```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### DEPTH MEASURES
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# RUN THE NEXT CODE ONLY ONCE!
# pack_list=c('aplpack','ddalpha','rgl','MASS','DepthProc')
# install.packages(pack_list)
library(MASS)
```

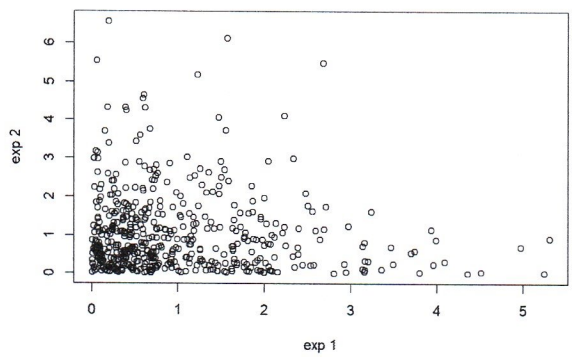
```
library(rgl)
```

```
library(DepthProc)
```

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### EXAMPLE 1 - Some general ideas
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# Simulate 500 bivariate datapoints whose distribution is marginally exponential
set.seed(2781991)
n          = 500
mixbivexp = cbind(rexp(n), rexp(n))
head(mixbivexp)
```

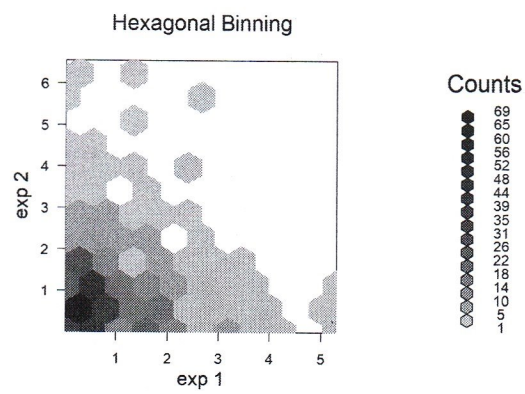
```
##           [,1]       [,2]
## [1,] 0.3004277 0.35894610
## [2,] 2.4430839 0.22392999
## [3,] 1.8444128 0.15064919
## [4,] 0.2383688 1.12290176
## [5,] 1.9613656 0.07966151
## [6,] 5.2480885 0.01762207
```

```
# Visualize a scatterplot of the data
plot(mixbivexp[,1],mixbivexp[,2], xlab="exp 1", ylab="exp 2")
```




```
# Let's try to start to have an idea of the density of this data, starting with a hexagonal binning
library(hexbin)
```

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



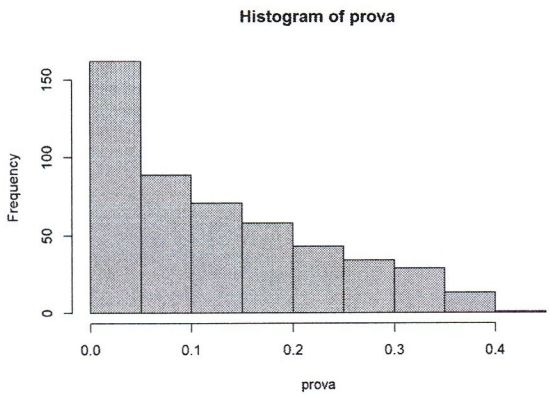
```
### Compute depths
# There are many possible packages on CRAN to work with depths in a multivariate setting:
# some do implement different depths, some others are faster, some again have better plotting
# facilities. I have chosen to use DepthProc, which I think is a good general purpose package,
# and has the best plotting of them all. You cannot find some depths, though (simplicial depth
# is not available, for instance). In any case, it just requires some programming to "import"
# additional depths in your favourite package.
help(depth)
```

In the package "**depth**" we also have **simplicial**/Liu depth and **Oja** depth, which are not present in "DepthProc"

```
prova = depth(mixbivexp,method='Tukey')
head(prova)
```

```
## [1] 0.140 0.050 0.074 0.172 0.038 0.000
```

```
hist(prova)
```




```
# It may be useful (spoiler alert, it'll be apparent why when we'll talk about nonparametric
# forecasting) to calculate the depth of a point relative to a sample
depth(c(0,0),mixbivexp,method='Tukey')
```

depth of the point (0,0) relative to the sample

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

```
depth(c(5,5),mixbivexp,method='Tukey')
```

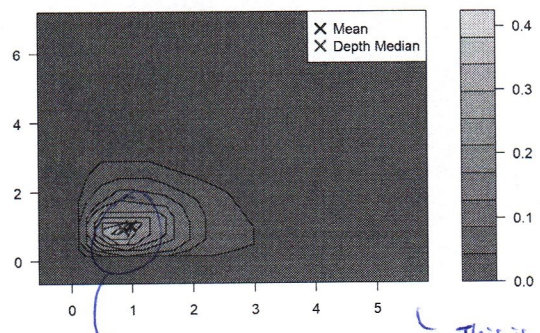
```
## Depth method:  Tukey
## [1] 0
```

```
# Median (deepest point)
help("depthMedian")
depthMedian(mixbivexp,depth_params = list(method='Tukey'))
```

this works for every-dimension dataset: if we put a 4 dimension data the output will be the median (4 values)

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

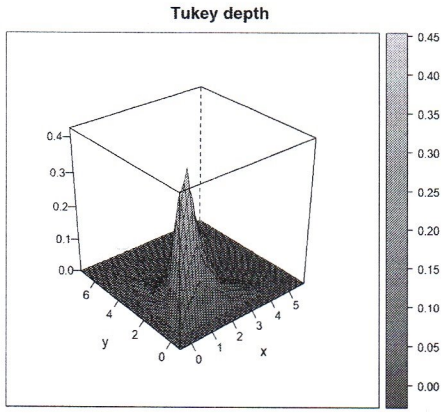
```
# default=Tukey (halfspace) depth, other options are available

