Lab 02 - Depth measures for functional data

Nonparametric statistics ay 2023/2024

2023/09/26

Loading necessary libraries

```
library(roahd)
```

Functional data analysis with R: a primer

Functional Data Analysis is one of the core research areas of our department, you have already seen during class that some of the state-of-the-art techniques were actually developed here. Depth-based techniques find a natural application in Functional Statistics. I assume that many of you already know what we mean by functional data (applied statistics course *docet*), nevertheless do not worry if you do not: I have structured this lab as a very basic primer to understand how to do some methodological and applied work in the framework of functional data analysis (FDA).

According to the FDA model, data can be seen as measurements of a quantity/quantities along a given, independent and continuous indexing variable (such as time or space). Observations are then treated as random functions and can be viewed as trajectories of stochastic processes defined on a given infinite dimensional functional space.

A natural way to reason (and actually comprehend why they need specific and peculiar techniques) is to think about them as **relatively dense longitudinal data**: unlike tabular data, functional data are not invariant to permutations of their dimensions (i.e., if I switch two columns of a standard multivariate dataset no harm is done, if I do it with a functional one, I will create a bloody mess).

Given its ubiquitous presence in applications FDA has been a very active research field in the past decades, with a plethora of R packages developed for helping practitioner efficiently and effectively employ such techniques. In what follows we will focus our attention on the `roahd` (*RObust Analysis of High Dimensional Data) package (developed by members and former members of our Department of Mathematics): a package meant to collect and provide methods for the analysis of univariate and multivariate functional datasets through the use of robust methods. First off we will dedicate some time in understanding how to represent these complex infinite dimensional objects (our computers only deal with finite approximations) and how to simulate functional data. We will then focus on computation of depths and outlier detection. This lab WILL NOT provide a thorough treatment of FDA, but we will limit to study some introductory concepts on how to deal with univariate (and bivariate, only very briefly) functional datasets.

Simulating functional data with roahd package

The way `roahd` package represents functional objects is by providing an evenly spaced grid $I = [t_0, t_1, \ldots, t_{P-1}]$ ($t_j - t_{j-1} = h > 0$, $\forall j = 1, \ldots, P-1$) over which the functional observations $D_{i,j} = X_i(t_j)$ (*i.e.* the ith function evaluated at the jth point of the grid) $\forall i = 1, \ldots, N$ and $\forall j = 0, \ldots, P-1$ are measured. This is very conveniently handled by the `fData` object class. In particular, the following model is considered for the generation of data:

$$X(t) = m(t) + \varepsilon(t), \text{ for all } t \text{ in } I$$

where $m(t)$ is the mean function (deterministic) and $\varepsilon(t)$ is a centered Gaussian process with covariance function $C(\cdot,\cdot)$. That is to say:

$$\text{Cov}(\varepsilon(\text{s}),\varepsilon(t)) = C(s,t), \text{ with } s,t \text{ in } I$$

The employed structure for $C(s,t)$ is the Exponential covariance function:

$$C(s,t) = \alpha e^{-\beta|s-t|}$$

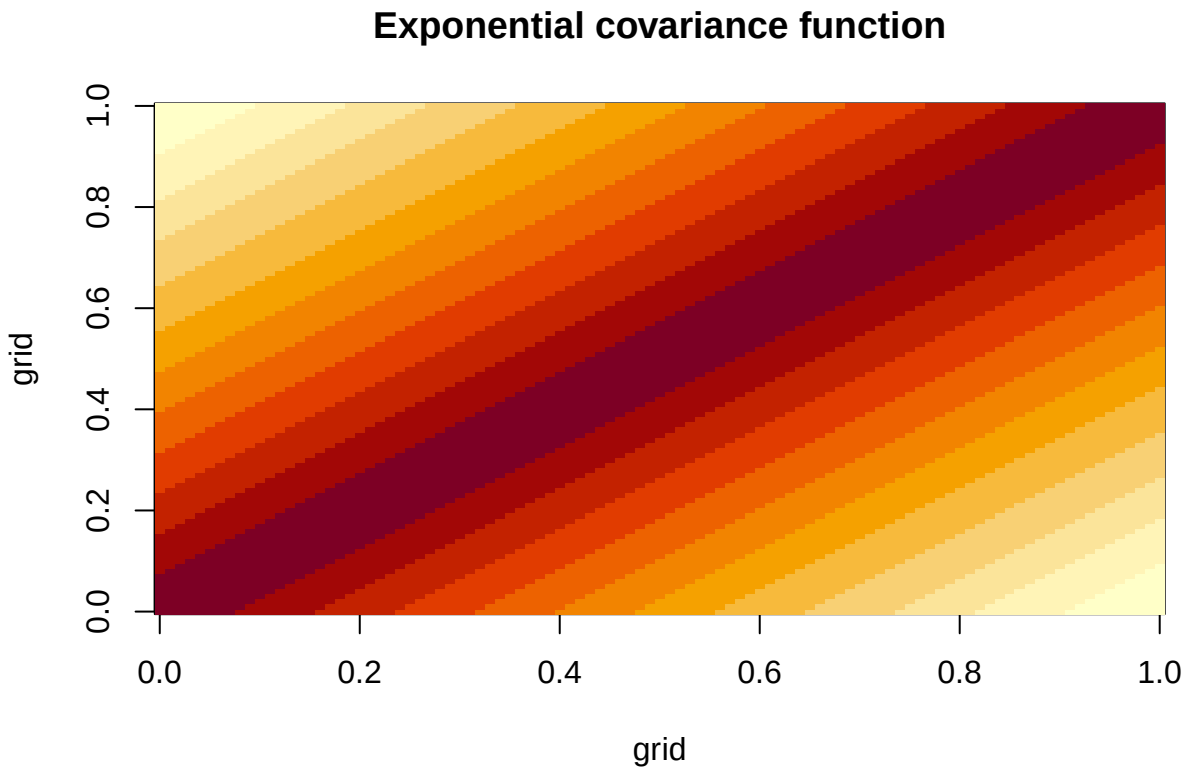
Where *alpha* controls the point-wise variance, and β the degree of covariance between points.

```
P <- 101
grid <-  seq( 0, 1, length.out =  P)

alpha <-  0.2
beta <-  0.2

C_st <- exp_cov_function( grid, alpha, beta )
dim(C_st)  # note of course it is a covariance matrix
```

```
## [1] 101 101
```

`C_st` contains a $P \times P$ matrix of values.

```
image( C_st,
       main = 'Exponential covariance function',
       xlab = 'grid', ylab = 'grid')
```



After having defined the mean function $m(t)$

```
m <- sin(pi*grid)+sin(2*pi*grid)
```

