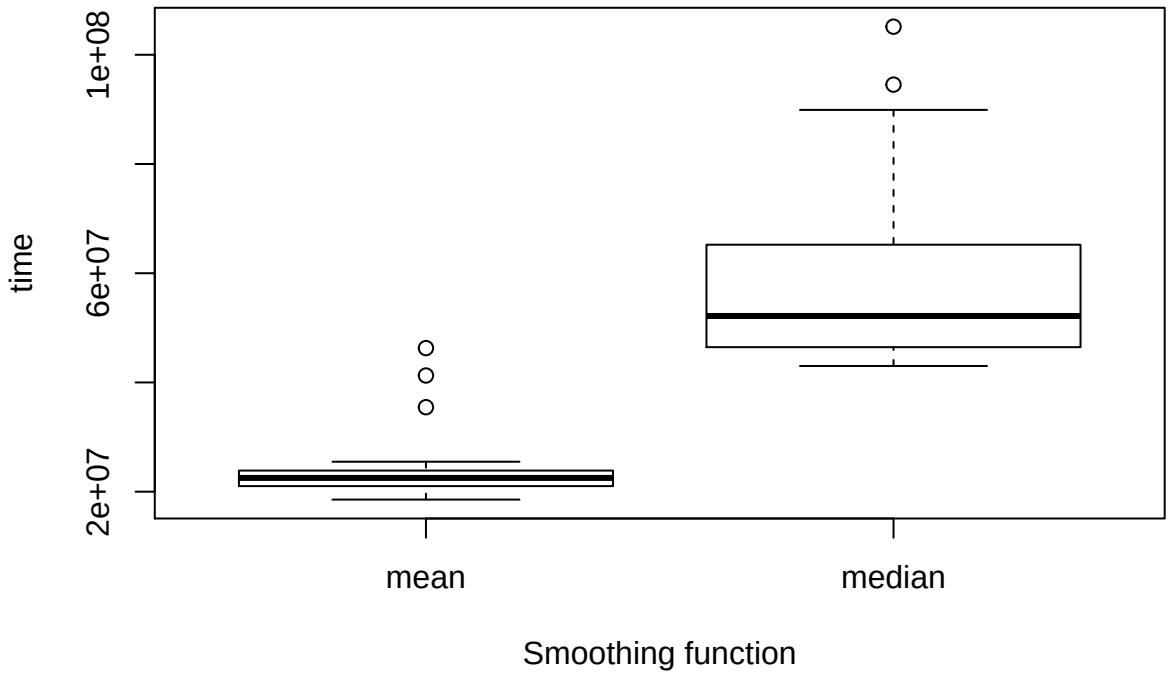
- How can we make this segmentation robust to outliers?

We can use a median instead of a mean for smoothing:

```
movav <- function(y, h, f=mean) {
    len <- length(y)
    z <- rep(NA, len)
    idxs <- seq(from=1, to=len)

    for (ii in idxs) {
        jj <- seq(from=max(ii-h, 1), to=min(ii+h, len))
        z[ii] <- f(y[jj])
    }
    z
}
```

It is slightly more computationally expensive, but still linear in the signal size: see the comparison below (where we have used the `microbenchmark` package to compare timings):

```
library("microbenchmark")
len <- 1000
h <- 10
res <- microbenchmark(movav(sim(len)$signal, h=h, f=mean),
                      movav(sim(len)$signal, h=h, f=median),
                      times=20)
levels(res$expr) <- c("mean", "median")
plot(res, xlab="Smoothing function")
```




```
sessionInfo()
```

```
## R version 3.2.0 (2015-04-16)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] microbenchmark_1.4-2 mclust_5.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.11.6      digest_0.6.8     MASS_7.3-41      grid_3.2.0
##  [5] plyr_1.8.3       gtable_0.1.2     formatR_1.2      magrittr_1.5
##  [9] scales_0.2.5     evaluate_0.7     ggplot2_1.0.1    stringi_0.5-2
## [13] reshape2_1.4.1   rmarkdown_0.7    proto_0.3-10     tools_3.2.0
## [17] stringr_1.0.0    munsell_0.4.2    yaml_2.1.13      colorspace_1.2-6
## [21] htmltools_0.2.6  knitr_1.10.5
```