# If you have the luck of having a bivariate dataset, you can easily visualize the depth surface
# in a very convenient way. DepthProc offers you two possible methods:
help('depthContour')
depthContour(mixbivexp,depth_params = list(method='Tukey'))
```



mean and median are **not** the same

This is equivalent to a boxplot: it does not have any forecasting or probabilistic meaning (IT'S **NOT A DENSITY**)
These are just the α regions: the region in space is determine as a union of points for which the depth (sample depth) is above a certain threshold.

```
# or
help('depthPersp')
depthPersp(mixbivexp,depth_params = list(method='Tukey'))
```




```
# For additional special effects:
depthPersp(mixbivexp,depth_params = list(method='Tukey'),plot_method = 'rgl')

# What if I want to try a different depth?
depth(c(0,0),mixbivexp,method='Mahalanobis')
```

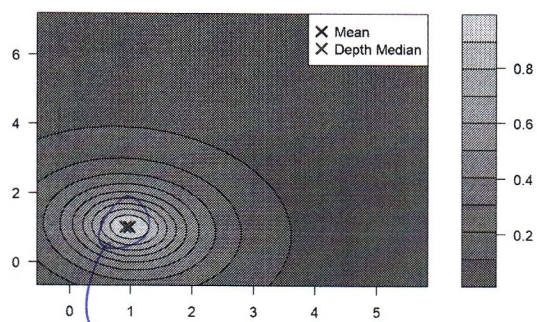
we're estimating a variance-covariance matrix that does not have any sense (*)

```
## Depth method:  Mahalanobis
## [1] 0.3048259
```

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

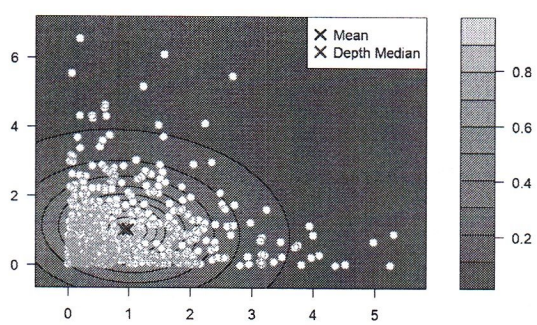
```
## [1] 0.9211997 1.0228290
```

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



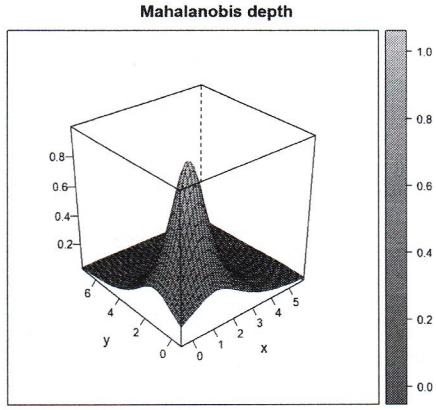
mean and median are closer (w.r.t. 'Tukey')

```
depthContour(mixbivexp,depth_params = list(method='Mahalanobis'), points=T)
```

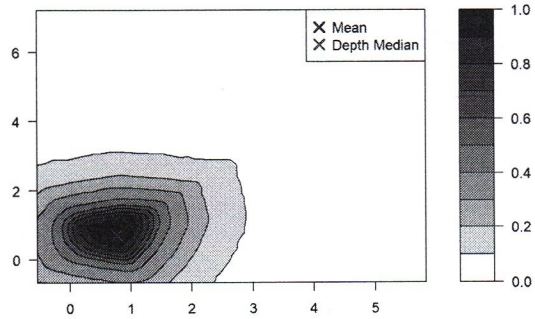


(*) the isolines do not underly the correct distribution of the points!
(The shape of the isolines don't make any sense)

```
# or
depthPersp(mixbivexp,depth_params = list(method='Mahalanobis'))
```




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

# Please note that ANYTHING that comes out of depthContour or depthPersp is NOT a density.
# The topic of bivariate density forecasting is an open (and very interesting one).
source('depthPredictiveContour.R')
depthPredictiveContour(mixbivexp) # we are using the projection depth, way faster
```

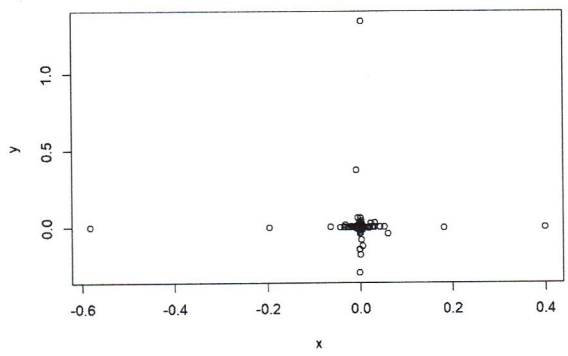


These isolines HAVE a PROBABILISTIC MEANING. !!
The whole region has a 90% coverage property in forecasting.
(with a 90% probability, a new observation will fall in the region)

