Bible NPS

Bob Stilson
Bill Chu

BLOCK I : Depth Theory :

1) Multivariate Case :

Loading necessary libraries :

```
library(MASS)
library(ddalpha)
library(depth.plot)
library(depthgram)
library(mvtnorm)
library(ggplot2)
library(ggforce)
library(ggpubr)
library(ggsignif)
# devtools::install_github("ggbiplot/ggbiplot") # for ellipses
```

Simulation of points and visualisation :

Let us start by simulating 700 bivariate elongated normal distribution in ellipsoidal orientation

```
set.seed(7654321) # for reproducibility
n <- 700
mu_bivariate_vec <- c(0,0,0,0,0,0,0,0)
sigma_bivariate_mat_diag
```

```
##  [1] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [2] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [3] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [4] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [5] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [6] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [7] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [8] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

and create their multivariate

```
pairs(mvrnorm(n,mu_vec,diag(sigma_bivariate_diag)), col=rgb(1,0,0,0.2))
```

We can further analyse a hexagonal binning plot to visualise the data density

```
plot(mvrnorm(n,mu_vec,diag(sigma_bivariate_diag)), xlim=c(-3,3), ylim=c(-3,3), pch=19, cex=0.2)
points(0,0,pch=19,col="blue",cex=2)
```

Depth measures (multivariate) :

Now in depths, there are many proposals such as Mahalanobis depth, Liu's (2004) or Liu's (1993) simplicial depth, Tukey's (1975) half-spaces, which is a great general-purpose average-case depth and is added in the great plotting capabilities. Even though not all depth measures are affine invariant, Tukey's is affine invariant. Tukey's depth has also good theoretical properties, such as its consistency and its robustness (it is 1/3-robust throughout the sample). Finally, **Tukey** and **Mahalanobis** depths, can go as low as one may like to be, equal. Number.

Tukey depth :

Tukey's depth of a given dataset is immediately accomplished by typing

```
library(depthgram)
data(bivariate)
```

It has its useful binary depth and it will become apparent when we study, cooperative forecasting to calculate the depth of a point relative to a sample. You can do it by typing

```
depth(c(1,0),c(1,1),D,D,plot=TRUE,method = "Tukey")
```

```
## depth method: Tukey
## [1] 0.750
```

```
# c(1,1) is the point that would be compared the depth
```

Two ways to compute the median :

Compute the median depth with mahalanobis function

```
depth(median(D),D,plot=TRUE,method = "Mahalanobis")
```

```
## [1] 0.8333333 0.5833333
```

Or, having already computed the Tukey depth for the whole sample, since the median is the deepest point, type

```
id_bivariate_depth=order(depth(D,D,plot=TRUE,method = "Tukey"))
id_bivariate_depth[1]
```

```
## [1] 2 0.5833333
```

Two ways to visualise the depth surface :

If you have the luxe of dealing with a bivariate dataset, you can easily visualise the depth surface in very convenient way: depthmap offers you impressive visualisation.

```
depthmap(data[,2],data[,1],depth_bivariate_plot,ncol=50,method="Tukey")
```

or in 3D, let usually more clearly see the deepest points

```
depthperspec(data[,2],data[,1],depth_bivariate,ncol=50,method="Tukey")
```

Fry additional graphic effects :

```
depthperspec(data[,2],data[,1],depth_bivariate,ncol=50,method="Tukey",zlim,method = "qq")
```

Computational cannot

Since Tukey Depth is obtained by counting points in the square of hyperplanes(they form a d^2 where d = 2 in the computational bivariate simulation (Chindo usually stick to Tukey's methods which tend to pick outliers) but will not be in the bivariate case depth measure also

```
# c(1,1) depth(bivariate,c(1,1),method="Tukey")
```

Mahalanobis depth :

Tukey's depth can be computed also by considering a different depth measure

```
Tukey_depth <- depth(D,D,plot=TRUE,method = "Mahalanobis")
```

This can be easily hard coded, except the definition you have seen in class

$$D_M(x|F,S) = \frac{1}{1 + (x - \mu)^T S^{-1} (x - \mu)}$$

This is the "hard code" :

```
sample_mean <- colMeans(bivariate_data)
sample_cov <- cov(bivariate_data)
n <- nrow(bivariate_data)
mahal_depth_points <- 1/(1+mahalanobis(t(t(bivariate_data)-sample_mean),c(0,0),sample_cov))
```

And check that the obtained result is equal to the one obtained via the depth function (beware of computing-freak).

