Lab 01 - Depth measures

Nonparametric statistics ay 2023/2024

2023-09-21

Disclaimer 1: The present material is an adaptation of the original R script prepared by Dr. Matteo Fontana for the a.y. 2020/2021 Nonparametric statistics course, and later Prof. Andrea Cappozzo (a.y 2022-2023) While I acknowledge Matteo and Andrea for the (great) work done I hereby assume responsibility for any error that may be present in this document.

Disclaimer 2: I will start from the assumption that you are all intermediate R users, if this is not the case please let me know, and we shall find a solution together.

Loading necessary libraries

```r
library(MASS)
library(rgl)
library(DepthProc)
library(hexbin)
library(aplpack)
library(robustbase)
library(MDBED)  # plot exponential bivariate distr+ibution
```

Computing depths with R: some general ideas

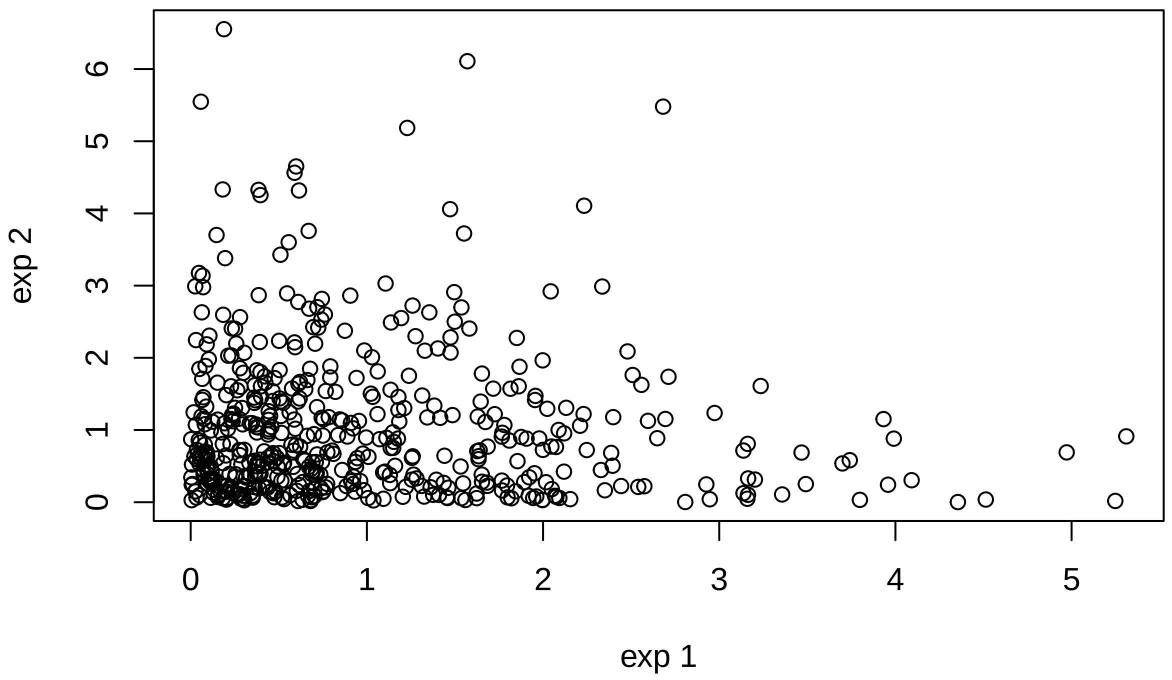
Let us start by simulating 500 bivariate datapoints whose distribution is marginally exponential

```r
set.seed(2781991) # reproducibility
n=500
df_bivariate_exp = cbind(rexp(n), rexp(n))
head(df_bivariate_exp)
```

```
##           [,1]       [,2]
## [1,] 0.3004277 0.35894610
## [2,] 2.4430839 0.22392999
## [3,] 1.8444128 0.15064919
## [4,] 0.2383688 1.12290176
## [5,] 1.9613656 0.07966151
## [6,] 5.2480885 0.01762207
```

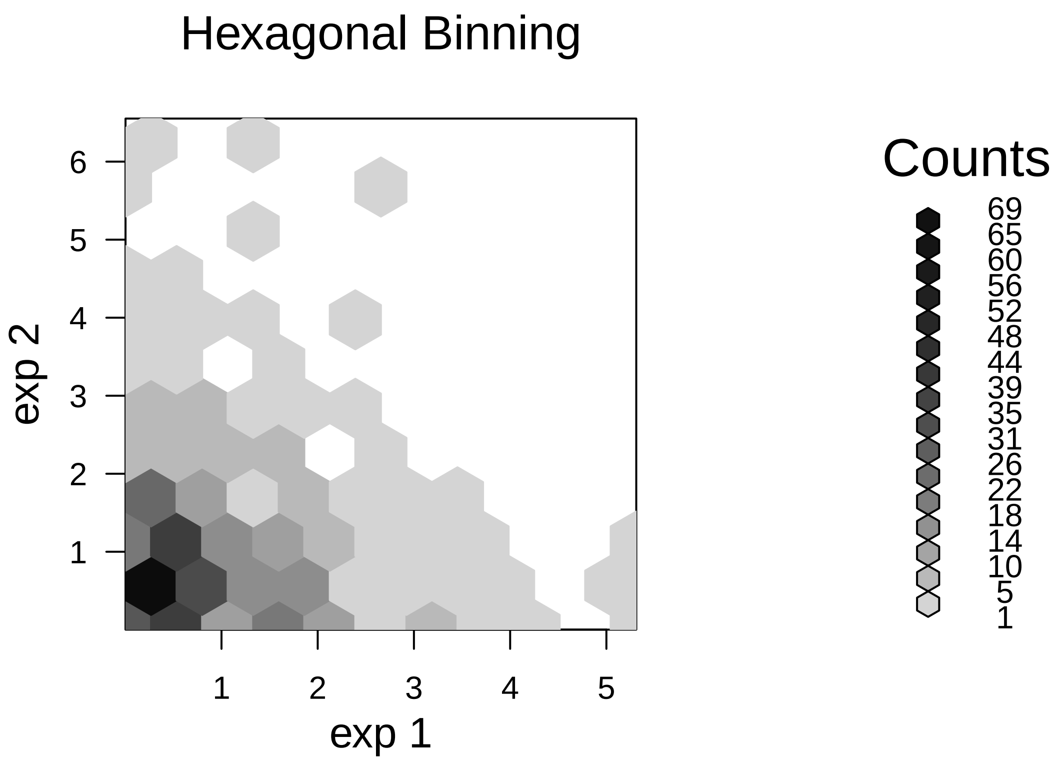
and visualize their scatterplot

```r
plot(df_bivariate_exp[,1],df_bivariate_exp[,2], xlab="exp 1", ylab="exp 2")
```



We can further employ a hexagonal binning plot to visualize the data density

```r
bin=hexbin(df_bivariate_exp[,1],df_bivariate_exp[,2], xbins=10, xlab="exp 1", ylab="exp 2")
plot(bin, main="Hexagonal Binning")
```



Depth measures (multivariate)

Now on depths: there are many possible packages hosted on CRAN to work with depths in a multivariate setting. We will hereafter use `DepthProc`, which is a good general purpose package and in addition it has good plotting capabilities. Even though not all depth measures introduced in class are directly available (simplicial, Oja, convex hull peeling depth), it includes the two depth measures we will be using throughout this section, namely **Tukey** and **Mahalanobis** depths. Let us look at the help file for the `depth` function.

Calculating depth for a given dataset is immediately accomplished by typing

```
tukey_depth=depth(u=df_bivariate_exp,method='Tukey')
```

It may be useful (spoiler alert, it will become apparent why when we study onparametric forecasting) to calculate the depth of a point relative to a sample. You can do it by:

```
depth(u = c(0, 0), X = df_bivariate_exp, method = 'Tukey')
```

```
## Depth method:  Tukey
## [1] 0
```

Compute the median (deepest point) with `depthMedian` function

```
depthMedian(df_bivariate_exp,depth_params = list(method='Tukey'))
```

```
## [1] 0.8364350 0.9297927
```