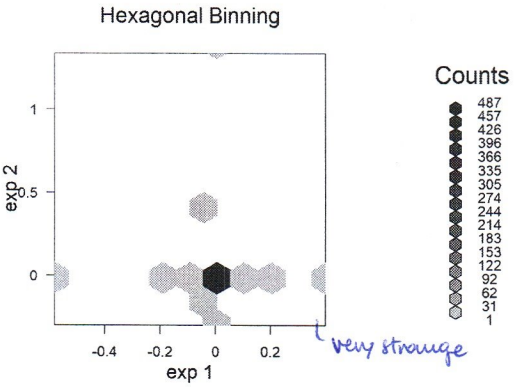
```
# Let's also try with something even more exotic:
set.seed(1992)
mixcauchy = cbind(rcauchy(n,location=0,scale=.001), rcauchy(n,location = 0,scale=.001))
head(mixcauchy)
```

```
##               [,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
```

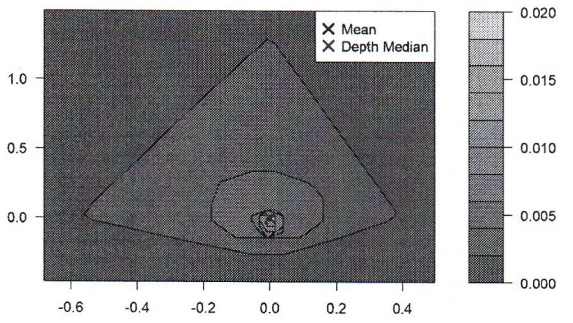
```
# The cauchy distribution is an example of a distibution that does not have either the 1-moment
# (the mean) nor the 2-moment (the variance). This means that the CLT does not apply.
# Visualize a scatterplot of the data:
plot(mixcauchy[,1],mixcauchy[,2], xlab="x", ylab="y")
```



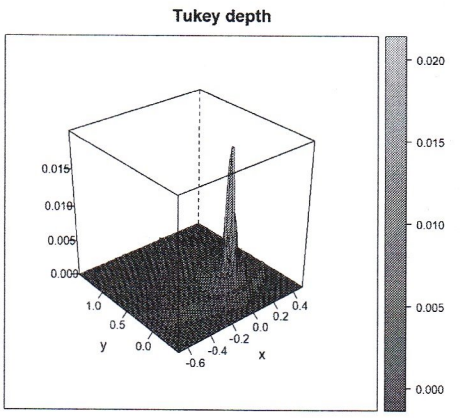
```
# Let's try to start to have an idea of the density of this data, starting with a hexagonal binning
library(hexbin)
bin = hexbin(mixcauchy[,1],mixcauchy[,2], xbins=10, xlab="exp 1", ylab="exp 2")
plot(bin, main="Hexagonal Binning")
```




```
# BUT! You can of course use a depth estimator of the median (which, instead, is actually defined),
# and thus get an idea of the central tendency of this distribution.
depthContour(mixcauchy,depth_params = list(method='Tukey'))
```




```
# or
depthPersp(mixcauchy,depth_params = list(method='Tukey'),plot_method = 'rgl')

# For additional special effects:
depthPersp(mixcauchy,depth_params = list(method='Tukey'))
```



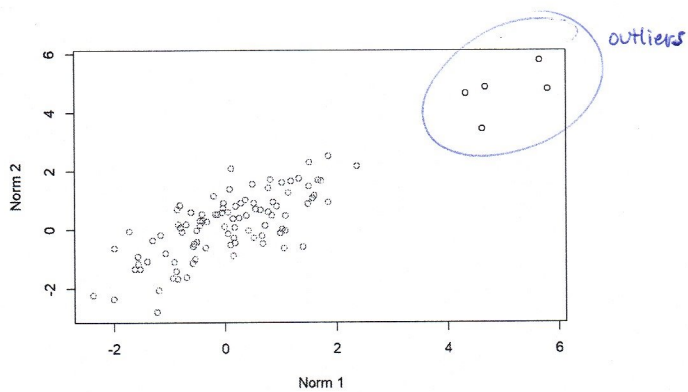
```
### --------------------------------------------------------------------------------
### --------------------------------------------------------------------------------
### EXAMPLE 2 - Multivariate outlier detection
### --------------------------------------------------------------------------------
### --------------------------------------------------------------------------------
# In class we have argued about the fact that depth measures allow us to port some
# very useful order statistics concepts in the multivariate setting. To see this,
# let's generate a dataset with outliers. Let's simulate 100 items, of which 95%
# from a multivariate normal (mu1, sigma) and the other 5% from another multivariate
# normal, which we assume is our outlier generator process.

# Define mu and sigma
mu1   = c(0,0)
mu2   = c(5,5)
sigma = matrix(c(1,.7,.7,1), nc = 2)
sigma
```

```
##      [,1] [,2]
## [1,]  1.0  0.7
## [2,]  0.7  1.0
```