```
id_bivariate_depth=order(depth(D,D,plot=TRUE,method = "Mahalanobis"))
id_bivariate_depth[1]
```

```
## [1] 2 0.5833333
```

Again, two ways to compute the median

```
depth(median(D),D,plot=TRUE,method = "Mahalanobis")
```

```
## [1] 0.8333333 0.5833333
```

```
id_bivariate_depth=order(depth(D,D,plot=TRUE,method = "Mahalanobis"))
id_bivariate_depth[1]
```

```
## [1] 2 0.5833333 0.5833333
```

And again the graphical depiction

```
depthmap(data[,2],data[,1],depth_bivariate,ncol=50,method="Mahalanobis")
```

Mahalanobis depth

```
depthperspec(data[,2],data[,1],depth_bivariate,ncol=50,method="Mahalanobis",zlim,method = "qq")
```

Please note that anything that comes out of depthperspec or depthmap is NOT a density: we are not doing density estimation here, we are performing data exploration by means of computational procedure. Let us plot the probability density function (PDF) the depth function of the bivariate exponential distribution (see below).

```
library(MASS)library(ggplot2)library(ggsignif)library(ggforce)library(ggpubr)library(mvtnorm)
set.seed(7654321)
n <- 700
```

PDF (NOT depth function) of the bivariate exponential distribution


```
depthmap(data[,2],data[,1],depth_bivariate,ncol=50,method="Tukey",zlim,zlim=TRUE)Tukey depth for the sea
```

Tukey depth for the exp distribution (unexpected around the median)

We notice that the tukey depth underestimates around the median, unlike all the exponential bivariate functions increases as x,y go to 0

Exercise about outlier detection via Depth Measure :

Try out the number seen so far on something even more exotic

```
library(ddalpha)
id_bivariate_outlier = depth(data,data,method="Mahalanobis",zero=TRUE)[order(depth(data,data,method="Mahalanobis",zero=TRUE))]
id_bivariate_outlier[1:10]
```

```
##  [1] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [9] 1.0 1.0
## [10] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [19] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [28] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [37] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [46] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [55] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [64] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [73] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [82] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [91] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [100] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [109] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [118] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [127] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [136] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [145] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [154] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [163] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [172] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [181] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [190] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [199] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [208] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [217] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [226] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [235] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [244] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [253] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [262] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [271] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [280] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [289] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [298] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [307] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [316] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [325] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [334] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [343] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [352] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [361] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [370] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [379] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [388] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [397] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [406] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [415] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [424] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [433] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [442] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [451] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [460] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [469] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [478] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [487] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [496] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [505] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [514] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [523] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [532] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [541] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [550] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [559] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [568] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [577] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [586] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [595] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [604] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [613] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [622] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [631] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [640] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [649] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [658] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [667] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [676] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [685] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [694] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [703] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [712] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [721] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [730] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [739] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [748] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [757] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [766] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [775] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [784] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [793] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [802] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [811] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [820] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [829] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [838] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [847] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [856] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [865] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [874] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [883] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [892] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [901] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [910] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [919] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [928] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [937] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [946] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [955] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [964] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [973] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [982] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [991] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [1000] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```


We can clearly see those red points at those. - Are we say that there is an outlierness before? The depth contour plot surely helps :

```
depthmap(data[,2],data[,1],depth_bivariate,ncol=50,method="Tukey",
plot=TRUE,
points = id_bivariate_outlier[1:10],
points.col = "red",
points.cex = 1,
points.pch = 19,
points.lwd = 1,
points.lty = 1,
points.bg = "red",
points.alpha = 1)
```


Bonnie!

- This tag is composed of the Mahalanobis DEPTH of the data in $\mathbb{R}^2$
- Tukey's depth is computed by counting points in the square of hyperplanes
- Tukey's depth is affine invariant, Mahalanobis depth is not affine invariant
- The bivariate exponential distribution is not affine invariant, Mahalanobis, and of the data plot the outlierness by looking depth histogram
- Tukey's depth is not affine invariant, but it is robust (1/3-robust) and it is consistent
- Tukey's depth is not affine invariant, but it is robust (1/3-robust) and it is consistent

```
depthgram(depthgram(data,data,method="Mahalanobis"),data,method="Tukey")
```