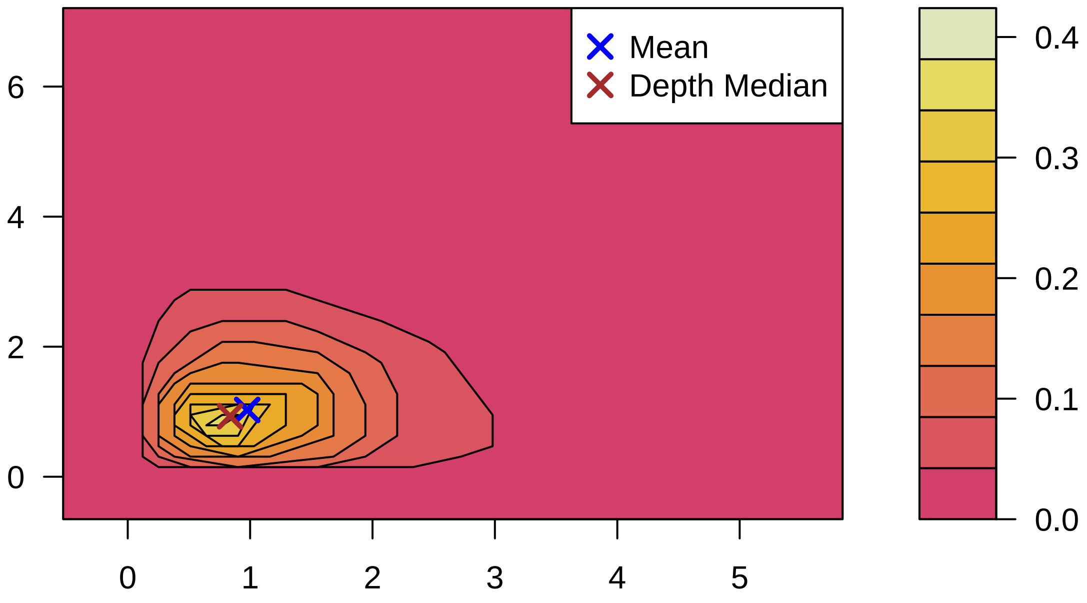
Or, having already computed the Tukey depth for the entire sample:

```
df_bivariate_exp[which.max(tukey_depth),]
```

```
## [1] 0.8364350 0.9297927
```

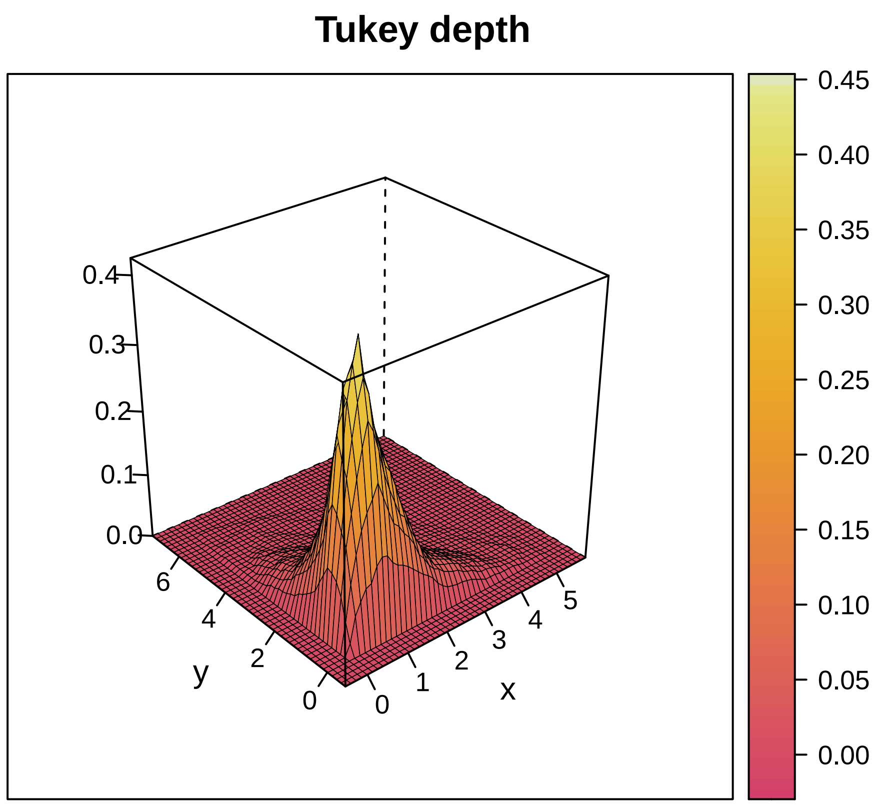
If you have the *luck* of dealing with a bivariate dataset, you can easily visualize the depth surface in a very convenient way. `DepthProc` offers you two possible methods:

```
depthContour(df_bivariate_exp,depth_params = list(method='Tukey'))
```



or

```
depthPersp(df_bivariate_exp,depth_params = list(method='Tukey'))
```



For additional special effects:

```r
depthPersp(df_bivariate_exp,depth_params = list(method='Tukey'),plot_method = 'rgl')
```

../../../../../../private/var/folders/7g/9rtz76hn4j5_ws3_z5kszxk40000gn/T/RtmpG0TYdV/filed12b25ba0de7.png

Computational caveat Since the Tukey depth is obtained by counting points in the space of all hyperplanes(surfaces) in $\mathbb{R}^d$, when $d > 2$, the computational burden increases. Libraries usually recur to heuristic methods which lead to poor estimates (we will see this in the functional data depth measures lab)

```r
a = depthPersp(df_bivariate_exp,depth_params = list(method='Tukey'))
```

The very same analysis can be carried out by considering a different depth measure:

```r
maha_depth <- depth(df_bivariate_exp,method='Mahalanobis')
```

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

$$M_h D(F; X^n) = \frac{1}{1 + (x - \mu_F)^T \Sigma^{-1}(x - \mu_F)}$$

```r
sample_mean <- colMeans(df_bivariate_exp)
sample_S <- cov(df_bivariate_exp)

maha_depth_manual <- 1/(1+mahalanobis(x = df_bivariate_exp,center = sample_mean,cov = sample_S))
```

And check that the obtained result is equal to the one obtained via the `depth` function (*beware of comparing floats*)

```r
all(abs(maha_depth-maha_depth_manual)<1e-15) # food for thought: sqrt(2) ^ 2 == 2?
```

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

```
depthMedian(df_bivariate_exp,depth_params = list(method='Mahalanobis'))
```

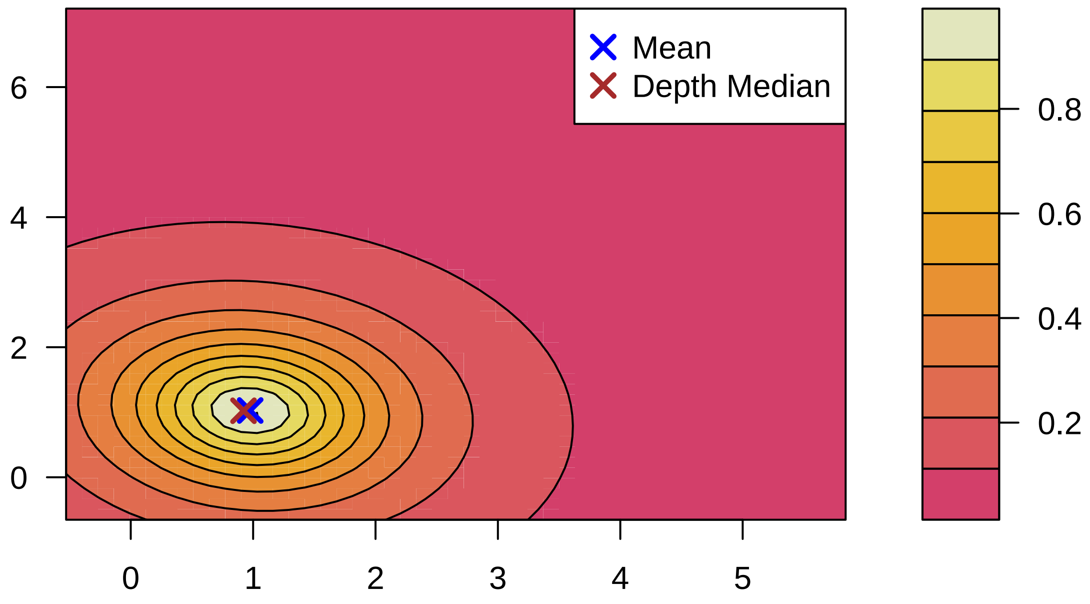
```
## [1] 0.9211997 1.0228290
```

```
df_bivariate_exp[which.max(maha_depth_manual),]
```

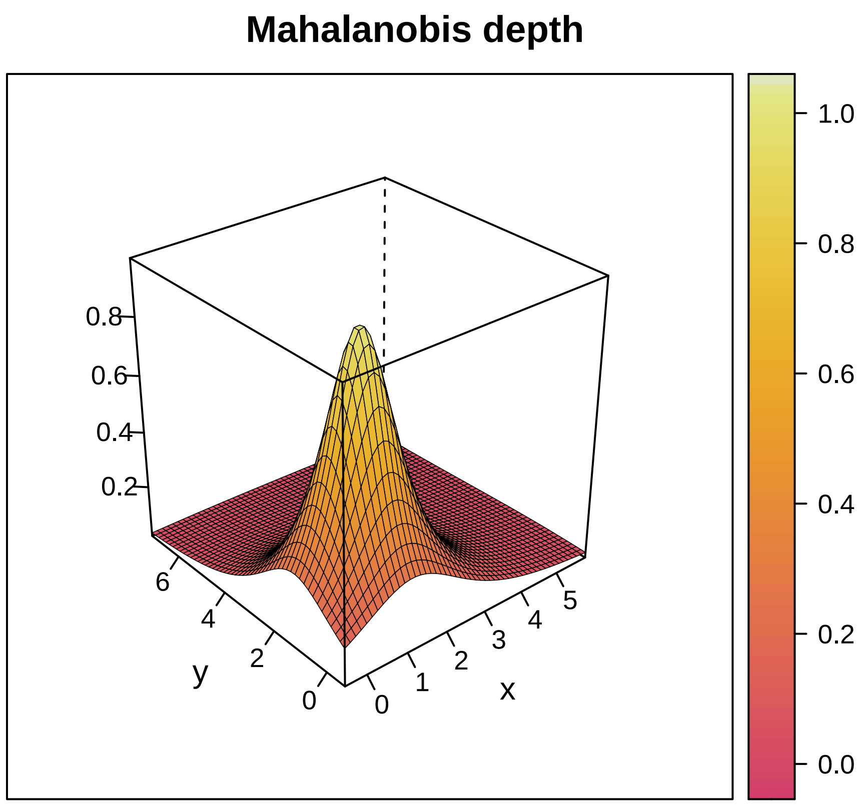
```
## [1] 0.9211997 1.0228290
```

And again the graphical outputs:

```
depthContour(df_bivariate_exp,depth_params = list(method='Mahalanobis'))
```



```
depthPersp(df_bivariate_exp,depth_params = list(method='Mahalanobis'))
```