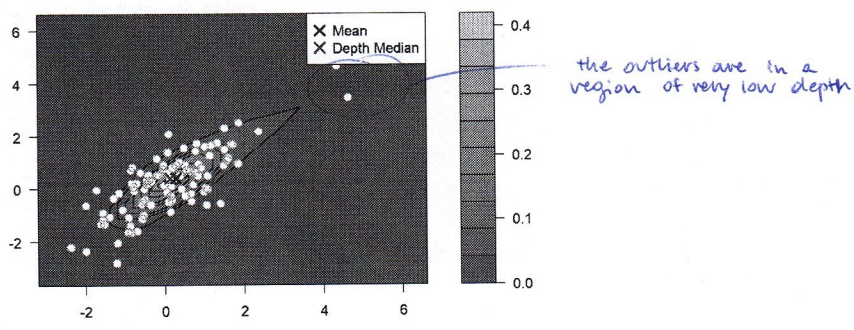
```
frac  = .05
n     = 100

# Sample points
n1          = ceiling(n*(1-frac))
n2          = n-n1
mixbivnorm = rbind(mvrnorm(n1, mu1, sigma), mvrnorm(n2, mu2, sigma))
head(mixbivnorm)
```

```
##             [,1]       [,2]
## [1,]  1.32109899  1.7428466
## [2,]  0.51253175  0.9014948
## [3,] -0.69979683  0.1930782
## [4,] -0.57096765 -0.5757005
## [5,] -0.03819337  0.7454022
## [6,] -0.53765358 -1.0034845
```

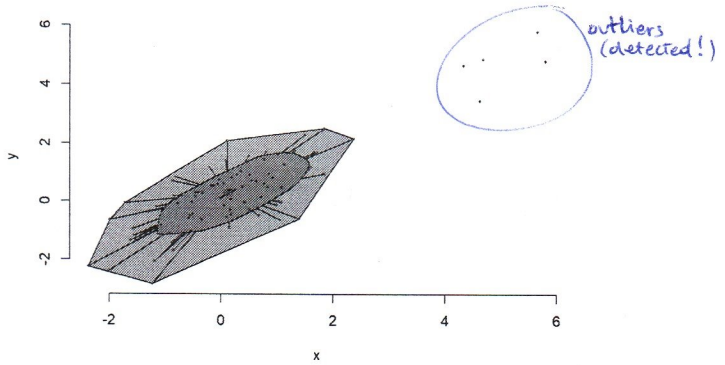
```
# Visualize a scatterplot of the data
color       = rep(1,n)
color[1:n1] = 2
plot(mixbivnorm[,1],mixbivnorm[,2], xlab="Norm 1", ylab="Norm 2",col=color)
```




```
# We can see those black points up there. How can I "spot" them in an automated fashion?
# Let's see the depth surfaces first.
depthContour(mixbivnorm,depth_params = list(method='Tukey'), points=T)
```



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

# The tool we use to spot multivariate outliers is by using the analogous of the boxplot: the BAGPLOT.
# Unfortunately for us, DepthProc doesn't have a built in way to draw them...
# so we need to use other packages, namely:
aplpack::bagplot(mixbivnorm)

# and, if you assign the result to something, you can actually store the outliers
bgplot = aplpack::bagplot(mixbivnorm)
```




```
bgplot$pxy.outlier
```

```
##             x        y
## [1,] 4.626944 3.444359
## [2,] 5.653867 5.774866
## [3,] 5.798651 4.793524
## [4,] 4.684623 4.842170
## [5,] 4.333848 4.642481
```

```
### Application 1 - Outlier detection
# Data: starsCYG
# 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).
# In the Hertzsprung-Russell diagram, which is the scatterplot of these data points,
# where the log temperature is plotted from left to right, two groups of points are seen:
# the majority which tend to follow a steep band, the so called Main Sequence,
# and four stars in the upper corner. In astronomy the 43 stars are said to lie on the Main
# sequence and the four remaining stars are the red giants... (the points 11, 20, 30, 34).

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

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

red stars are older
blue stars are newer

```
head(starsCYG)
```

```
##   log.Te log.light
## 1   4.37      5.23
## 2   4.56      5.74
## 3   4.26      4.93
## 4   4.56      5.74
## 5   4.30      5.19
## 6   4.46      5.46
```

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

# Here you understand why "red giants" the four stars on the upper left corner of the plot
# are very very bright, but emit light with a very low color-temperature
# (and thus their surface temperature is still fairly low)

# Compute the depth of the bivariate mean wrt the sample
smean = apply(starsCYG,2,mean)
smean
```

```
##    log.Te log.light
##  4.310000  5.012128
```

```
colMeans(starsCYG)
```

```
##    log.Te log.light
##  4.310000  5.012128
```

```
smax = apply(starsCYG,1,max)
smax
```

```
##  [1] 5.23 5.74 4.93 5.74 5.19 5.46 4.65 5.27 5.57 5.12 5.73 5.45 5.42 4.05 4.29
## [16] 4.58 4.23 4.42 4.23 5.89 4.38 4.29 4.42 4.85 5.02 4.66 4.66 4.90 4.39 6.05
## [31] 4.42 5.10 5.22 6.29 4.34 5.62 5.10 5.22 5.18 5.57 4.62 5.06 5.34 5.34 5.54
## [46] 4.98 4.50
```

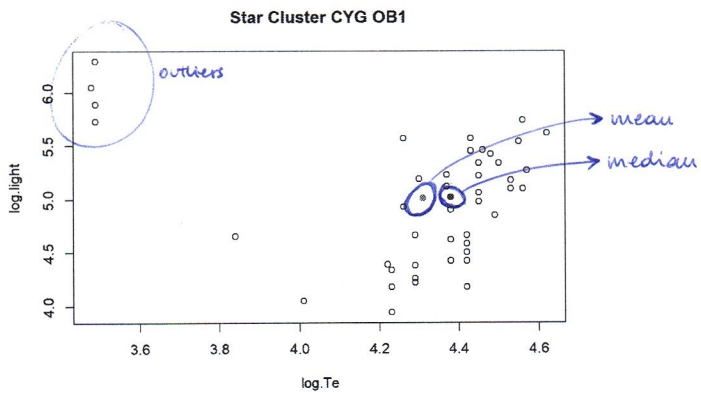
```
depth(smean, starsCYG, method = 'Tukey')
```