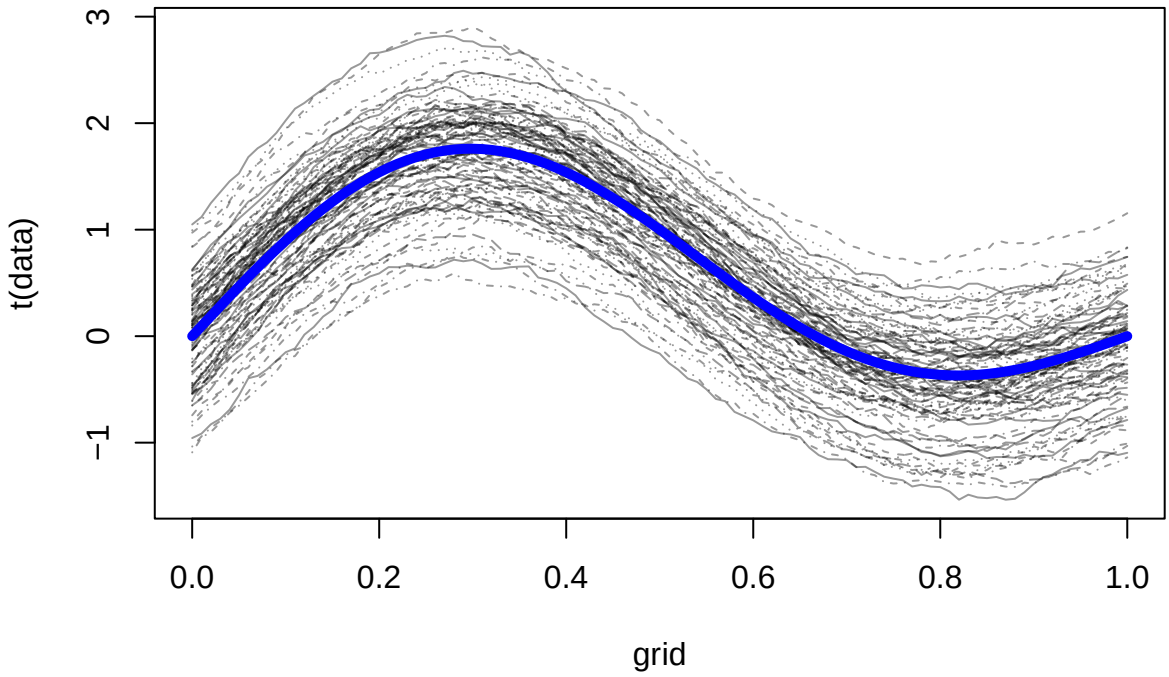
we are ready to generate functional data as follows

```r
n <- 100
set.seed(26111992)
data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)
dim(data) # 100 functions evaluated at 101 points each
```

```
## [1] 100 101
```

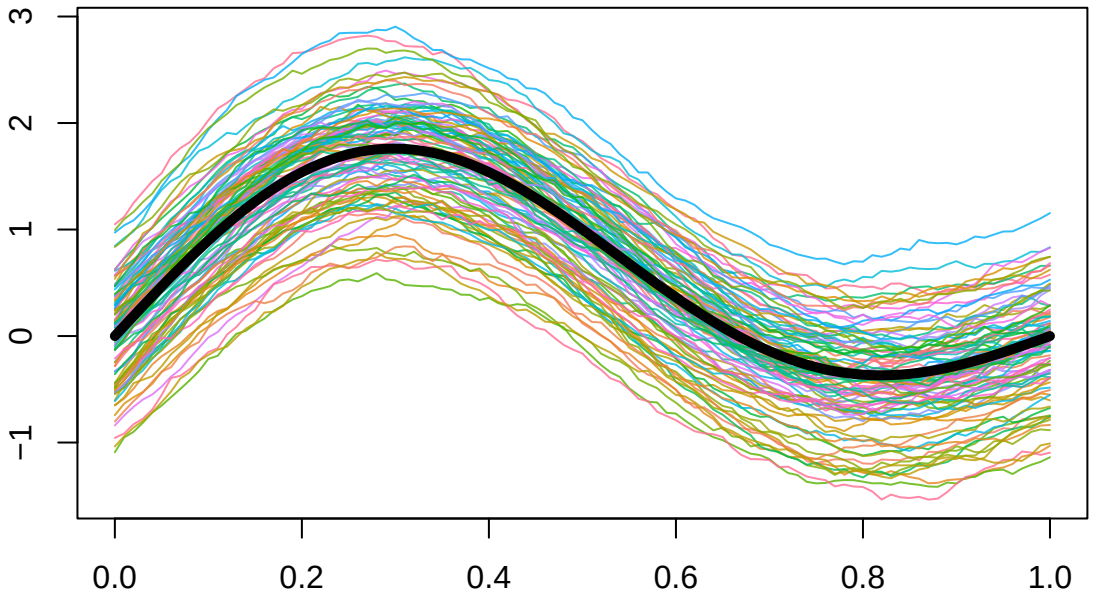
The output of the previous chunk is actually a $n \times P$ matrix, where P is equal to the length of the grid. That is, instead of having a "proper" functional datum, I have its evaluation on a relatively fine grid (this is actually the best we can do). We can plot it with:

```r
matplot(grid,t(data), type="l", col=adjustcolor(col=1,alpha.f = .4))
lines(grid,m, col="blue", lwd=5)  # add something else on top of existing graph
```



Or even better we exploit the features of the `roahd` package to construct a functional object:

```r
f_data <- fData(grid, data)
plot(f_data) # what happens if I do plot(data)?
lines(grid,m, col="black", lwd=5)
```



The first command defines an object of class fdata

```
class(f_data)
```

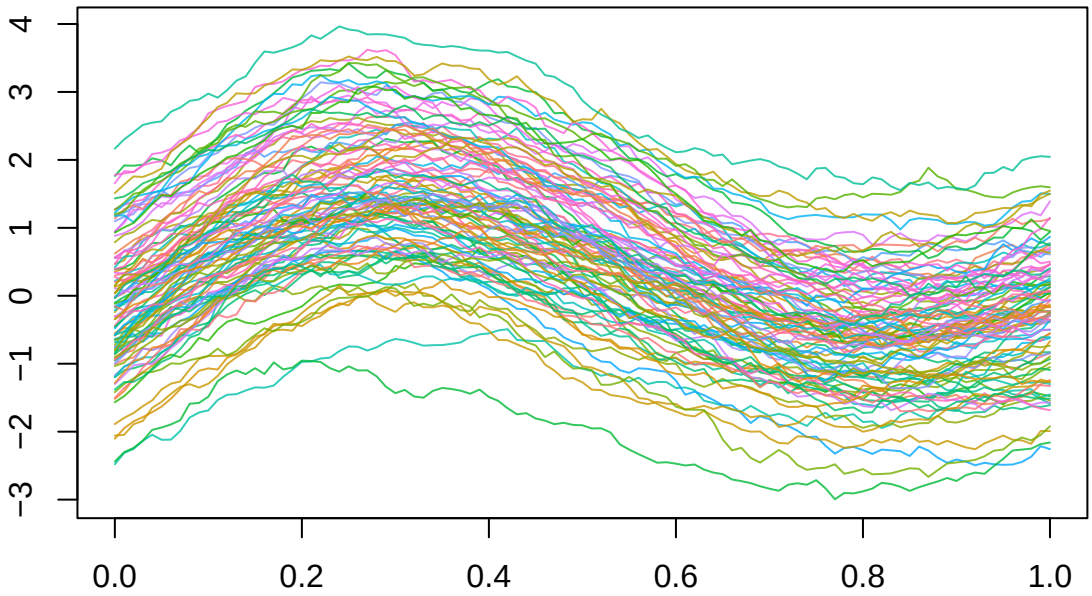
```
## [1] "fData"
```

with dedicated methods for plots, the four basic algebraic operations, subsetting and much more. In particular, using objects of class `fdata` makes the computation of depth measures and related quantities much easier (see next section). For a thorough account on the potential of the `roahd` package, check this paper.