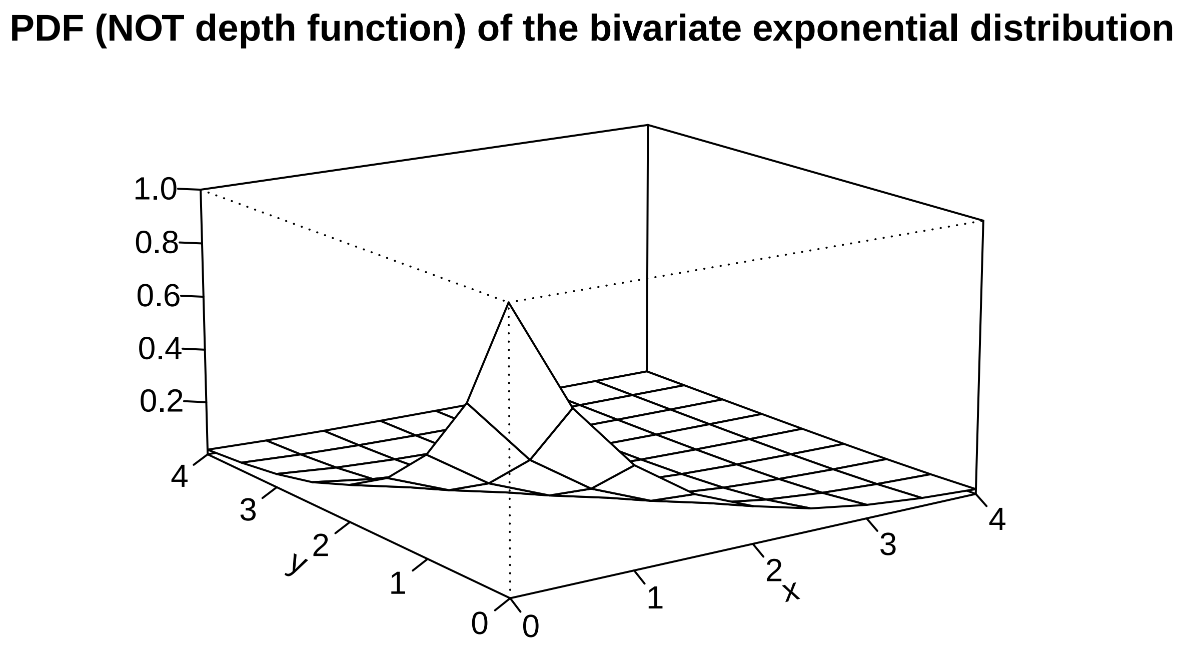


```
depthPersp(df_bivariate_exp,depth_params = list(method='Mahalanobis'),plot_method = 'rgl')
```

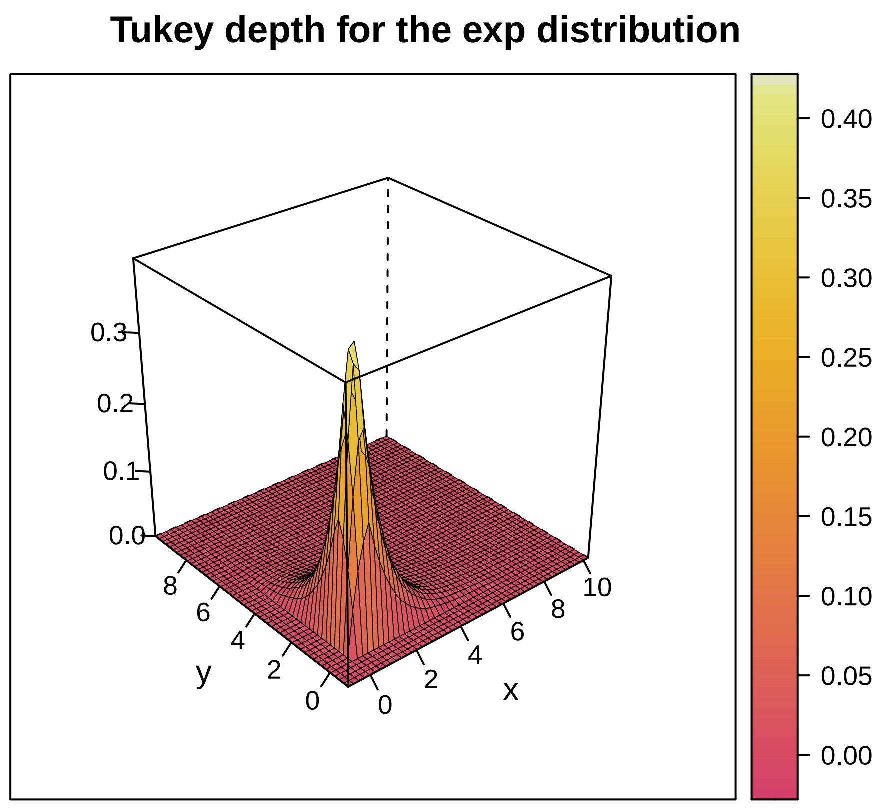
../../../../../../private/var/folders/7g/9rtz76hn4j5_ws3_z5kszxk40000gn/T/RtmpG0TYdV/filed12b6e24baf0.png

Please note that anything that comes out of `depthContour` or `depthPersp` is *NOT* a density: we are not doing density estimation here, we are performing data exploration by means of a nonparametric procedure. Let us plot the probability density function (NOT the depth function) of the bivariate exponential distribution we drew a sample from:

```
PDF_3dPlot(rho = 0, Betax = 1, Betay = 1, title="PDF (NOT depth function) of the bivariate exponential d
```



```r
depthPersp(cbind(rexp(1e4), rexp(1e4)),depth_params = list(method='Tukey'), plot_title="Tukey depth for
```



We notice that the tukey depth concentrates around the median, where as the exponential bivariate function increases as x,y go to 0.

Exercise

Try out the routines seen so far on something even more exotic:

```
set.seed(1992)
df_bivariate_cauchy = cbind(rcauchy(n,location=0,scale=.001), rcauchy(n,location = 0,scale=.001))
head(df_bivariate_cauchy)
```

```
##               [,1]         [,2]
## [1,] -0.0024184800 0.0008798105
## [2,] -0.0006282804 0.0009932694
## [3,] -0.0006975300 0.0010347342
## [4,] -0.0038080204 0.0141419885
## [5,]  0.0061098712 0.0014929490
## [6,] -0.0012696290 0.0008865863
```

The Cauchy distribution is a distribution that has neither the first moment (the mean) nor the second moment (the variance). This means that the CLT does not apply.

- Can we still perform non-parametric data exploration with depth measures?
- Would the Mahalanobis depth be a sensible measure to be used in this context? Why not?