In addition, if we save the output of the bagplot or sun plot, we can automatically extract the outliers

```
bagplot_outl_indexes <- bagplot(D,D,method=TRUE,indexes=TRUE)
```

```
sunburst_outl_indexes <- sunburst(D,D,method=TRUE,indexes=TRUE)
```

```
outlying_obs <- bagplot_outl_indexes$outlyingness
```

Once the outlying obs have been identified, one can choose them from the original data and keep working on the dual subset only. There are many ways to do this, for instance :

In base R an "overload" operator that is a pure code alternative :

```
id_bivariate_1 <-
  id_bivariate_depth[1:10]
  - id_bivariate_depth[1:10]
  - id_bivariate_depth[1:10]
  - id_bivariate_depth[1:10]
  - id_bivariate_depth[1:10]
  - id_bivariate_depth[1:10]
```

A more "object-oriented programming" approach :

```
id_bivariate_1 <- id_bivariate_depth[-id_bivariate_depth[1:10]]id_bivariate_1 <- id_bivariate_depth[-id_bivariate_depth[1:10]]
```

```
id_bivariate_1[1:10]
```

```
## [1] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

Outlier detection in Star Cluster CYG-OB1 data

Data for the mesmerizing Russell Diagram of the Star Cluster CYG-OB1, which contains 57 stars in the direction of Cygnus, from C.Slobin.

- The data set is the Russell diagram of the star cluster CYG OB1 (Slobin, Chomiuk, Carlo, 2018)
- The data set is the Russell diagram of the star cluster CYG OB1 The Astronomy Russell diagram is the scatterplot of these data points, where the x-axis is the star's temperature and the y-axis is the star's luminosity.

```
data(cyg_obs)cyg_obs[,c(-1)]cyg_obs[,c(-1)]
```

```
## [1] "log.Teff"  "log.Lbol"
```

```
plot(cyg_obs[,c(-1)], main="Star Cluster CYG OB1")
```


We can see the same groups of points. The majority which stays together is clearly visible. We can also Bagplot, Sunburst plot, and Sun plots to the upper left corner. Let's see what happens when we use the depth measure to detect outliers. We can see that there are 10 points, 10 outliers. Let's see what happens when we use the depth measure to detect outliers. We can see that there are 10 points, 10 outliers. We can see that there are 10 points, 10 outliers. We can see that there are 10 points, 10

So you can see, this sample mean is biased due to the presence of the 14 bad points, whereas the Tukey Median is not. Let us compute it:

```
depth(NHANESsample[,c(1,2)])
```

```
## [1] 14.00 14.00
## attr(,"maxdepth")
## [1] 16
```

```
TukeyMedian(NHANESsample[,c(1,2)])
```

```
## [1] 13.97 13.97
## attr(,"depth")
## [1] 16
```

```
bad_points <- length(which(NHANES$Age.imputed))
TukeyMedian(NHANESsample[,c(1,2)])[1]/mean(NHANES[,c(1,2)][-bad_points,])
[1] 0.9997143
```

```
## [1] 0.99 0.99 0.99
```

Moving beyond $\mathbb{R}^2$

Tukey Median is NP-hard for 3 variables. Even though the computational complexity escalates quickly when it comes to computing depth measures in $\mathbb{R}^d$, with $d > 2$, we still appreciate their usefulness in exploratory analysis. Let us generate a trivariate dataset with:

```
rm(list=ls())
n <- 1000; d <- 3
mu_vector <- c(0,0,0)
sigma_matrix <- diag(d)

for(i in 1:d){
  for(j in 1:d){
    sigma_matrix[i,j] <- 0.5^(abs(i-j))
  }
}
library(MASS)
set.seed(1)
X <- mvrnorm(n=n, mu=mu_vector, Sigma=sigma_matrix)
colnames(X) <- c("X1","X2","X3")
dim(X)
```

```
## [1] 1000    3
```

```
# 3-variate scatterplot
library(car)
scatter3d(X[,1]~X[,2]+X[,3], data=as.data.frame(X),
          fit="smooth", bg.col="white", axis.scales=FALSE,
          grid=TRUE, ellipsoid=TRUE)
```

To visualize the outliers in the contour we return to a bagplot matrix:

```
bagplot.pairs(X, mc=FALSE, verbose=FALSE)
```

WARNING: The rest of the talk contains theoretical discussions only. While all hold the dimension invariance, as

- some centers may be strongly flagged as genuine points
- some good points may be strongly flagged as outliers

Tukey's Median is not dimension invariant, meaning, as we increase the dimension, we run into a bias problem, because of the concentration of measure.

Let us consider a random sample of size n from a d-variate standard normal distribution F and $\mathbb{D}(\cdot, F)$ with $\mathbb{D}(\cdot, F)$ an affine invariant depth, we can show that with probability at least $1 - \delta$ the following inequality holds:

$$\mathbb{D}(F,F) \geq \left(\frac{1}{2} - \frac{d}{n} - \sqrt{\frac{d}{n}} - \sqrt{\frac{\log(1/\delta)}{2n}}\right)$$

So sample co-variance! calculated for the sample $\mathbb{X} = \{X_1, ..., X_n\}$ from F with $X_i \sim F$, $\Sigma_n(\mathbb{X})$ is defined as

$$\mathbb{D}(\Sigma_n(\mathbb{X}), F) \geq \left(\frac{1}{2} - \frac{d}{n} - \sqrt{\frac{d}{n}} - \sqrt{\frac{\log(1/\delta)}{2n}}\right) \times \mathbb{D}(\Sigma, F)$$

The optimal Tukey Median asymptotically computes and plots the:

```
# depth = d1 (1/2 - d/n)
# d1 = Tukey Median depth
# defining d1 as depth of the Tukey median = (1/2-d/n)*(Tukey Median depth)
d1 <- (1/2 - d/n)
d1
```

```
## [1] 0.4833333
```

```
# Tukey Median
TM <- 1000
```

The result from all of this rather thick line plot shows:

```
# plot the Tukey depth vs depth
depth_vec <- depth(X, X, d=1, approx=T, seed=1)
plot(depth_vec, ylim=c(0,1), xlab="X depth", ylab="Tukey depth", main="Depth vs depth plot")
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 2 (10)
depth_vec2 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 3 (10)
depth_vec3 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 4 (10)
depth_vec4 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 5 (10)
depth_vec5 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 6 (10)
depth_vec6 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 7 (10)
depth_vec7 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 8 (10)
depth_vec8 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 9 (10)
depth_vec9 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 10 (10)
depth_vec10 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 11 (10)
depth_vec11 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 12 (10)
depth_vec12 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 13 (10)
depth_vec13 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 14 (10)
depth_vec14 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 15 (10)
depth_vec15 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 16 (10)
depth_vec16 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 17 (10)
depth_vec17 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 18 (10)
depth_vec18 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 19 (10)
depth_vec19 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```



```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

```
# plot 20 (10)
depth_vec20 <- depth(rbind(matrix(rnorm(n, mu=10, sigma=sigma_matrix)),
X), X, d=1, approx=T, seed=1)
abline(h=d1, col="red")
abline(h=TM/n, col="blue")
```

```
## [1] 0.4833333
## [1] 1000
```

```
## [1] 0.4833333
## [1] 1000
```

Clearly the distributions are not the same. What if we have some samples coming from F (e.g. the trivariate normal centered at 10)?

2) Functional Case :

Loading necessary libraries

```
library(ddalpha)
```

Functional data analysis with R's primer

Functional Data Analysis is one of the new research areas of our department.

Functional data analysis (FDA) is a branch of statistics that analyzes data providing curves, surfaces or manifolds as observations, or more generally data in an infinite-dimensional space. In contrast to the more familiar multivariate data analysis, the ordering of the data matters and small intervals of the data are informative about the nature of the curve.

A typical way to analyze such data is to first smooth the data, then take derivatives and perform functional analysis. Some of the tools we use in our research are **robust depth measures**, **functional principal component analysis**, **functional regression**, **functional clustering**, **functional classification**, **functional outlier detection**, **functional time series**, **functional inference**, **functional manifolds**, **functional shape analysis**, **functional topological data analysis**, **functional manifold learning**, **functional optimal transport**, **functional Bayesian inference**, **functional Bayesian optimization**, **functional Bayesian neural networks**, **functional Bayesian deep learning**, **functional Bayesian nonparametrics**, **functional Bayesian dynamical systems**, **functional Bayesian Markov chain Monte Carlo**, **functional Bayesian Markov chains**, **functional Bayesian Markov processes**, **functional Bayesian Markov random fields**, **functional Bayesian Markov decision processes**, **functional Bayesian Markov decision processes in reinforcement learning**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time with discrete observations**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time with discrete observations and decision making under uncertainty**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time with discrete observations and decision making under uncertainty with continuous state and action spaces**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time with discrete observations and decision making under uncertainty with continuous state and action spaces in continuous time**, **functional Bayesian Markov decision processes in reinforcement learning with continuous state and action spaces in continuous time with discrete observations and decision making under uncertainty

