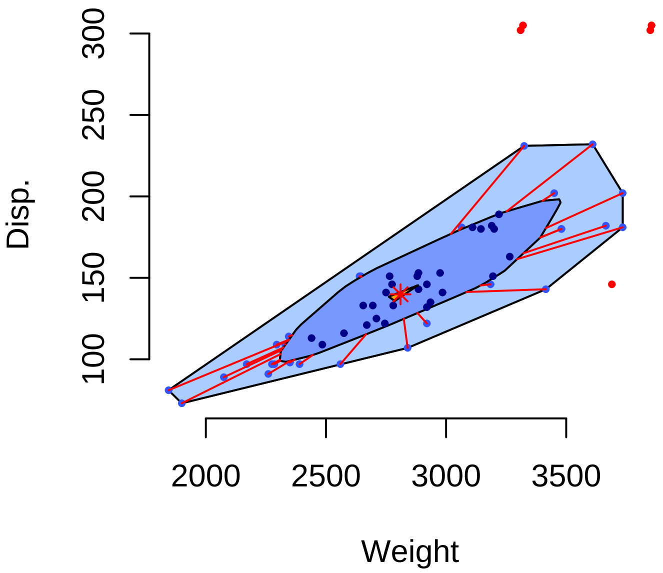
A rough R Impementation of the Bagplot

File: bagplot.rev
in: /home/wiwi/pwolf/R/work/bagplot

Version: 31th August, 2007



Contents

In this paper we describe a rough implementation of the bagplot. The first section shows some examples. Section 2 compares our bagplot function to the solution of Rousseeuw, Ruts, and Tukey (1999). Then the arguments, the help page of the function `bagplot` and some links are listed. In section 4 you find the definition of the function. In the appendix further examples for testing are given and some old code chunks are listed.

1 ⟨*version of bagplot* 1⟩ ≡

```
## 2007/08/31 peter wolf ##
```

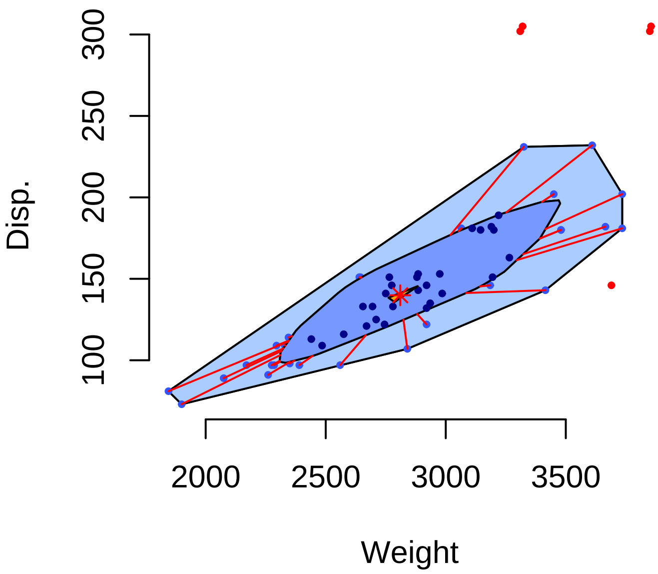
1 Examples

1.1 Example: car data (Chambers / Hastie 1992)

The first example is a bagplot of the famous car data of Chambers and Hastie. In the code chunk the data set is assigned to `cardata` and `bagplot()` is called with some parameters that are described later in this paper.

2 ⟨*cardata* 2⟩ ≡

```
⟨define bagplot 29⟩
library(rpart); cardata<-car.test.frame[,6:7]; par(mfrow=c(1,1))
bagplot(cardata,verbose=FALSE,factor=3,show.baghull=TRUE,dkmethod=2,
show.loophull=TRUE,precision=1)$center
#title("car data Chambers/Hastie 1992")
```



The Tukey median of our bagplot function is (2810.431, 139.879). Splus computes a slightly different point: (2806.63, 139.513). In difference to Rousseeuw et al. our bagplot as well as the bagplot of Splus classified the data point of Nissan Van 4 as outlier. To get the Splus results you have to download `bagplot*`, the car data and ...

```
Splus CHAPTER bagplot.f
Splus make
Splus ...
> dyn.open("S.so"); source("bagplot.s") #; postscript("hello.ps")
> bagplot(cardata[,1],cardata[,2]) #; dev.off()
```