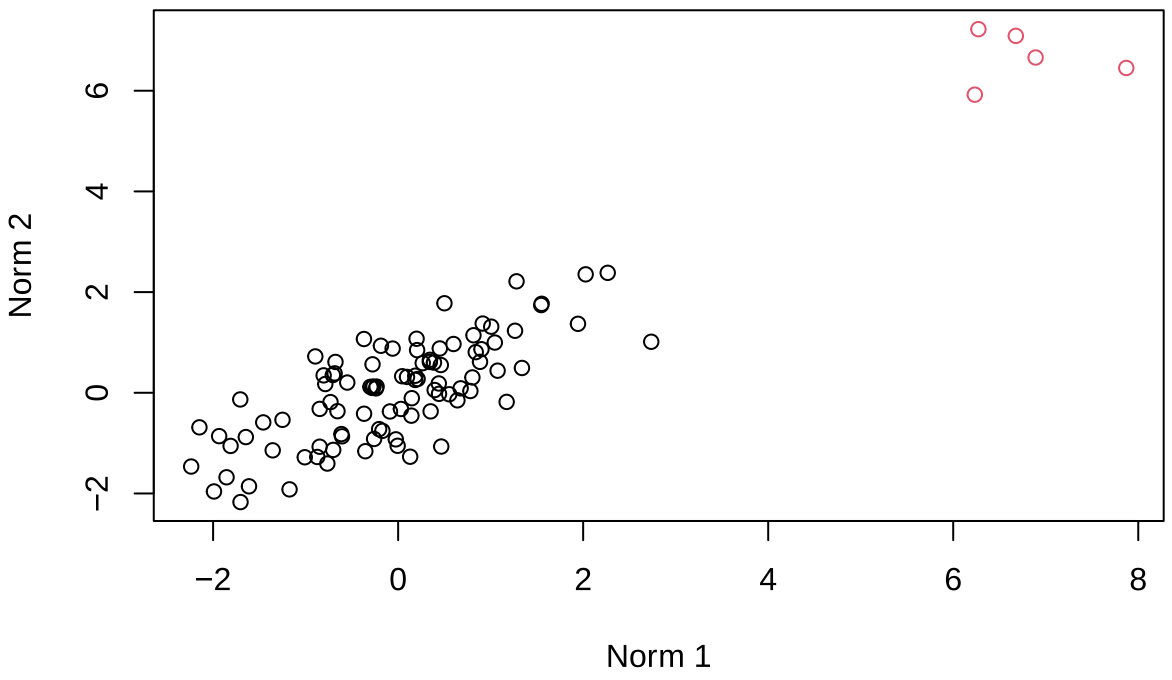
Multivariate outlier detection via depth measures

In the previous Section we have computed depth measures for a given dataset and we have seen how to effectively plot them. This resulted in a convenient way to nonparametrically explore a (bivariate) dataset. Nevertheless, one of the main aims for a statistician to employ depth measures is to perform multivariate outlier detection. To appreciate this, let us simulate 100 data points, of which 95% comes from a multivariate normal with mean vector `mu_good` and covariance matrix `sigma_common` and the other 5% from another multivariate normal, which we assume is our outlier generator process.

```
mu_good = c(0,0)
mu_outliers = c(7,7)

sigma_common = matrix(c(1,.7,.7,1), ncol = 2)

frac = .05
n=100
# sample points
n_good=ceiling(n*(1-frac))
n_outliers=n-n_good
df_contaminated_normals = data.frame(rbind(
  mvrnorm(n_good, mu_good, sigma_common),
  mvrnorm(n_outliers, mu_outliers, sigma_common)
))
```

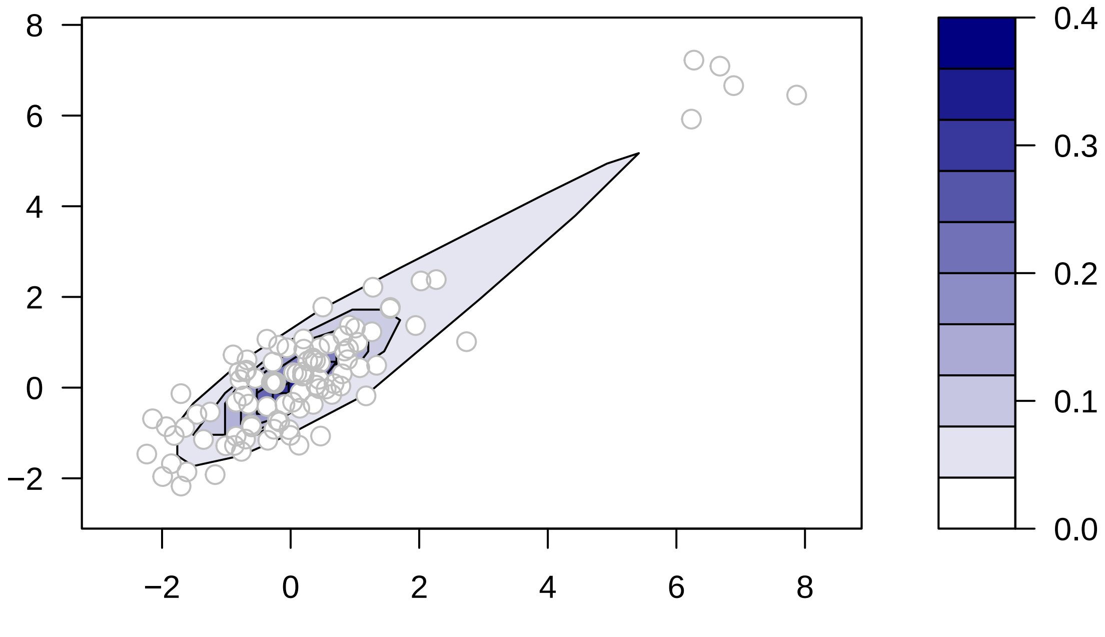
Let us visualize the true nature of our dataset

```
class <- c(rep(1,n_good),rep(2,n_outliers))
plot(df_contaminated_normals,xlab="Norm 1", ylab="Norm 2",col=class)
```



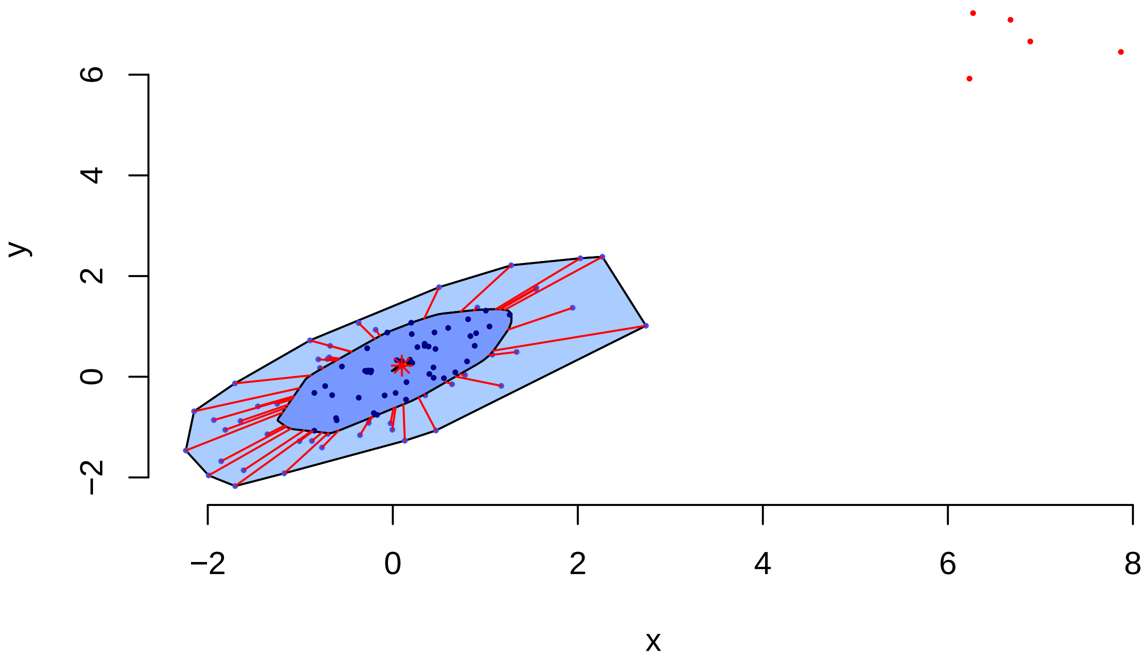
We can clearly see those red points up there… how can we flag them in an automated fashion? The depth contour plot surely helps

```r
depthContour(
  df_contaminated_normals,
  depth_params = list(method = 'Tukey'),
  points = TRUE,
  colors = colorRampPalette(c('white', 'navy')),
  levels = 10,
  pdmedian = F,
  graph_params = list(cex=.01, pch=1),
  pmean = F
)
```



But, as we have seen in class, a very handy graphical tool for spotting multivariate outliers is the bagplot. The `bagplot` function from the `aplpack` package can be used to display bagplots for bivariate data