```
## Depth method:  Tukey
## [1] 0.1914894
```

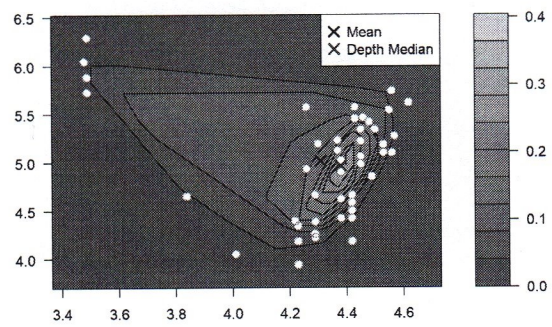
```
points(smean[1],smean[2], pch=19, col="green") #plot the barycenter

# Find the Tukey's median
smed = depthMedian(starsCYG)
smed
```

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

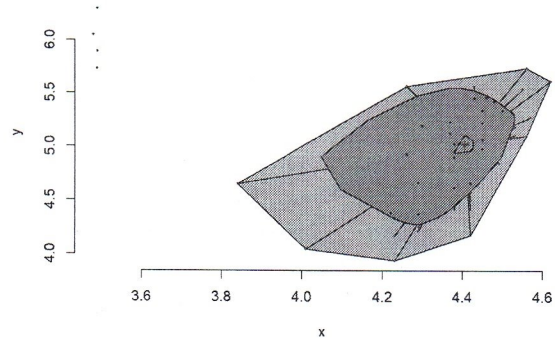
```
points(smed[1],smed[2], pch=19, col="red") #plot the median
```




```
# Perspective plot of the depth surface
depthContour(as.matrix(starsCYG), depth_params = list(method='Tukey'),points=T)
```




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

# Plot the bagplot
aplpack::bagplot(log.Te,log.light)
```




```
# The four stars in the upper left corner are outliers; they are the "giants" stars
detach(starsCYG)
```

```
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### EXAMPLE 3 - Computational Burden
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
# Simulated data (bivariate Normal)
x = matrix(rnorm(9999), nc = 3) #data in R^3

# Let's try to compute this depth with different methods (with different performance).
# Let's calculate the depth of each point of the dataset wrt to every other one.

depthwrapper = function(row){depth::depth(row,x)}
st1          = system.time(apply(x[1:10,], 1, depthwrapper)) #Let's clock it'
st1
```

```
##    user  system elapsed
##   18.30    0.07   18.62
```

```
# depth p
depthwrapper = function(row){depth::depth(row,x,approx=T)}
st2          = system.time(apply(x[1:10,], 1, depthwrapper))
st2
```

```
##    user  system elapsed
##    0.28    0.00    0.30
```

```
st1/st2 #second method is 60X faster
```

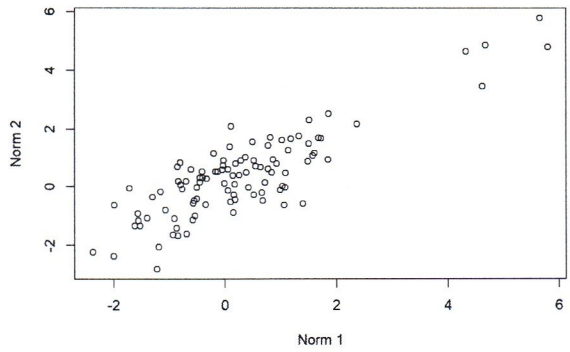
```
##     user   system  elapsed
## 65.35714      Inf 62.06667
```

```
# cpp implementation
st3 = system.time(depth(x[1:10,], x, method='Tukey'))
st3
```

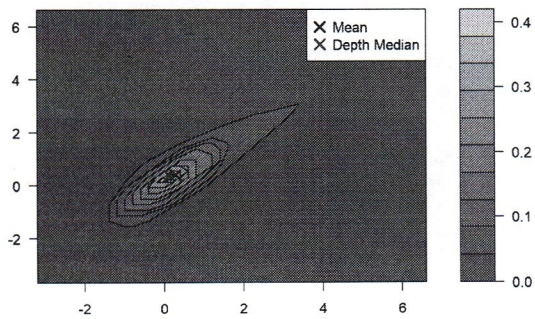
```
##    user  system elapsed
##    0.97    0.03    1.02
```

```
# cpp slighly worse than fortran implementation
```

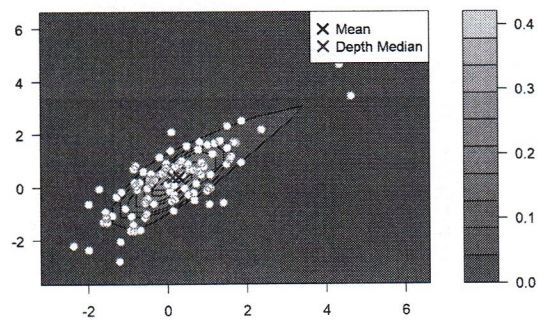
```
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### EXAMPLE 4 - Let's play with the contour and the persp plot
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
# Let's recover the data of example 2
plot(mixbivnorm[,1],mixbivnorm[,2], xlab="Norm 1", ylab="Norm 2")
```




```
# We have seen that the contour plot of the DepthProc package can be generated like:
depthContour(mixbivnorm, depth_params = list(method='Tukey'))
```