```
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!*

```
do.call(argv = 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

```
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 class fdata
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)
```

Two plotting tools to detect magnitude and shape outliers :

Now let us see whether we can employ the two plotting tools we have seen in class, namely *functional boxplots and outliergrams 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*MBD^_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(k = 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] 0.56028525 0.32246061 0.18765510 0.43759798 0.28053818 0.46371354
## [19] 0.05539611 0.51289253 0.38057091 0.38679071 0.49317657 0.19324024
## [25] 0.41456040 0.41412768 0.49339938 0.41653576 0.34968216657
## [31] 0.49830626 0.50972686 0.10227394 0.51254788 0.40362545 0.47394566
## [37] 0.48527192 0.51431515 0.45584525 0.49264242 0.27655596 0.51354828
## [43] 0.36465374 0.32214141 0.49133051 0.39458626 0.13835919 0.50617293
## [49] 0.24231394 0.30308364 0.49071960 0.39998545 0.11199232 0.20417293
## [55] 0.26579192 0.26277293 0.41747596 0.49104929 0.46184444 0.16736768
## [61] 0.33716687 0.51357687 0.46712485 0.20010182 0.47616222 0.43141293
## [67] 0.50855758 0.07877455 0.47681414 0.32361333 0.10765657 0.46227111
## [73] 0.28953414 0.04140800 0.44161809 0.49325855 0.51115991 0.51179434
## [79] 0.44490426 0.51419515 0.49036525 0.43991354 0.25115556 0.50871717
## [85] 0.50398263 0.32841657 0.15565293 0.33634263 0.43330788 0.22293818
## [91] 0.06660888 0.02221737 0.04668525 0.19022644 0.19340566 0.23793455
## [97] 0.16605798 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 8.855735e-04 1.127495e-03 4.222739e-04 1.098482e-03
##   [6] 2.947677e-04 3.419747e-04 3.475376e-04 6.537599e-04 4.339497e-04
##  [11] 9.480089e-04 4.377914e-04 6.715473e-04 1.124969e-03 2.084394e-04
##  [16] 6.551911e-04 3.353840e-04 1.115539e-03 1.284798e-04 2.108897e-04
##  [21] 6.721325e-04 1.562766e-03 7.557669e-04 7.828232e-04 1.027989e-03
##  [26] 2.835160e-04 6.863384e-04 5.127749e-04 1.022838e-03 6.640616e-04
##  [31] 5.331038e-04 8.907143e-04 3.741331e-04 7.272559e-04 1.011489e-03
##  [36] 1.545786e-03 1.936275e-03 5.431968e-04 8.601679e-04 1.237878e-03
##  [41] 1.564736e-03 6.241216e-04 1.791773e-03 1.947646e-03 3.233535e-04
##  [46] 5.221675e-04 4.076717e-04 1.394364e-03 1.683872e-03 4.322182e-04
##  [51] 9.824663e-04 5.548934e-04 2.376166e-04 2.155505e-04 2.653333e-04
##  [56] 4.203990e-04 8.011796e-04 4.132453e-04 1.999279e-03 1.126823e-03
##  [61] 2.302244e-04 1.019648e-04 2.563384e-04 1.544756e-03 2.064194e-04
##  [66] 9.710869e-04 1.063515e-03 2.523182e-04 3.516713e-04 4.632026e-04
##  [71] 2.061798e-03 3.111790e-04 5.722736e-04 3.146659e-04 3.745939e-04
##  [76] 1.770198e-03 3.815836e-04 1.979132e-03 4.656111e-04 7.241568e-04
##  [81] 2.392050e-03 1.252632e-03 1.123660e-03 4.050842e-04 3.182466e-04
##  [86] 8.841947e-04 2.120826e-04 5.453574e-04 5.529212e-03 1.786094e-03
##  [91] 4.759596e-05 2.009212e-05 3.808000e-05 5.914614e-02 2.998338e-01
##  [96] 7.055863e-02 2.644226e-01 2.276808e-02 1.815623e-01 1.279333e-02
##
## $ID_outliers
##  [1]  91  92  93  94  95  96  97  98  99 100
```