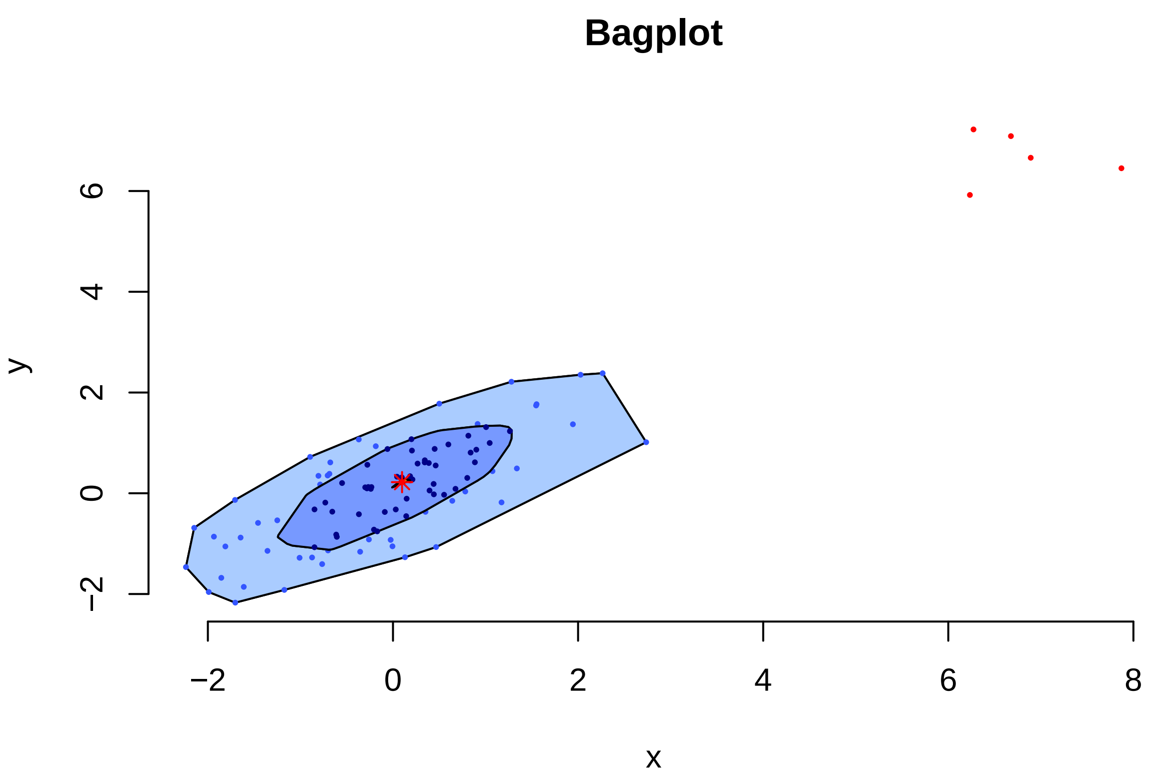
```
bagplot(df_contaminated_normals, factor = 3, show.whiskers = T)
```



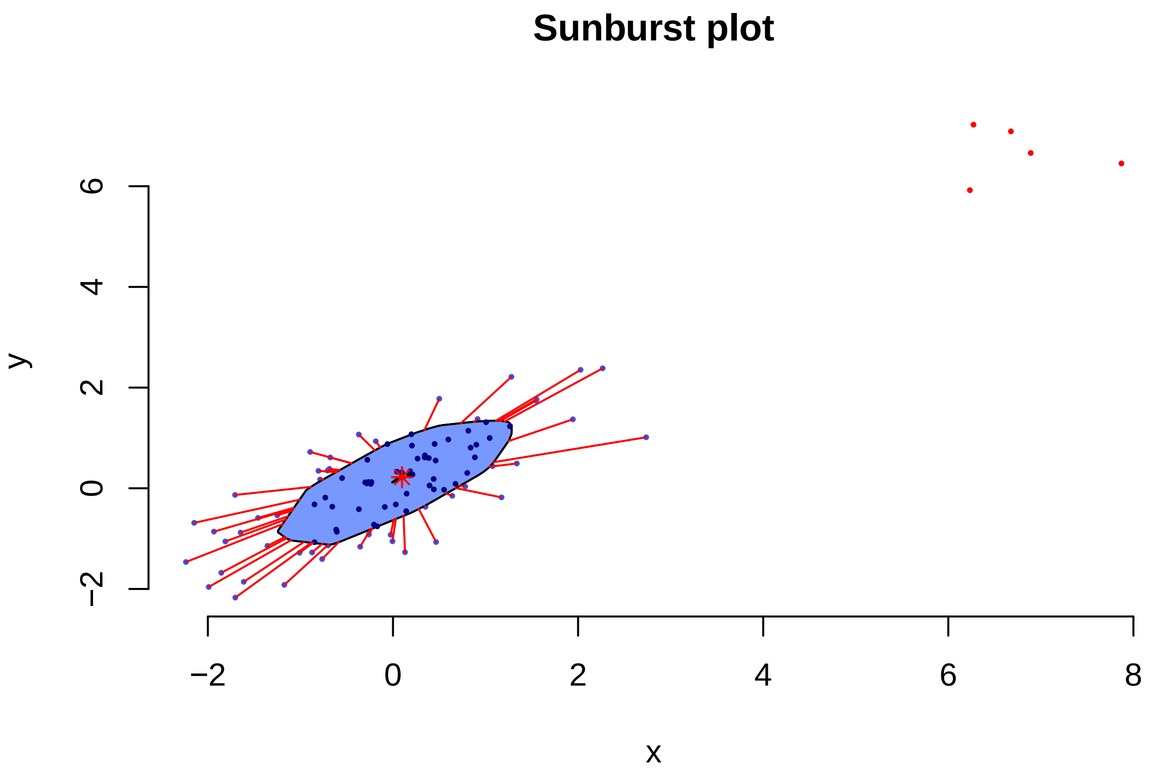
Remember:

- The *bag* is composed of the deepest 50% of the data (i.e. $C_{.5}$)
- The *fence* separates the outliers from the other observations
- The *loop* contains data between the bag and the fence
- Naturally, the median (i.e. the deepest datum) is inside the bag. If we check the help page for the `bagplot` function, we see that we have a lot of room for customization:

```
aplpack::bagplot(df_contaminated_normals,show.whiskers = F,main="Bagplot")
```

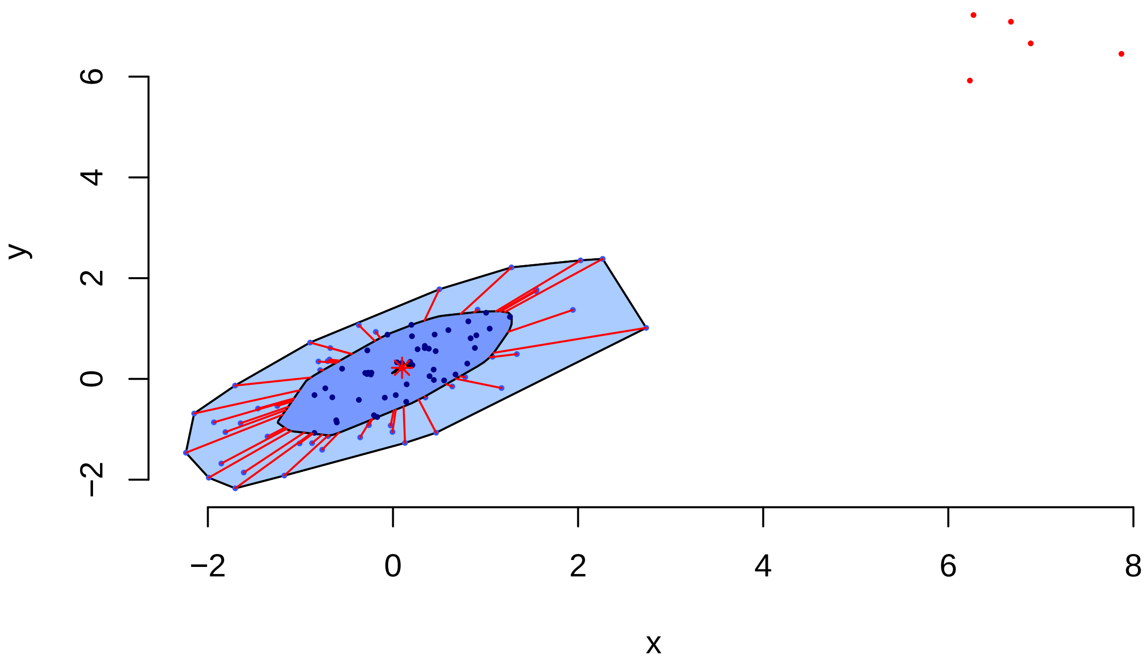


```r
aplpack::bagplot(df_contaminated_normals,show.loophull = F,main="Sunburst plot")
```



In addition, if we save the output of the bagplot to an object, we can automatically extract the outliers

```
bagplot_cont_normals <- bagplot(df_contaminated_normals)
```



```
outlying_obs <- bagplot_cont_normals$pxy.outlier
```

Once the outlying units have been identified, one can discard them from the original data and keep working on the clean subset only. There are several ways to do this.