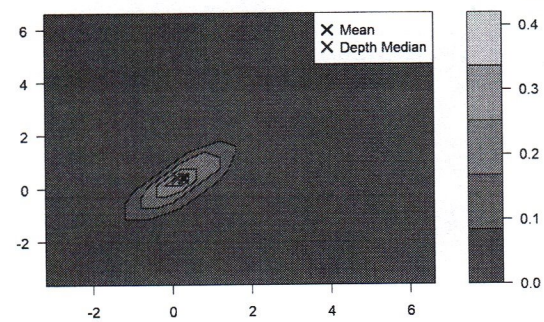



```
# You can decide to tweak a LOT of possible options
# add points
depthContour(mixbivnorm, depth_params = list(method='Tukey'),points=T)
```

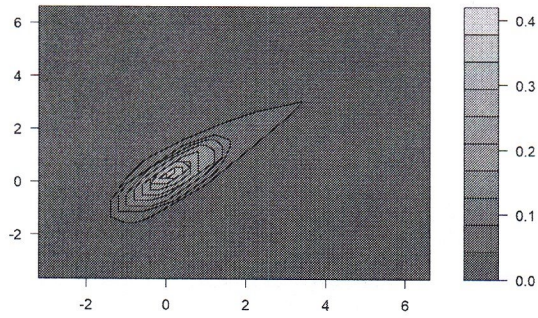



```
# change color scale levels
depthContour(mixbivnorm, depth_params = list(method='Tukey'),levels=5)
```

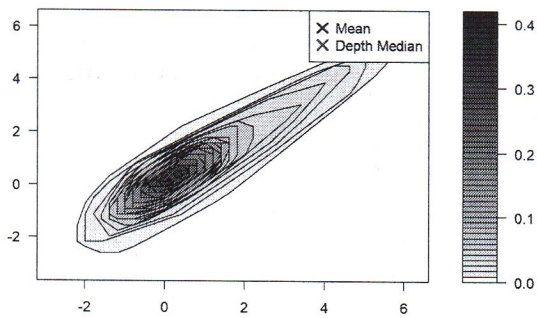
we're deciding how many levels we want to see



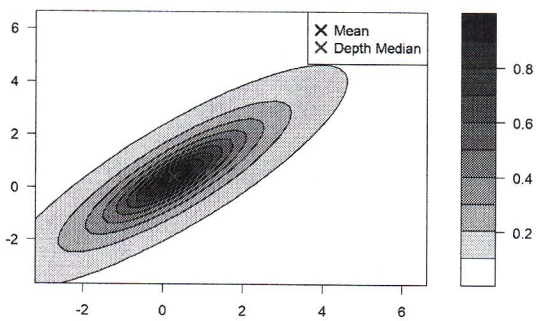
```
# decide what to show
depthContour(mixbivnorm, depth_params = list(method='Tukey'),pmean=F,pdmedian=F)
```



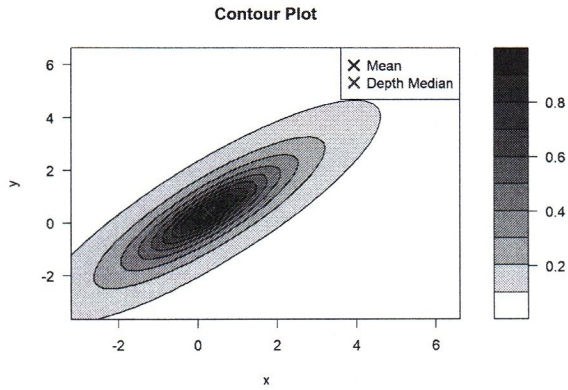

```
#customize the colors
gradfun = colorRampPalette(c('white','navy'))
depthContour(mixbivnorm, depth_params = list(method='Tukey'),colors=gradfun,levels=50)
```



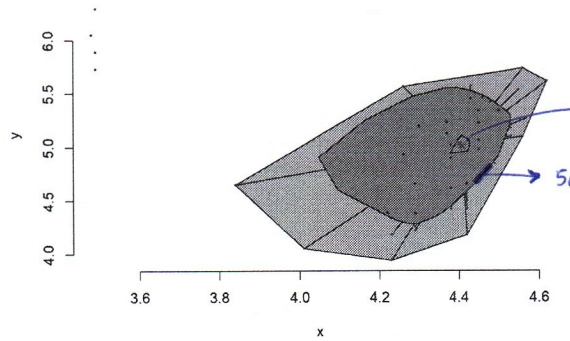

```
# or, probably more importantly, you can choose what depth to use.
# We are actually already doing it, but let's be aware:
depthContour(mixbivnorm, depth_params = list(method='Mahalanobis'),colors=gradfun,levels=10)
```




```
# perfect ellipsoids in this case

# If you want to add additional parameters to the plot, the way to do it is:
depthContour(mixbivnorm,
             depth_params = list(method='Mahalanobis'),
             graph_params = list(main='Contour Plot',xlab='x',ylab='y'),
             colors       = gradfun,levels=10)
```




```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### EXAMPLE 5 - Let's play with the bagplot
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# Let's use the starsCYG dataset
aplpack::bagplot(starsCYG)
```



set of points with maximal depth

50% quantile

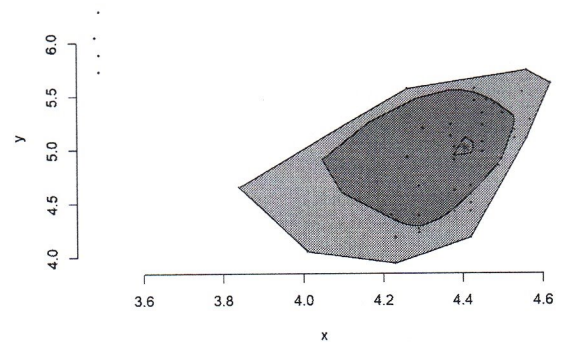
HOW TO **FIND THE MEDIAN**?