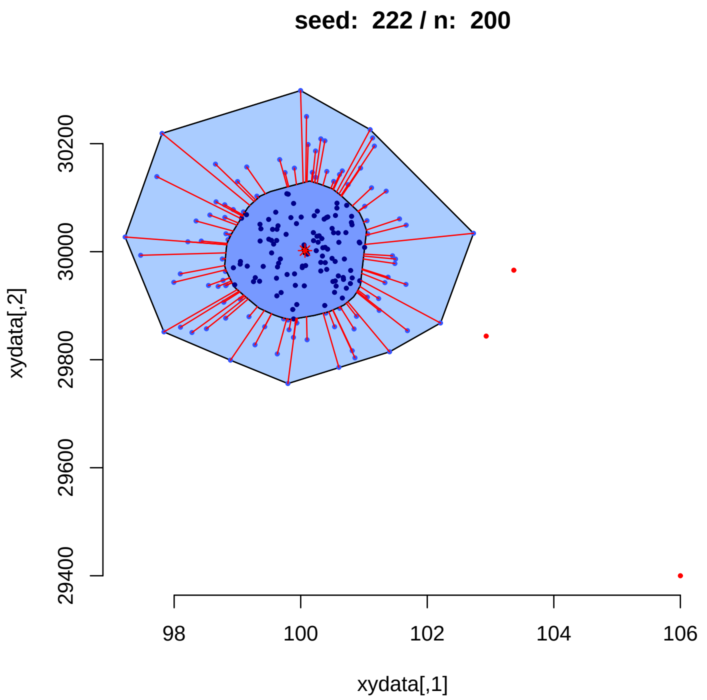
For R have a l ook at: http://www.statistik.tuwien.ac.at/public/filz/students/edavis/ws0607/skriptum/page134.html

1.2 The normal case

A bagplot of an rnorm sample with one heavy outlier is shown by the following code chunk.

3 ⟨*rnorm* 3⟩ ≡

```
⟨define bagplot 29⟩
seed<-222; n<-200
⟨define rnorm data data, seed: seed, size: n 76⟩
datan<-rbind(data,c(106,294)); datan[,2]<-datan[,2]*100
bagplot(datan,factor=3,create.plot=TRUE,approx.limit=300,
    show.outlier=TRUE,show.looppoints=TRUE,show.bagpoints=TRUE,
    show.whiskers=TRUE,show.loophull=TRUE,show.baghull=TRUE,
    verbose=FALSE)
title(paste("seed: ",seed,"/ n: ",n))
```

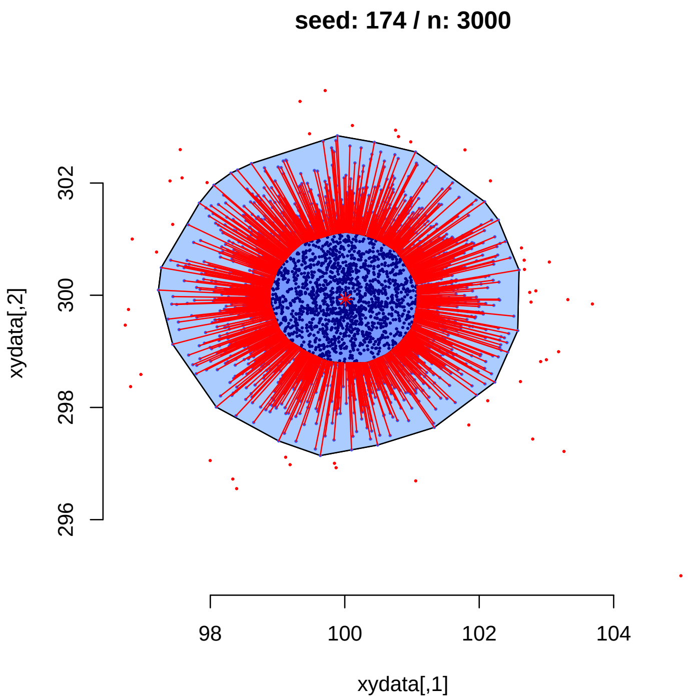


1.3 Large data sets

What about very large data sets? The algorithm computes some of the quantities of the bagplot on base of a sample if there are more then `approx.limit` data points.

4 ⟨*large* 4⟩ ≡

```
seed<-174; n<-3000
⟨define rnorm data data, seed: seed, size: n 76⟩
datan<-rbind(data,c(105,295))
bagplot(datan,factor=2.5,create.plot=TRUE,approx.limit=1000,
   cex=0.2,show.outlier=TRUE,show.looppoints=TRUE,
   show.bagpoints=TRUE,dkmethod=2,show.loophull=TRUE,
   show.baghull=TRUE,verbose=FALSE,debug.plots="no")
title(paste("seed:",seed,"/ n:",n))
```