By changing the hyperparameters α and β in the Exponential covariance function, we can generate functional datum with different degree of dependence and variability. In details, α controls the overall level of variability in the signal, while the parameter β affects the autocorrelation length of the signal's noise, with lower values of β leading to wider correlation lengths and vice-versa

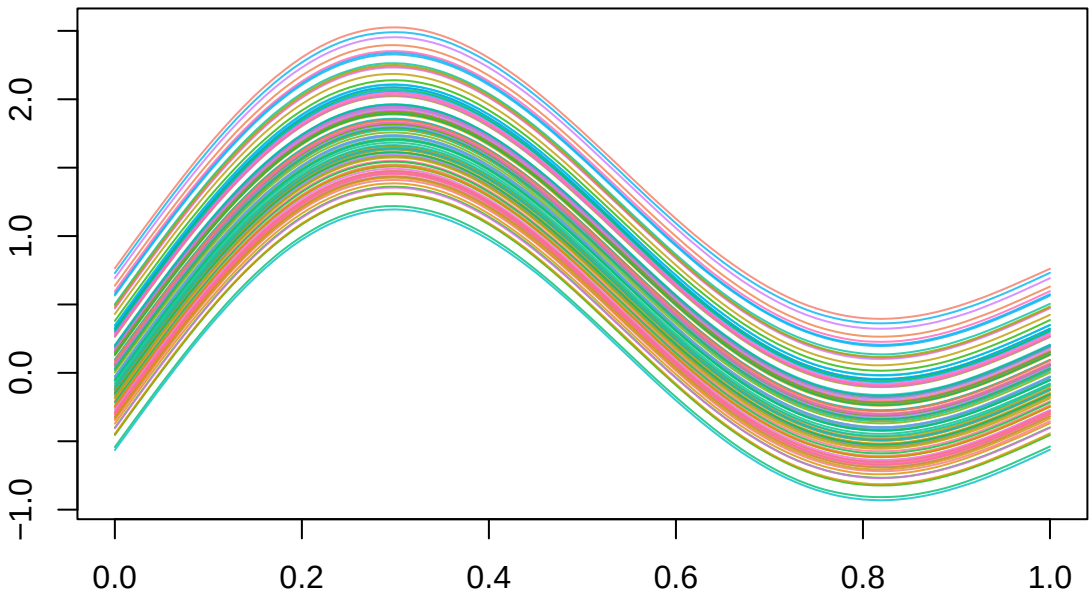
```
alpha <-  1
beta <-  0.2

C_st <- exp_cov_function( grid, alpha, beta )

data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)

f_data <- fData(grid,data)
plot(f_data, main="High overall level of variability")
```

High overall level of variability

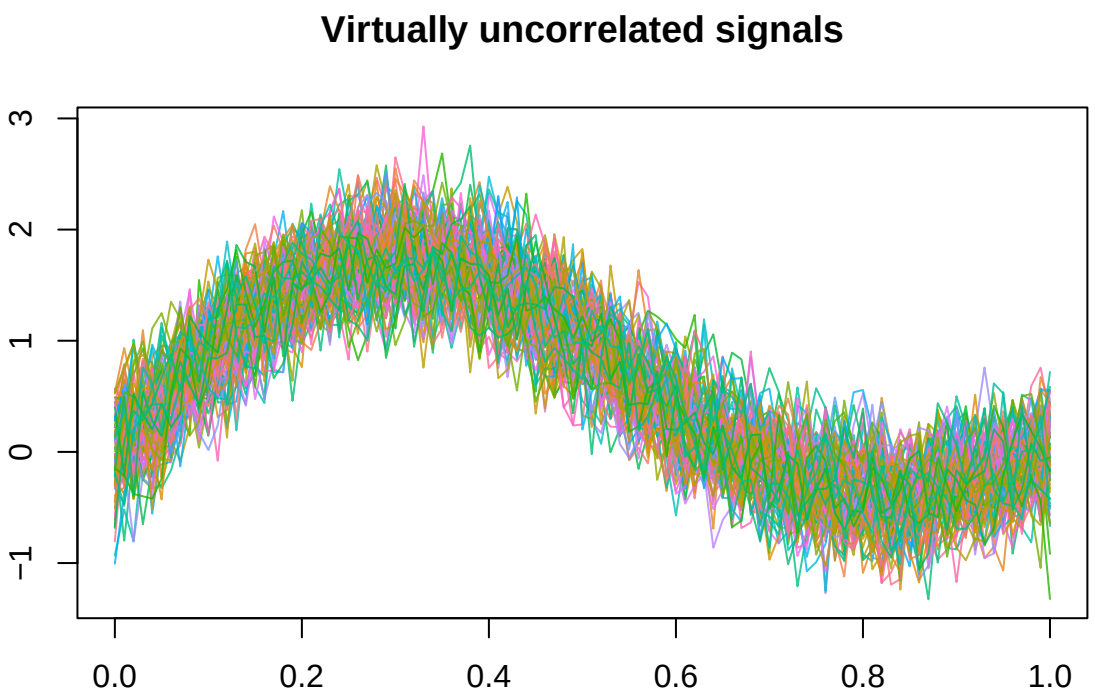



```
alpha <-  .1
beta <-  0.0001

C_st <- exp_cov_function( grid, alpha, beta )

data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)

f_data <- fData(grid,data)
plot(f_data, main="High smothness")
```

High smothness



```r
alpha <-  .1
beta <-  100

C_st <- exp_cov_function( grid, alpha, beta )

data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)

f_data <- fData(grid,data)
plot(f_data, main="Virtually uncorrelated signals")
```



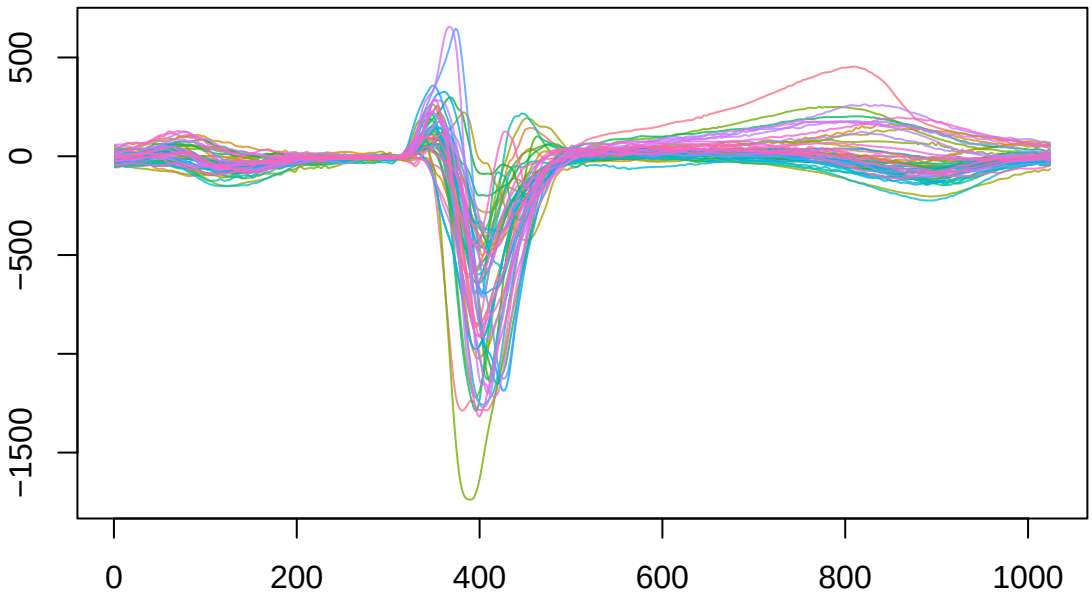
Computing depth measures in a FDA framework