(we don't have a single median, we have a set of medians)
To obtain a single value for the median (a single point, 2 coordinates) we can for instance average all the points inside this smaller bag.

we compute the depths of the points that we have, we find the ones with the maximal depth and:

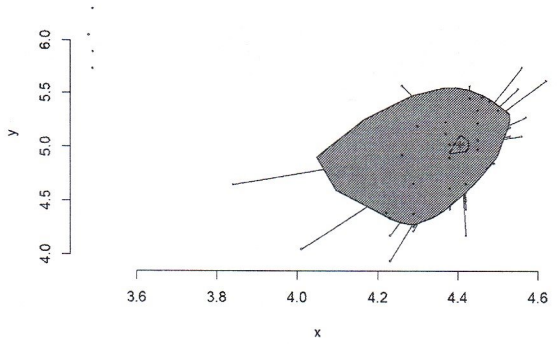
- if ∃! point with maximal depth ⇒ that's the median
- if there are multiple points with the maximal depth then we have to average them

```
# Let's now try to recreate a simple fence plot, and a simple sunburst plot
# bagplot
aplpack::bagplot(starsCYG,show.whiskers = F)
```

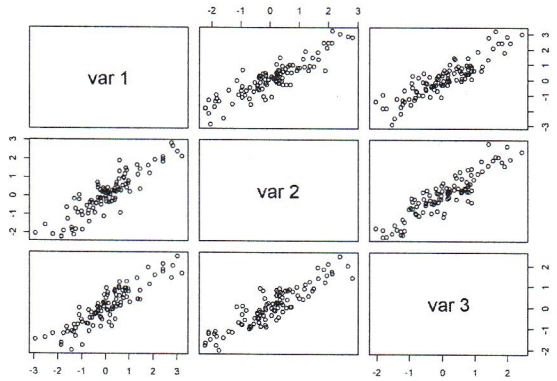


Possible exam question:
What is the shape of the bagplot of a given distribution?
Rivedi i comandi per generare due determinate distribuzioni!
(prof: "vi diciamo ad esempio di generare un dataset di cui una colonna è distr. — e l'altra — ")

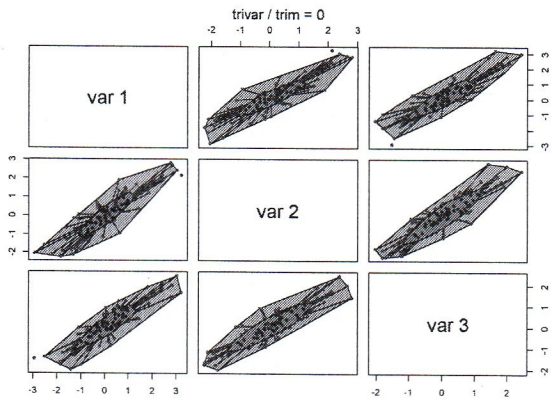
```
# sunburst
aplpack::bagplot(starsCYG,show.loophull = F)
```




```
# What happens, instead, if I have a multivariate dataset, and not just a bivariate one?
# bagplot/sunburst plot matrices!
# Let's generate a trivariate normal dataset
p      = 3
mu     = rep(0,3)
A      = matrix(runif(p^2),ncol=p)
sigma  = t(A) %*% A
trivar = mvrnorm(n=100,mu,sigma)
pairs(trivar)
```



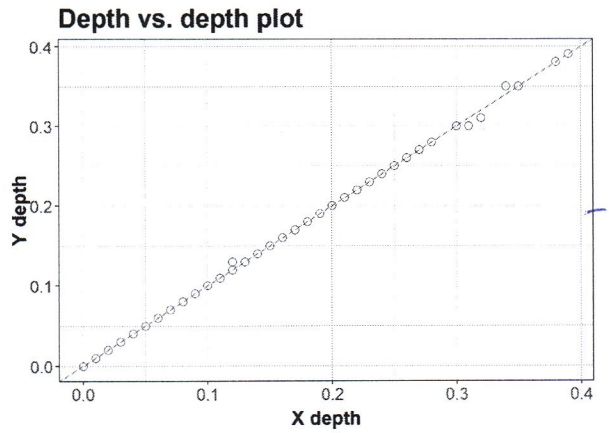

```
aplpack::bagplot.pairs(trivar)
```




```
##               [,1]        [,2]        [,3]
##   [1,]  2.417899028  1.83827875  1.82206362
##   [2,]  2.081780939  1.95254376  1.54973558
##   [3,]  ..
##  [99,]  0.140974690 -0.36275515 -0.72836817
## [100,]  1.457668447  0.35475689  0.35943170
```