A more "sql-oriented" approach (this is just a code alternative):

```
df_clean_1 <-
  dplyr::anti_join(
    x = df_contaminated_normals,
    y = outlying_obs,
    by = c("X1" = "x", "X2" = "y"),
    copy = TRUE
  )
```

A more "object-oriented programming" approach:

```
ind_outlying_obs <- which(apply(df_contaminated_normals,1,function(x) all(x %in% outlying_obs)))
df_clean_2 <- df_contaminated_normals[-ind_outlying_obs,]
```

```
all.equal(df_clean_1,df_clean_2)
```

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

Outlier detection in Star Cluster CYG OB1 data

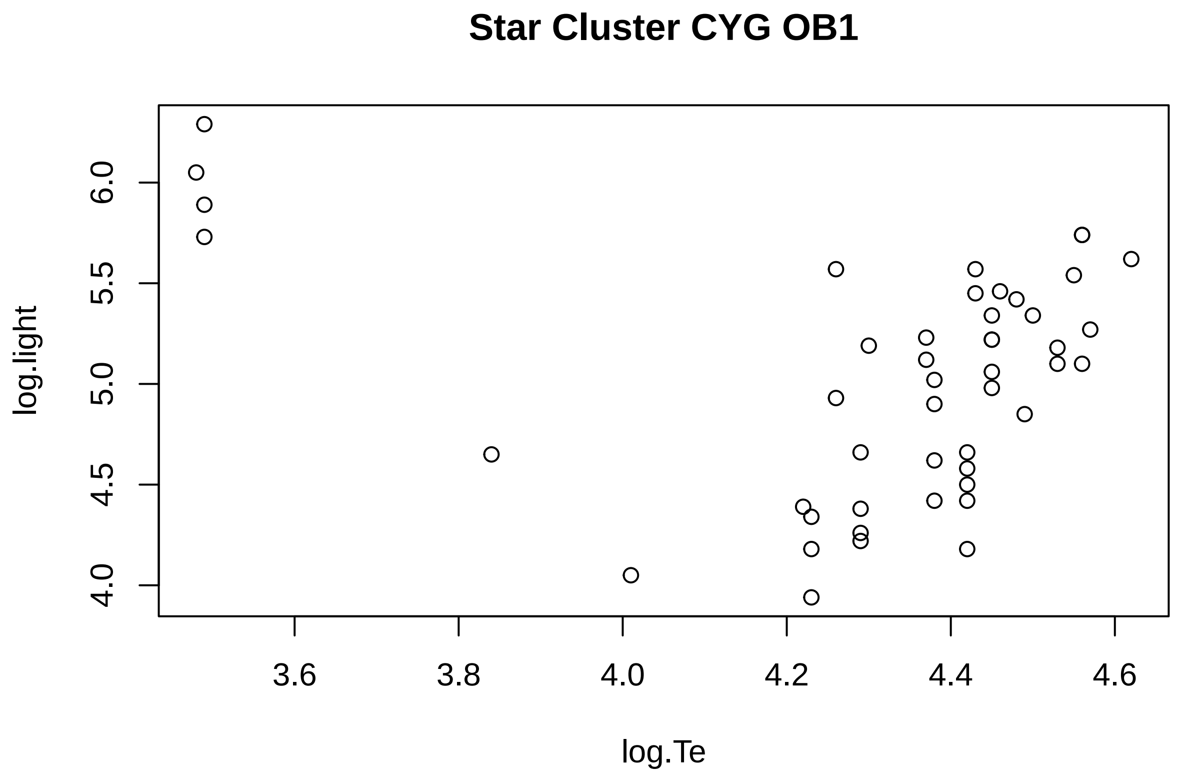
Data for the Hertzsprung-Russell Diagram of the Star Cluster CYG OB1, which contains 47 stars in the direction of Cygnus, from C.Doom.

- The first variable is the logarithm of the effective temperature at the surface of the star (Te)
- The second one is the logarithm of its light intensity (L/L0). The Hertzsprung-Russell diagram is the scatterplot of these data points, where the log temperature is plotted from left to right.

```
data(starsCYG, package = "robustbase")
names(starsCYG)
```

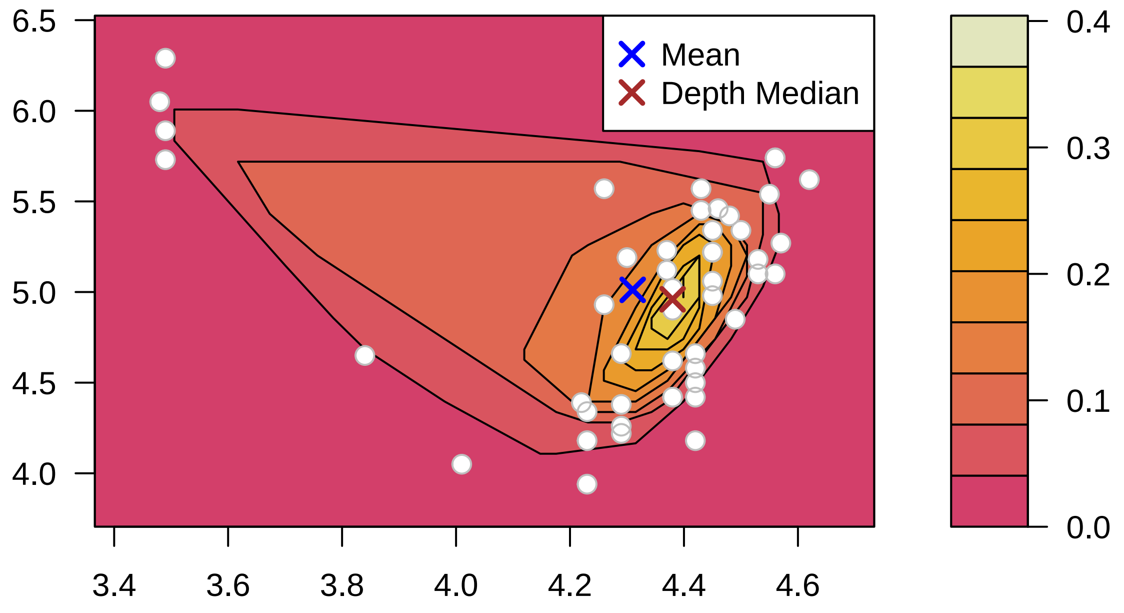
```
## [1] "log.Te"    "log.light"
```

```
plot(starsCYG, main="Star Cluster CYG OB1")
```



We can see two groups of points: the majority which tends to follow a steep band, the so called Main Sequence, and four stars in the upper-left corner. In astronomy the 43 stars are said to lie on the Main sequence and the four remaining stars are the red giants, namely points with indexes 11, 20, 30 and 34. In details, the red giants are very bright, but they emit light with a very low color-temperature (and thus their surface temperature is still fairly low). We can easily isolate them thanks to the procedure seen so far. Let us look at the perspective plot of the depth surface

```
depthContour(as.matrix(starsCYG), depth_params = list(method='Tukey'), points=TRUE)
```



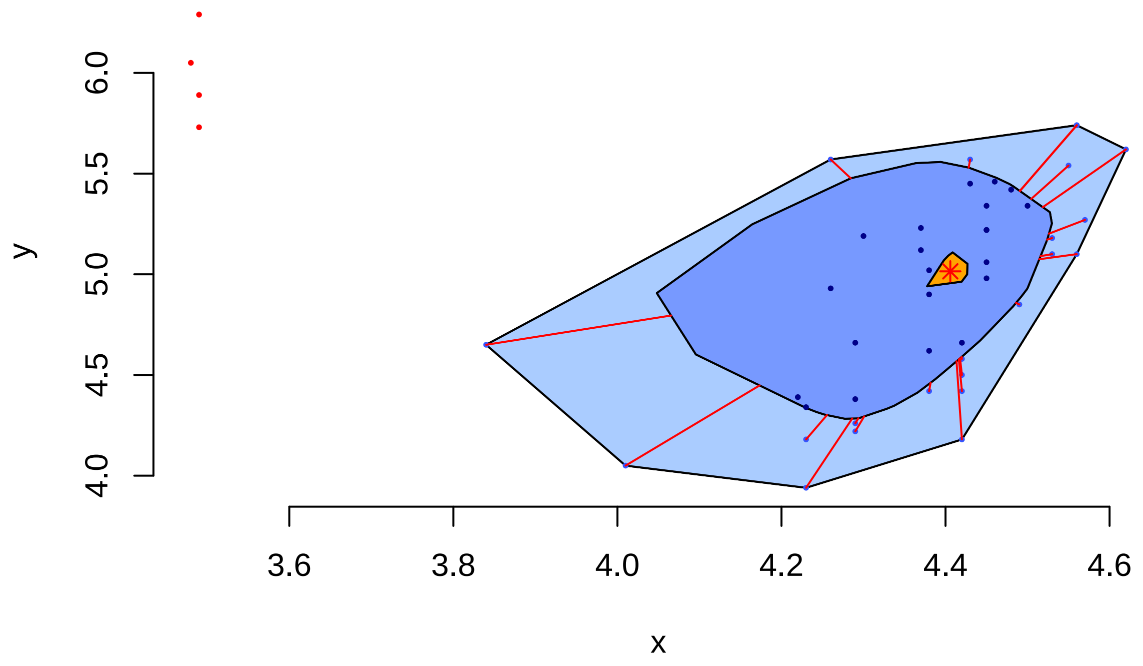
As you can see, the sample mean is biased due to the presence of the 4 red giants, whereas the Tukey Median is not. Let us compute it:

```r
depthMedian(starsCYG)
```

```
##    log.Te log.light
##      4.38      5.02
```

As before, we can use the bagplot to visualize and flag the outlying data points.

```r
bagplot_starsCYG <- with(starsCYG,aplpack::bagplot(log.Te,log.light))
```