Let us now consider the data you have seen in the Case Study: ECG signals. The registered and smoothed signals are contained in the `mfD_healthy` object.

```r
data("mfD_healthy") # included in roahd
univariate_fdata <- mfD_healthy$fDList[[1]] # I consider the first lead only
print(class(univariate_fdata))  #fData object
```

```
## [1] "fData"
```

```r
plot(univariate_fdata) # can do it directly with an fData object
```



With `roadhd` it is very easy to compute Band depths and Modified band depths for a given `fdata` object:

```
band_depth <- BD(Data = univariate_fdata)
modified_band_depth <- MBD(Data = univariate_fdata)
```

We can compute the median curve

```
median_curve <- median_fData(fData = univariate_fdata, type = "MBD") # still an fData object
```

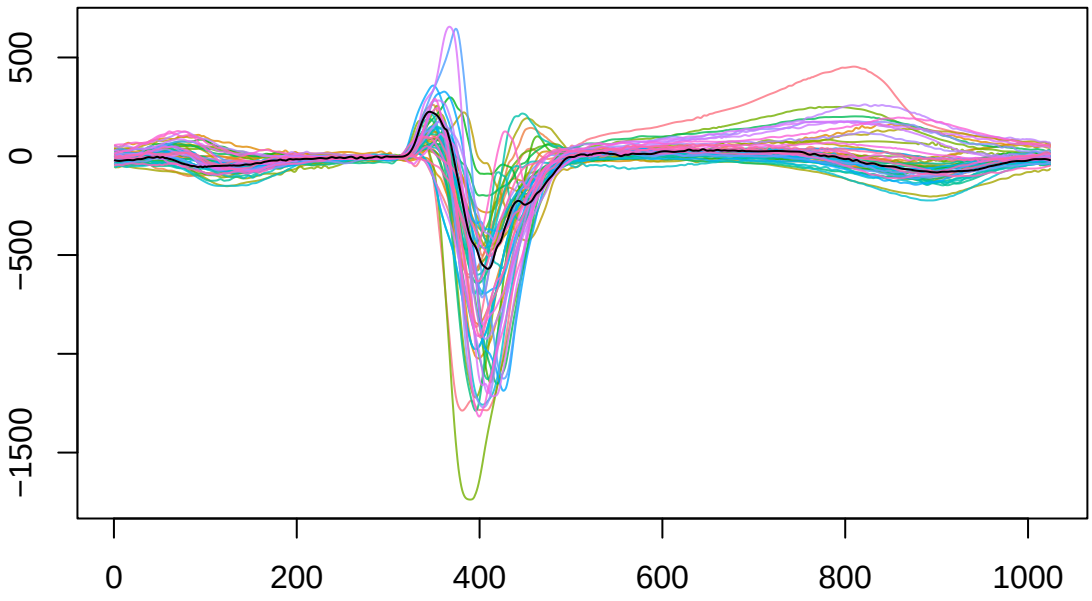
Or manually by firstly computing the Modified Band Depth and then identifying the curve with max MBD

```
median_curve_manual <- univariate_fdata[which.max(modified_band_depth),] # still an fData object
```

```
all(median_curve_manual$values==median_curve$values)
```

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

```
plot(univariate_fdata)
grid_ecg <- seq(median_curve_manual$t0,median_curve_manual$tP,by=median_curve_manual$h)
lines(grid_ecg,median_curve_manual$values)
```



Parenthesis: the failure of multivariate depth functions in high dimensinons

What if we used the Tukey Median for a functional datum? Let's try it:

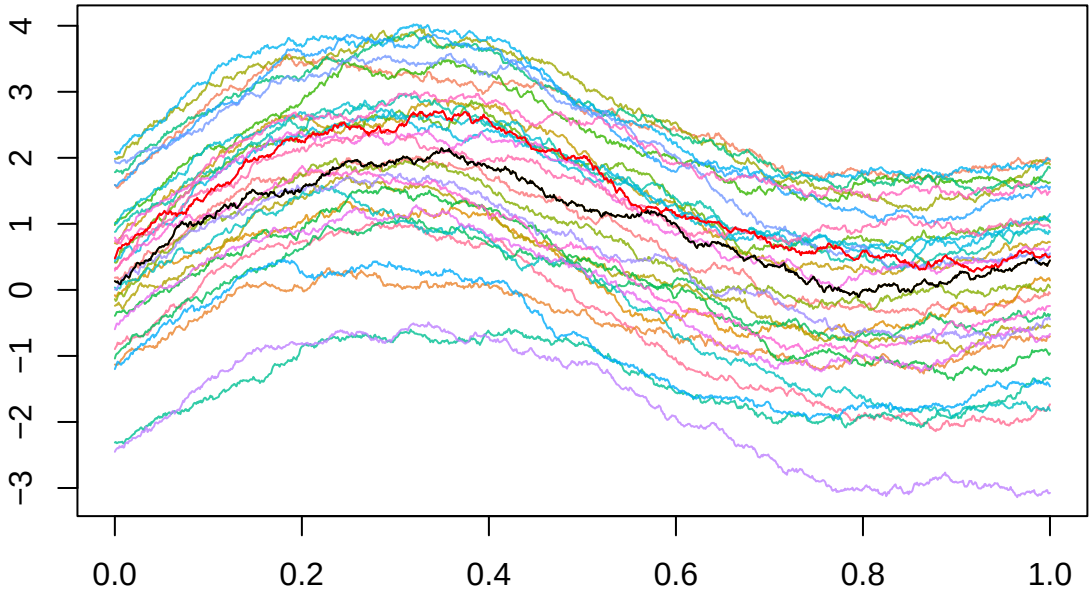
```
set.seed(2)
n = 32
# generate functional datset
grid = seq(0,1,length.out=1000)
alpha <-  1
beta <-  0.2
C_st <- exp_cov_function( grid, alpha, beta )
m <- sin(pi*grid)+sin(2*pi*grid)
data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)
f_data = fData(grid=grid, values=data)  # cast to fData
plot(f_data)
median.mbd =  median_fData(fData = f_data, type = "MBD")

lines(grid, median.mbd$values, col="black")
library(DepthProc)
```

```
## Loading required package: ggplot2

## Loading required package: Rcpp

## Loading required package: rrcov

## Loading required package: robustbase

## Scalable Robust Estimators with High Breakdown Point (version 1.7-4)

## Loading required package: MASS

## Loading required package: np

## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-17)
## [vignette("np_faq",package="np") provides answers to frequently asked questions]
## [vignette("np",package="np") an overview]
## [vignette("entropy_np",package="np") an overview of entropy-based methods]
```

```
##
## Attaching package: 'DepthProc'
```

```
## The following object is masked from 'package:base':
##
##     as.matrix
```

```
tukey.depth=depth(u=f_data$values,method='Tukey')
tukey.deepest.idx = which(tukey.depth==max(tukey.depth))
lines(grid, f_data$values[tukey.deepest.idx[1],], col="red")
```



It does not seem like a correct estimator for location... This is due to the fact that as d increases, heuristic approaches are utilised for classical depth measures (such as Tukey depth), yielding poor estimation. Remember, in the multivariate setting, depth measures deal with the "**center-outward order**", whereas in FDA we try to obtain a "**down-upward order**".