multivariate equivalent of the QQ-plot

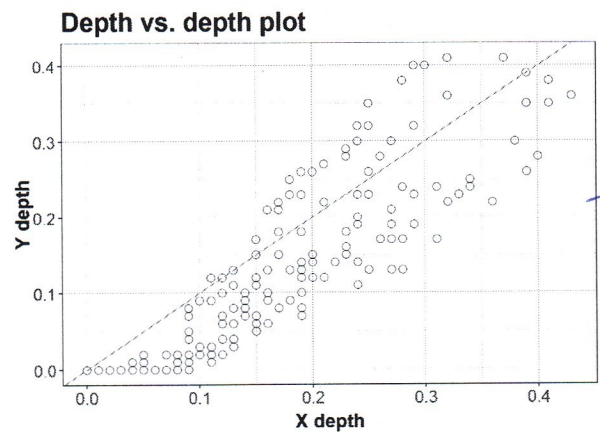
```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### EXAMPLE 6 - the DD-Plot
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
# Let's check, in a graphical way, if a given distributional assumption holds or not.
# Let's generate from a bivariate t-student distribution, and compare the depth quantiles
# with a bivariate normal
bivt    = cbind(rt(100,2),rt(100,2))
bivnorm = cbind(rnorm(100),rnorm(100))
# will it change something if I use a gaussian with a different mean?

ddPlot(bivnorm,bivnorm,depth_params = list(method='Tukey'))
```



The distributional assumptions stand

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

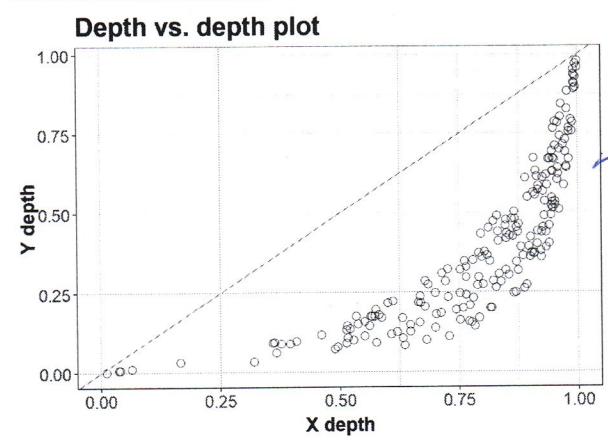
```
ddPlot(bivt,bivnorm,depth_params = list(method='Tukey'))
```



The situation is not under control, we can conclude that the distr. assumptions don't stand.

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

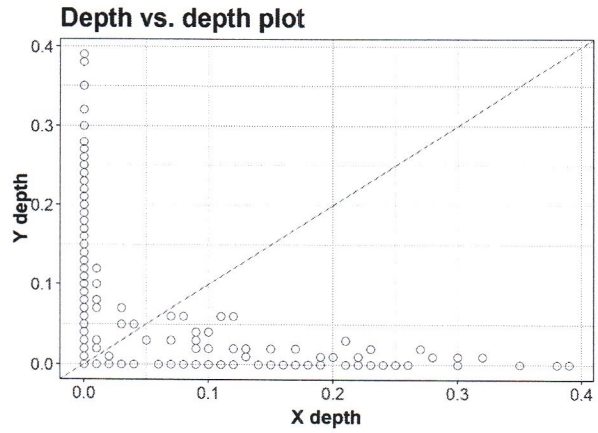
```
ddPlot(bivt,bivnorm,depth_params = list(method='Mahalanobis'))
```



More clear than before that the distr. assumptions don't hold

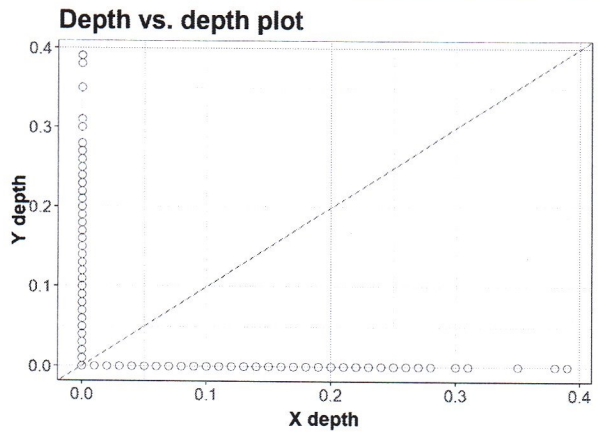
```
##
## Depth Metohod:
##   Mahalanobis
```

```
# Let's try to shift the distributions...
ddPlot(bivnorm,bivnorm+2,depth_params = list(method='Tukey'))
```



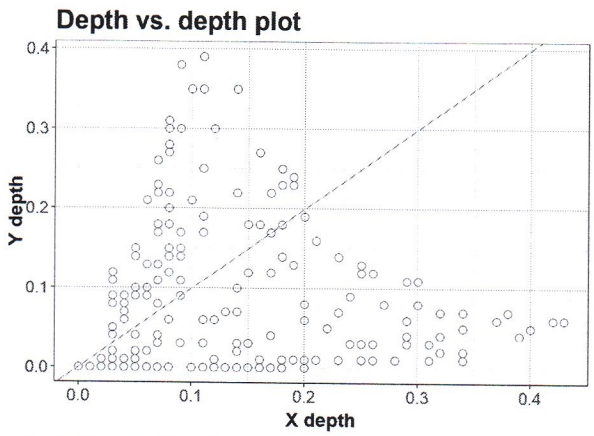

```
##
## Depth Metohod:
##   Tukey
```

```
ddPlot(bivnorm,bivnorm+3,depth_params = list(method='Tukey'))
```




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

```
ddPlot(bivt,bivnorm+1,depth_params = list(method='Tukey'))
```



!! This allows us to "test" for a center of a given distribution !!

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