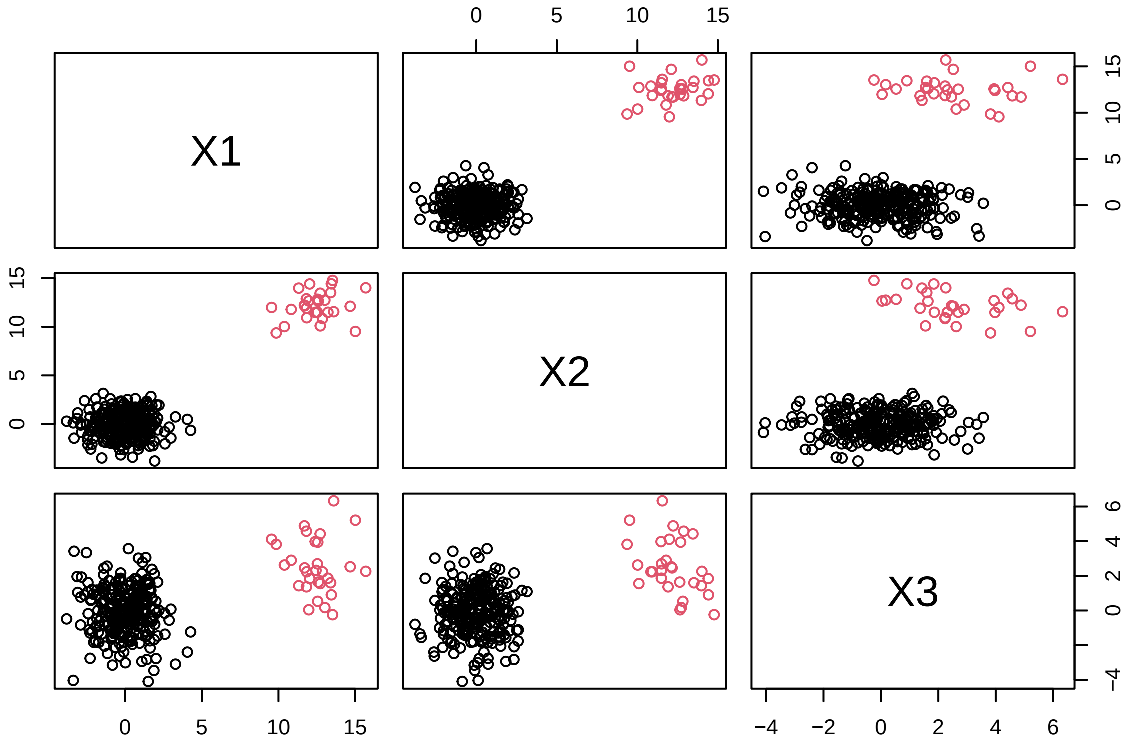
```r
red_giants <- bagplot_starsCYG$pxy.outlier
ind_outlying_obs <- which(apply(starsCYG,1,function(x) all(x %in% red_giants)))
ind_outlying_obs
```

```
## [1] 11 20 30 34
```

Moving beyond R^2

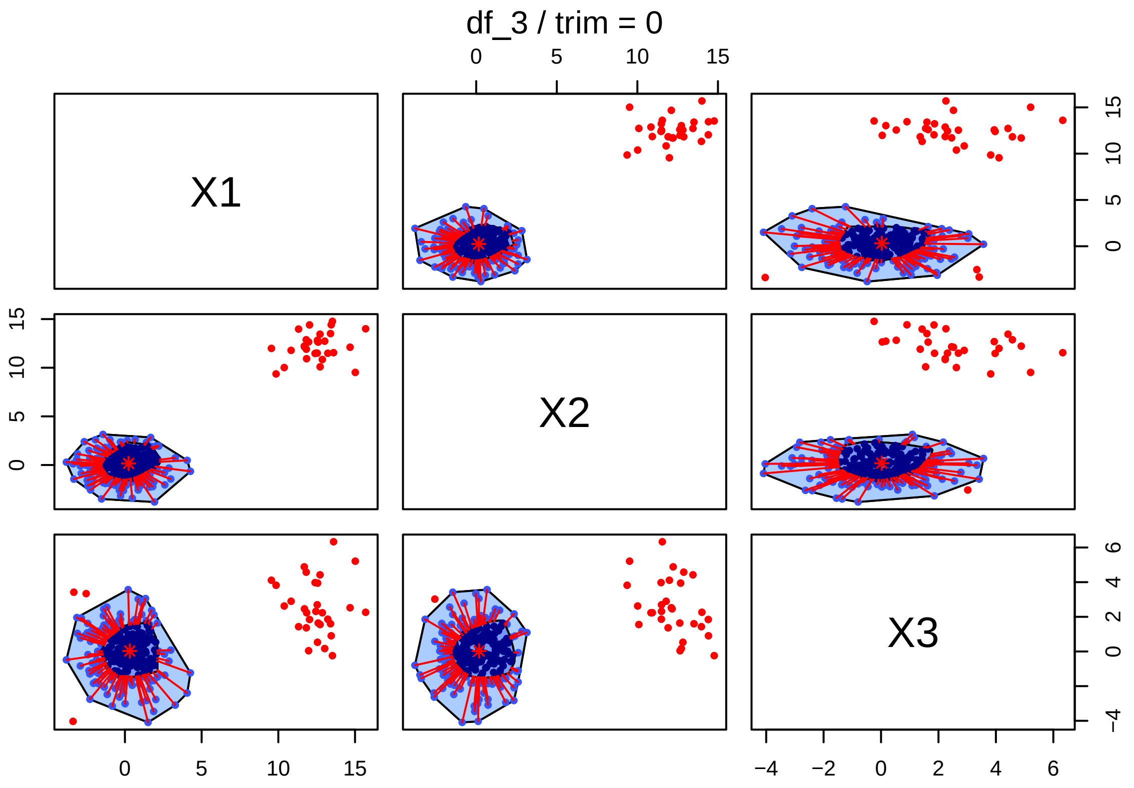
So far, we have only dealt with bivariate data. Even though the computational complexity escalates quickly when it comes to compute depth measures in R^d, with $d > 2$, we can still appreciate their usefulness in moderate dimension. Let us generate a trivariate dataset with contamination.

```
mu_good = rep(0,3)
mu_outliers = c(12,12,3)

sigma_common = diag(3)*2

frac = .1
n=300
# sample points
n_good=ceiling(n*(1-frac))
n_outliers=n-n_good
df_3 = data.frame(rbind(
  mvrnorm(n_good, mu_good, sigma_common),
  mvrnorm(n_outliers, mu_outliers, sigma_common)
))
class <- c(rep(1,n_good),rep(2,n_outliers))
pairs(df_3, col=class)
```



To visualize the outliers in this context we retort to a bagplot matrix:

```r
bagplot_matrix <- aplpack::bagplot.pairs(df_3)
```



You can notice that outlier detection becomes more difficult when the dimension increases, as

- some outliers may be wrongly flagged as genuine points
- some good points may be wrongly flagged as outliers

These phenomena are respectively denoted as **masking** and **swamping**, we will cover it in details during the robust statistics module of this course!

Let us conclude this lab session by looking at some DD-plots. For two probability distributions F and G , both in R^d, and $D(\cdot)$ an affine-invariant depth, we can define depth vs. depth plot being very useful generalization of the one dimensional quantile-quantile plot:

$$DD(F,G) = \left\{(D_F(x), D_G(x)) \text{ for all } x \in \mathbb{R}^d\right\}$$

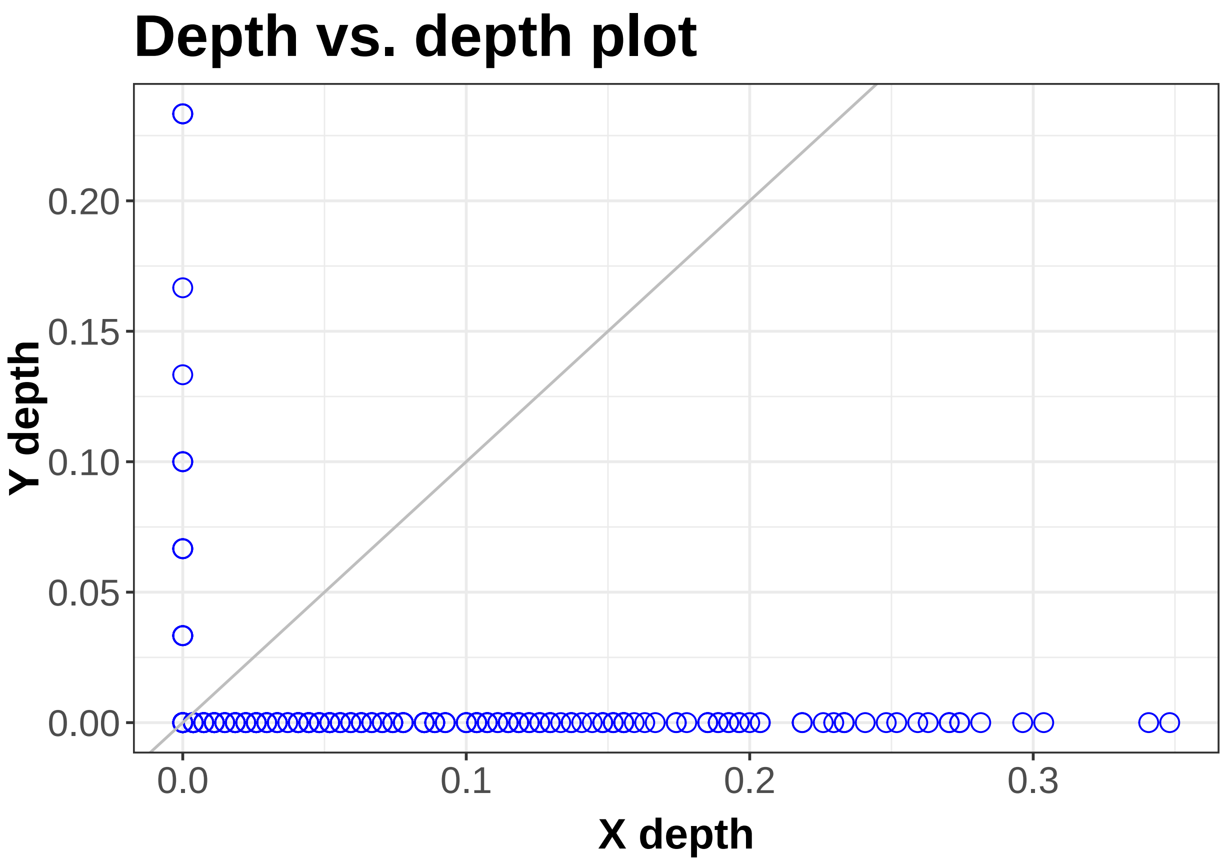
Its sample counterpart calculated for two samples $\mathbf{X} = \{X_1, \cdots, X_n\}$ from F, and $\mathbf{Y} = \{Y_1, \cdots, Y_m\}$ from G is defined as