Spearman's correlation index for functiona data

Epigraph Index (EI) and Hypograph Index (HI) or their corresponding Modified versions (MEI and MHI) for providing down-upward/up-downward order of data can also be computed: see functions `EI`, `HI`, `MEI`, `MHI`. As an example, let us compute the Spearman's correlation index between the first and second lead of the ECG signals. Recall that for a bivariate functional dataset $[\mathbf{x}, \mathbf{y}]$, the estimate for Spearman's correlation index $\hat{\rho}_s(\mathbf{x}, \mathbf{y})$ is given

$$\hat{\rho}_s(\mathbf{x}, \mathbf{y}) = \hat{\rho}_p\left(IL\,\text{grade}_n(\mathbf{x}), IL\,\text{grade}_n(\mathbf{y})\right)$$

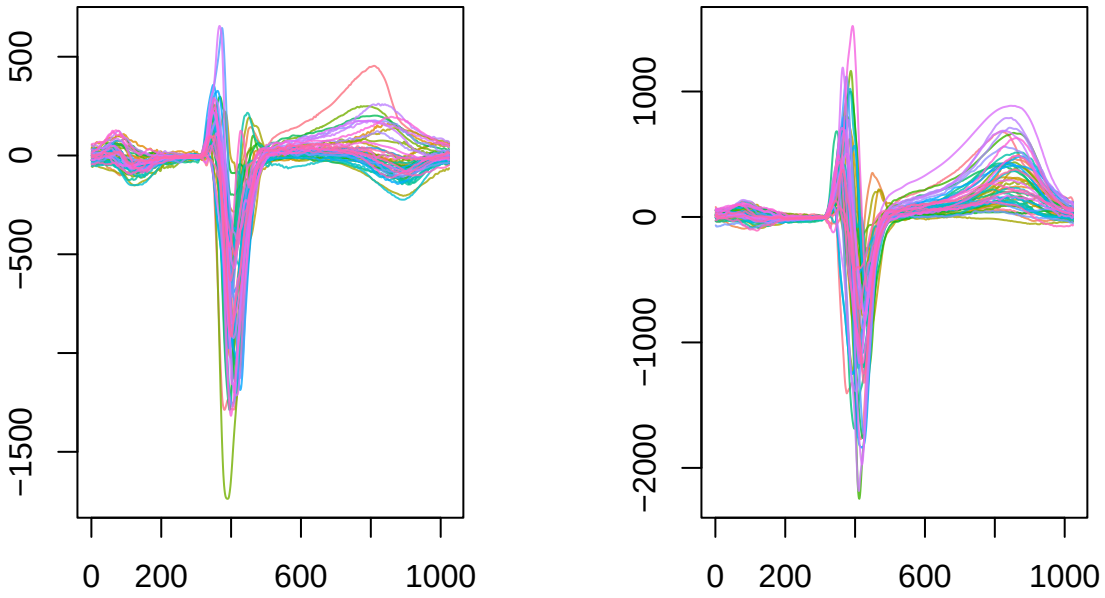
where $\hat{\rho}_p$ is the sample Pearson correlation coefficient and $IL\,\text{grade}_n$ is the Inferior-Length w.r.t the sample and

$$IL_n - \text{grade}(\mathbf{x}) = \left(IL_n - \text{grade}\,(x_1), IL_n - \text{grade}\,(x_2), \ldots, IL_n - \text{grade}\,(x_n)\right)$$
$$IL_n - \text{grade}(\mathbf{y}) = \left(IL_n - \text{grade}\,(y_1), IL_n - \text{grade}\,(y_2), \ldots, IL_n - \text{grade}\,(y_n)\right).$$

where the inferior length grade $IL\,\text{grade}_n(x_i)$ can be interpreted as the "proportion of time" that the sample $x_1, \ldots, x_n$ is smaller than x_i, that is it defines the relative position of a curve with respect to the sample (it pretty much counts what percentage of the domain curve i is higher than other samples)

The Spearman's correlation index is immediately obtained via the `cor_spearman` function

```r
bivariate_data <- as.mfData(list(mfD_healthy$fDList[[1]], mfD_healthy$fDList[[2]]))
plot(bivariate_data)
```




```r
cor_spearman(bivariate_data, ordering='MHI')
```

```
## [1] 0.6811366
```

Or actually we can manually compute it:

```r
MHI_first_lead <- MHI(bivariate_data$fDList[[1]]) # modified hypograph
MHI_second_lead <- MHI(bivariate_data$fDList[[2]])

cor(MHI_first_lead, MHI_second_lead)
```

```
## [1] 0.6811366
```

Coding tip: never forget the power of functional programming!

```r
do.call(args = lapply(1:2, function(ind)
  MHI(bivariate_data$fDList[[ind]])) # list of arguments
  , what = "cor" # name of the function
  )
```

```
## [1] 0.6811366
```

Outlier detection with roahd: functional boxplots and outliergrams

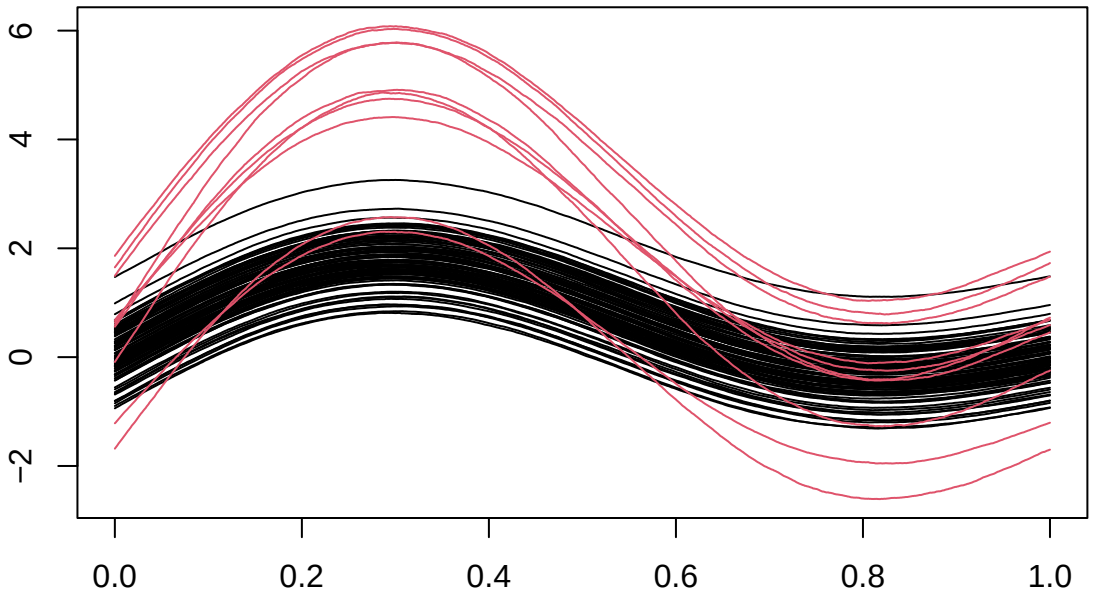
We conclude this lab by looking at some graphical tools for identyfing outliers in a functional sample. Consider the following univariate functional data

```r
n = 100
alpha <-  0.2
beta <-  0.002

C_st <- exp_cov_function( grid, alpha, beta )

data <- generate_gauss_fdata(N = n,centerline = m,Cov=C_st)
```

```
f_data <- fData(grid,data)
```

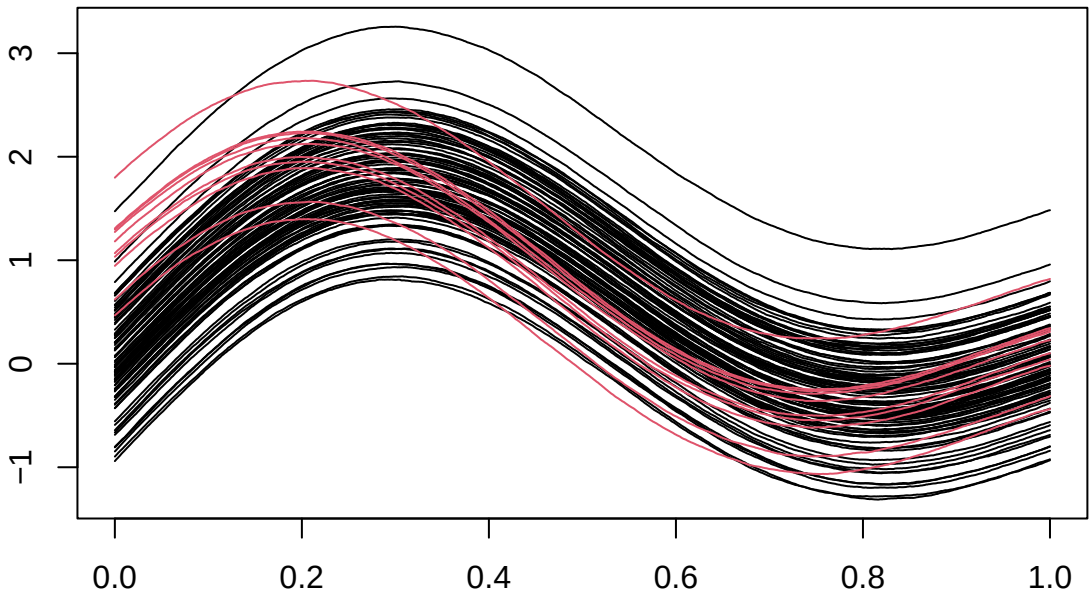
We consider two cases: first a dataset with some magnitude outliers, obtained inflating the last 10 curves by a number generated from a uniform distribution in $[2, 3]$

```
set.seed(33) # reproducibility
outlier_share <- .1
n_outliers <-   n*outlier_share
out_highlighter <- rep(c(1,2),c(n-n_outliers,n_outliers))
f_data_temp <- f_data[1:(n*(1-outlier_share)) ,] # Coding tip: subsetting is mabe possible by the S3 cl
mag_temp <- f_data[(n*(1-outlier_share)+1):n,] * runif(10,2,3)

f_data_mag <- append_fData(f_data_temp,mag_temp)
plot(f_data_mag, col=out_highlighter)
```



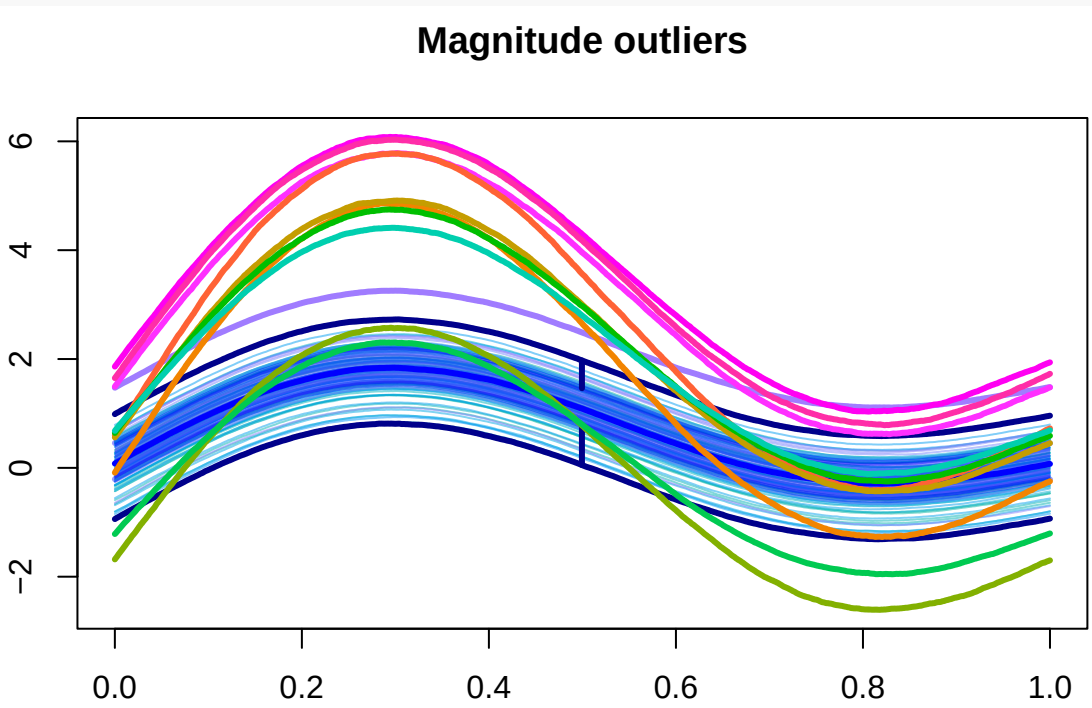
And then some shape outliers by shifting the generating mechanism by the quantity `shift_q`:

```
shift_q <- .5

mu_warp=mu=sin(pi*grid+shift_q)+sin(2*pi*grid+shift_q)

shape_temp=generate_gauss_fdata(N = n_outliers, mu_warp, Cov=C_st)
shape_temp=fData(grid,shape_temp)
f_data_shape=append_fData(f_data_temp,shape_temp)
plot(f_data_shape, col=out_highlighter)
```



Now let us see whether we can employ the two plotting tools we have seen in class, namely functional boxplots and outliegrams to detect magnitude and shape outliers, respectively.

```
invisible(fbplot(f_data_mag, main="Magnitude outliers"))
```



Recall:

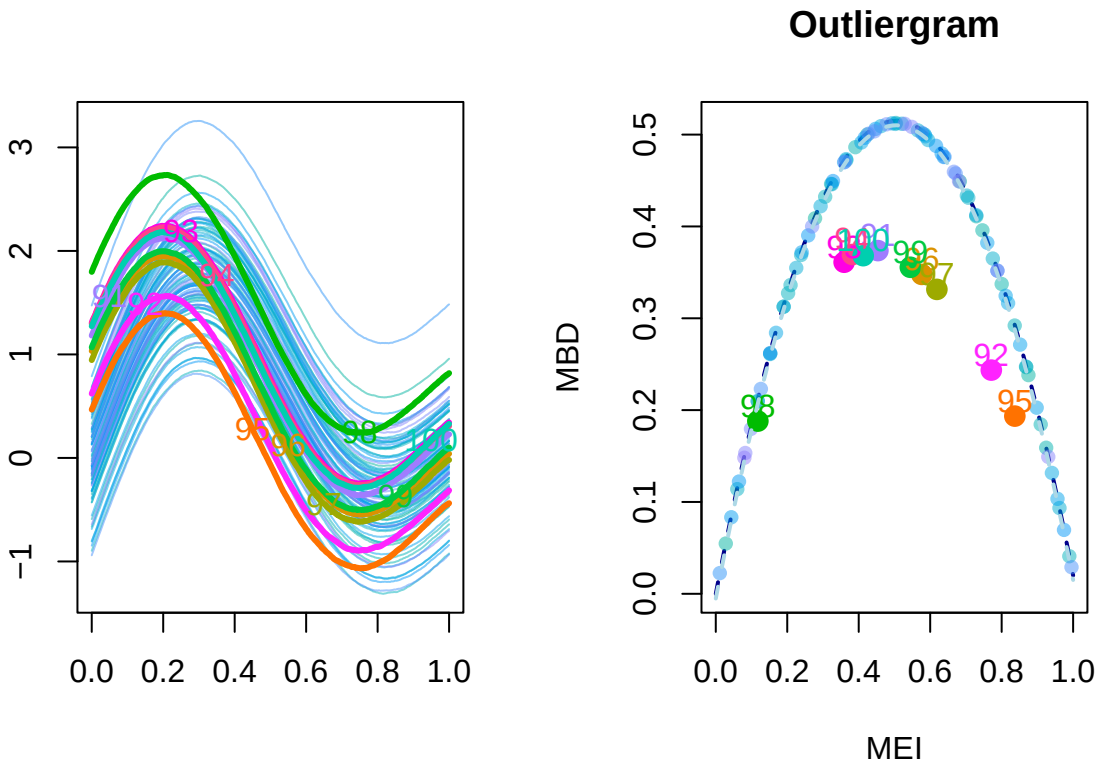
- The central region ("functional bag") contains the functions with the highest MBD value
- The central region is inflated by factor F to obtain the fences.

I used `invisible()` in the previous chunk to avoid the output of the call to be printed on the console (and in the knitted html file). If I do not use it I obtain the following:

```
#fbplot(f_data_shape, main="Shape outliers")
```

Now, we plot the outliergram.

```
invisible(outliergram(f_data_shape))
```



We appreciate how the shape outliers are almost entirely missed by the functional boxplot, whereas the outliergram effectively uncovers them. Recall the important result:

$$MBD(f) \leq a_0 + a_1 MEI(f) + a_2 N^2 MEI^2(f)$$

where MBD represents the **Modified Band Depth** and MEI the **Modified Epygraph Index**.

As usual, saving the output of the plots to an object allows to recover the ID of the identified outliers:

```
out_shape <- outliergram(f_data_shape, display = FALSE)
out_shape$ID_outliers
```

```
##  [1]  91  92  93  94  95  96  97  98  99 100
```

The outliergram is actually based on some simple manipulations of MEI and MBD, that can be hard coded:

```
MEI_out_shape <- MEI(f_data_shape)
MBD_out_shape <- MBD(f_data_shape)
a_0 <- a_2 <- -2/(f_data_shape$N*(f_data_shape$N-1))
a_1 <- 2*(f_data_shape$N+1)/(f_data_shape$N-1)
d_manual <- a_0+a_1*MEI_out_shape+a_2*f_data_shape$N^2*MEI_out_shape^2-MBD_out_shape

critical.quantile = quantile(d_manual,probs = .75)
inflation =out_shape$Fvalue
inter.q.range = IQR(x = d_manual)

ID_outliers_manual <- which(d_manual > critical.quantile + inflation *inter.q.range )
```

Let us check that our manual computations are correct:

```
all(dplyr::near(d_manual, out_shape$d))
```

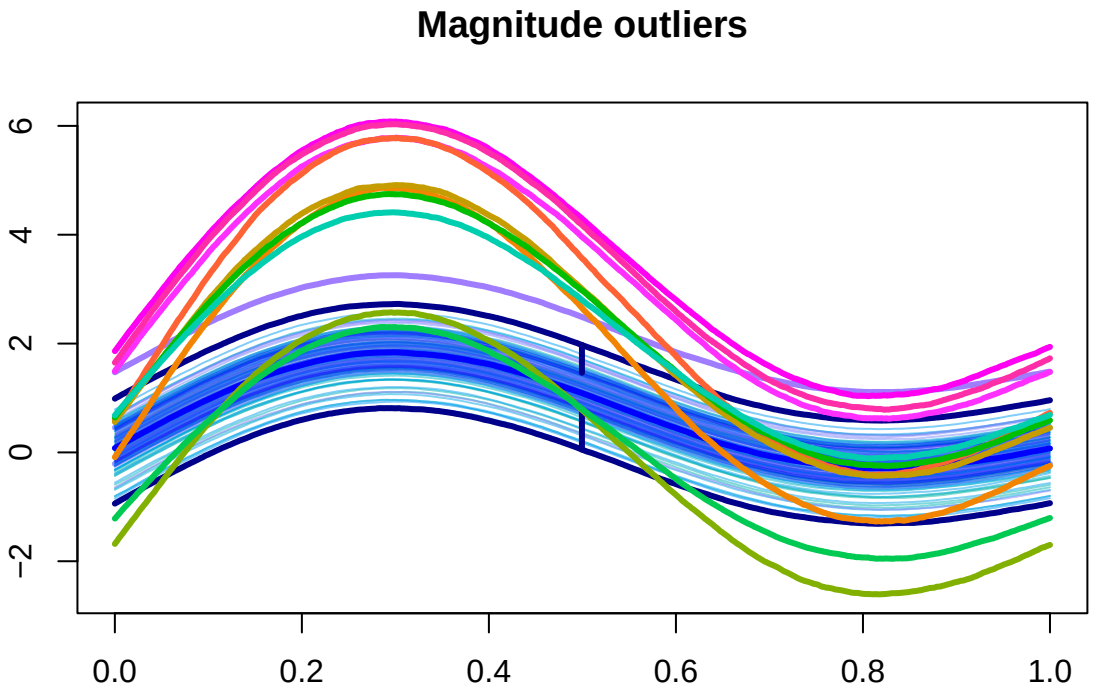
```
## [1] TRUE
```

```
all(ID_outliers_manual==out_shape$ID_outliers)
```

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

One last comment: we have seen in class that the value F used to build the fences in the functional boxplot can be adjusted. Despite the algorithmic implementation/ mathematical theory behind it being quite tedious, the automated selection of F via the `roahd` package is actually pretty easy:

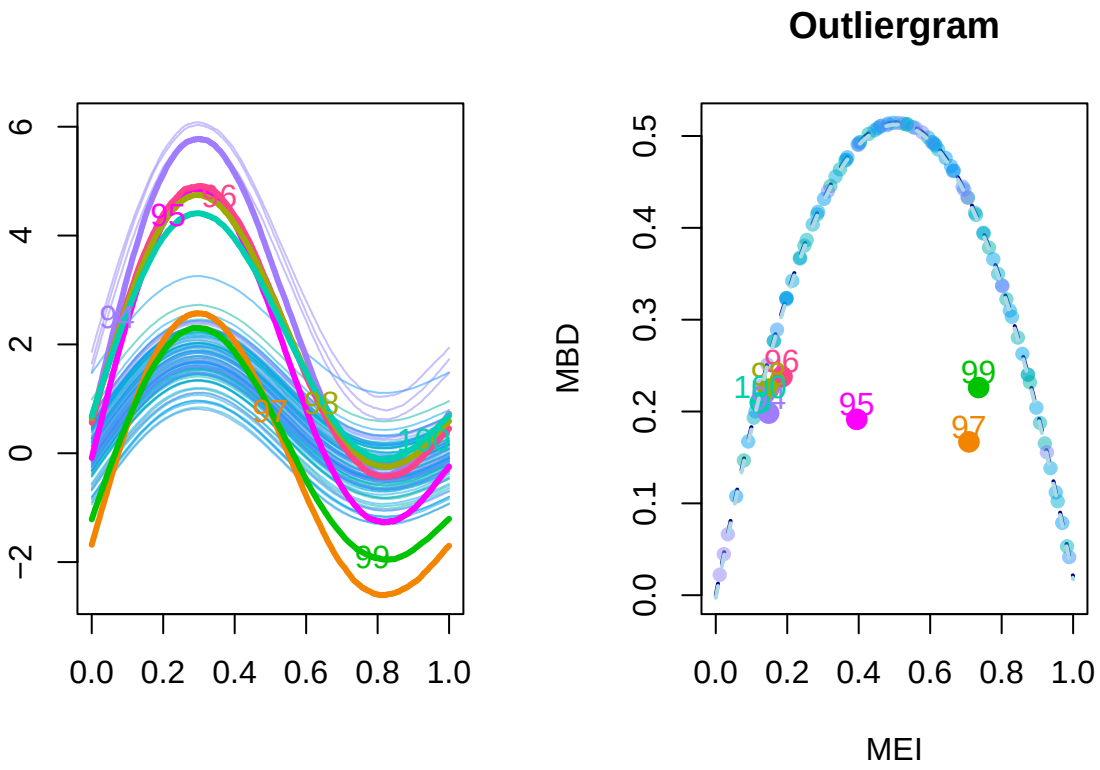
```
set.seed(22)
fbplot(f_data_mag, main="Magnitude outliers",adjust = F)
```




```
## $Depth
##   [1] 0.43266061 0.44557414 0.14673778 0.23181859 0.27697455 0.39373939
##   [7] 0.37845980 0.24007152 0.32229091 0.51057212 0.36747838 0.16524040
##  [13] 0.50228525 0.32346061 0.18765010 0.47359798 0.28053818 0.46371354
##  [19] 0.05319111 0.51289253 0.38057091 0.38679071 0.49317657 0.19332404
##  [25] 0.41456040 0.41412768 0.49339838 0.41633576 0.34986222 0.24021657
##  [31] 0.49830626 0.50972606 0.10227394 0.51254788 0.40362545 0.47396566
##  [37] 0.48527192 0.51431515 0.45584525 0.49268242 0.27655596 0.51354828
##  [43] 0.36665374 0.32214141 0.49133051 0.39458626 0.13835919 0.50617293
##  [49] 0.34231394 0.30308364 0.48971960 0.30998545 0.11199232 0.20417293
##  [55] 0.36579192 0.26277293 0.41747596 0.49104929 0.46184444 0.16736768
##  [61] 0.33736687 0.51357697 0.46712485 0.20010182 0.47614222 0.43161293
##  [67] 0.50855758 0.07877455 0.47681414 0.32361333 0.10765657 0.46227111
##  [73] 0.28913414 0.04148000 0.44161899 0.49318505 0.51117091 0.51179434
##  [79] 0.44490626 0.51419515 0.49036525 0.43991354 0.25115556 0.50871717
##  [85] 0.50398263 0.22841657 0.15565293 0.33634263 0.43330788 0.22293818
##  [91] 0.06640808 0.02221737 0.04468525 0.19820444 0.19148566 0.23793455
##  [97] 0.16695798 0.22435515 0.22609455 0.20964444
##
```

```
## $Fvalue
## [1] 1.5
##
## $ID_outliers
##  [1]  71  91  92  93  94  95  96  97  98  99 100
```

```
outliergram(f_data_mag,adjust = F)
```




```
## $Fvalue
## [1] 1.5
##
## $d
##   [1] 6.806010e-04 8.885735e-04 1.127495e-03 4.222739e-04 1.098482e-03
##   [6] 2.947677e-04 3.419747e-04 3.475376e-04 6.537598e-04 4.399497e-04
##  [11] 9.680089e-04 4.371941e-04 6.715473e-04 1.124969e-03 3.084394e-04
##  [16] 6.551911e-04 3.353840e-04 1.115539e-03 1.284798e-04 2.108897e-04
##  [21] 8.721325e-04 1.582768e-03 7.557689e-04 7.828232e-04 1.627989e-03
##  [26] 2.835160e-04 6.863384e-04 5.127749e-04 1.023838e-03 6.640614e-04
##  [31] 5.331038e-04 8.907143e-04 3.741331e-04 7.272558e-04 1.011489e-03
##  [36] 1.545786e-03 1.936275e-03 5.631968e-04 8.601679e-04 1.278783e-03
##  [41] 1.544740e-03 6.241261e-04 1.781773e-03 1.947660e-03 3.235354e-04
##  [46] 5.231675e-04 4.076717e-04 1.398364e-03 1.683872e-03 4.323182e-04
##  [51] 9.824063e-04 9.548436e-04 2.376160e-04 3.155505e-04 2.615335e-04
##  [56] 4.203990e-04 8.011796e-04 4.132453e-04 1.999279e-03 1.126823e-03
##  [61] 2.362244e-04 3.893648e-04 2.560386e-04 1.544756e-03 2.064194e-04
##  [66] 9.710699e-04 1.063515e-03 2.523182e-04 3.516717e-04 4.633204e-04
##  [71] 2.061798e-03 3.111790e-04 9.722778e-04 3.146659e-04 3.749293e-04
##  [76] 1.770198e-03 3.815836e-04 1.979132e-03 4.656111e-04 7.241568e-04
##  [81] 2.192000e-03 1.256224e-03 1.123668e-03 4.006432e-04 3.341446e-04
##  [86] 8.841947e-04 2.120063e-04 5.653574e-04 1.529212e-03 1.786094e-03
```

```
##  [91] 4.759596e-05 2.009212e-05 3.808000e-05 5.916414e-02 2.988338e-01
##  [96] 7.055863e-02 2.644220e-01 3.276808e-02 1.815623e-01 1.279333e-02
##
## $ID_outliers
## [1]  94  95  96  97  98  99 100
```