$$DD\left(F_n, G_m\right) = \left\{\left(D_{F_n}(z), D_{G_m}(z)\right), z \in \{\mathbf{X} \cup \mathbf{Y}\}\right\}$$

The `ddPlot` function automatically computes and plots it:

```r
df_good <- df_3[1:n_good,]
df_out <- df_3[(n_good+1):n,]
ddPlot(x = df_good,y = df_out,depth_params = list(method='Tukey'))
```

```
## DDPlot
```

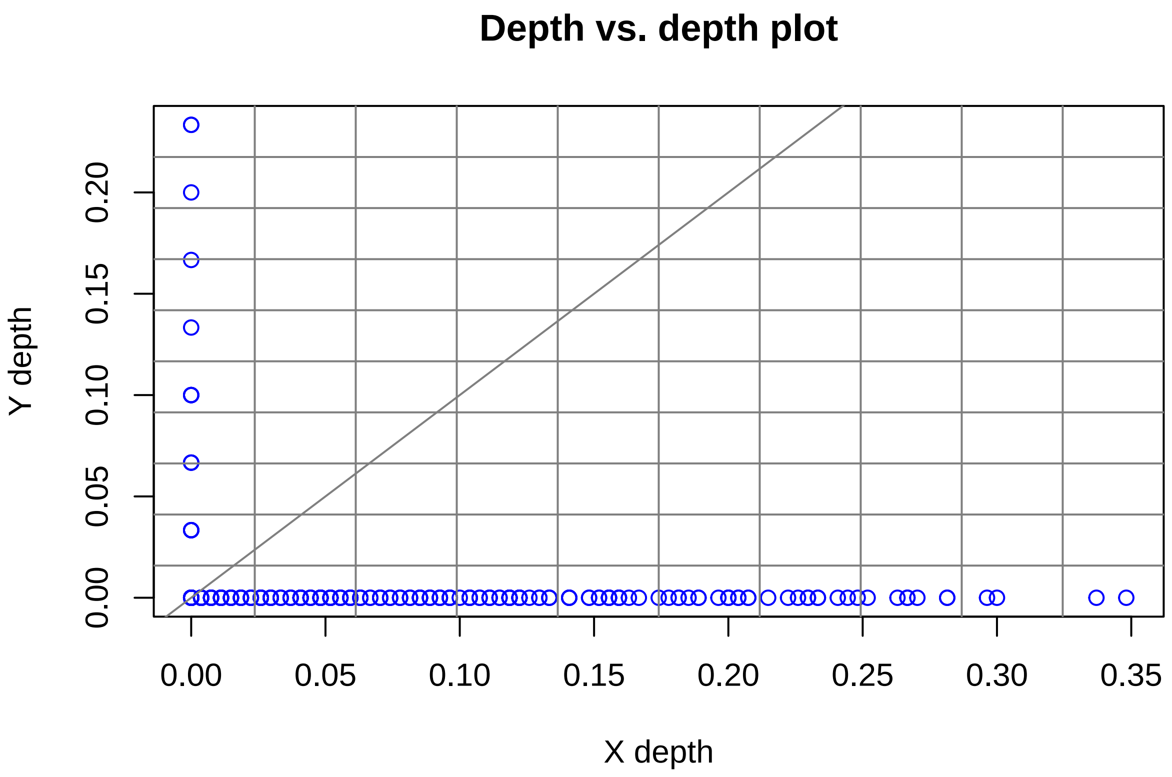


```
##
## Depth Metohod:
##    Tukey
```

They would line up if they came from the same distribution

It is easy to manually build a DD-plot

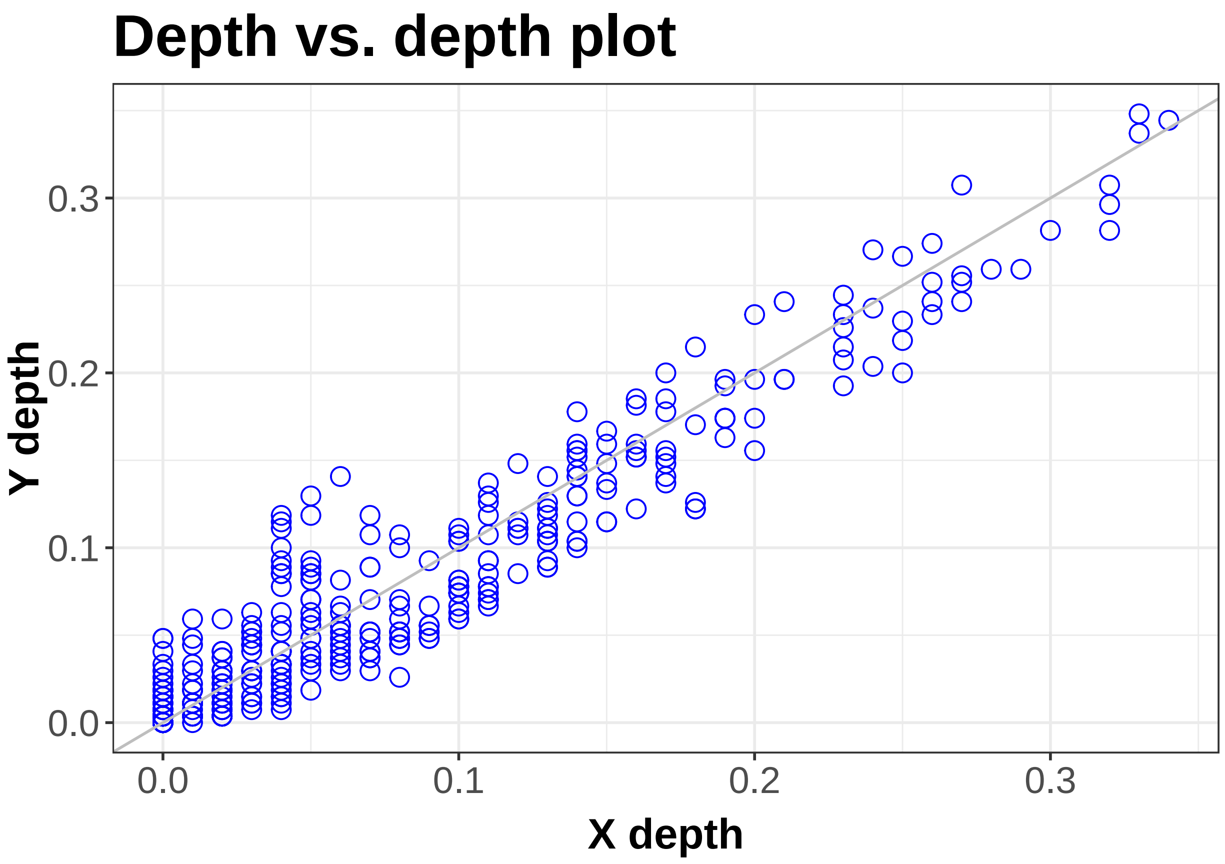
```
depth_good <- depth(u = df_3,X = df_good,method = "Tukey")
depth_out <- depth(u = df_3,X = df_out,method = "Tukey")
plot(depth_good,depth_out, col="blue", xlab="X depth", ylab="Y depth", main= "Depth vs. depth plot")
grid(10, 10, col="grey50", lty=1)
abline(0,1, col="grey50")
```



Clearly the distributions are not the same. . . What if we had some extra samples coming from F (e.g., the trivariate normal centered at 0)?

```
n_extra <- 100
df_extra <- data.frame(mvrnorm(n_extra, mu_good, sigma_common))
ddPlot(x = df_extra, df_good,depth_params=list(method='Tukey'))
```

```
## DDPlot
```



```
##
## Depth Metohod:
##    Tukey
```

Indeed this time we can conclude the same distribution generated the data.