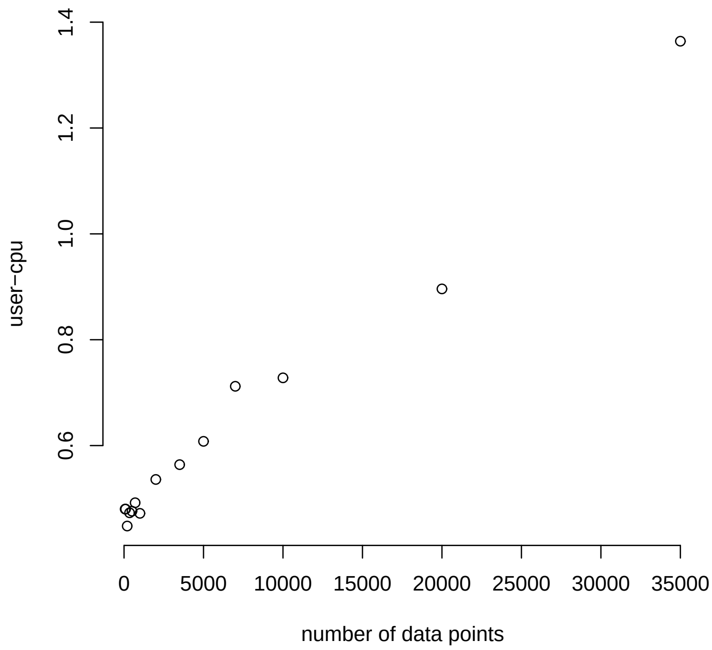


1.4 Size of data set

The time for computation increases with the number of observations. To illustrate the run times we measure the times necessary for rnorm data sets of different sizes and plot the results.

5 ⟨*rnorm, different sizes* 5⟩ ≡

```
⟨define bagplot 29⟩
nn<-c(50,70,100,200,350); nn<-c(nn,10*nn,100*nn);nn<-nn[-1]
result<-1:length(nn)
for(j in seq(along=nn)){
  seed<-111; set.seed(seed); n<-nn[j]
  xy<-cbind(rnorm(n),rnorm(n))
  result[j]<-system.time(
    bagplot(xy,factor=3,create.plot=FALSE,approx.limit=300,
     show.outlier=TRUE,show.looppoints=TRUE,show.bagpoints=TRUE,
     show.whiskers=TRUE,show.loophull=TRUE,show.baghull=TRUE,
     verbose=FALSE)
    )[1]
}
plot(nn,result,bty="n",ylab="user-cpu",xlab="number of data points")
names(result)<-nn; result
```



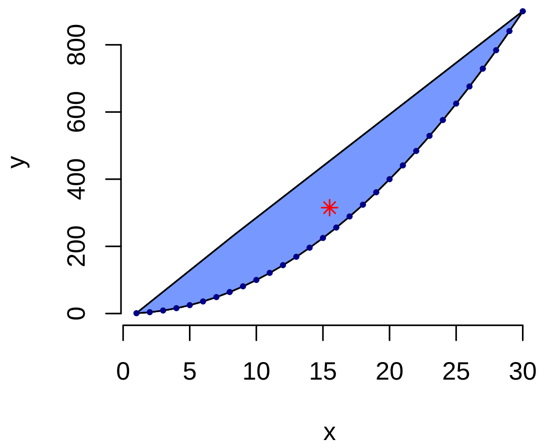

```
Wed Aug 29 11:29:35 2007
   70   100   200   350   500   700  1000  2000  3500  5000  7000 10000 20000 35000
0.480 0.480 0.448 0.473 0.476 0.492 0.472 0.536 0.564 0.608 0.712 0.728 0.896 1.364
```

1.5 "Depth-One" data sets

It is very interesting to test extrem cases. What happens if the depths of all points are one?

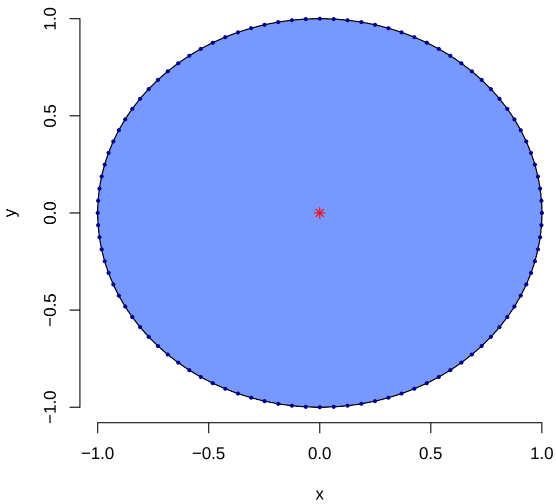
6 ⟨*quadratic* 6⟩ ≡

```
⟨define bagplot 29⟩
bagplot(x=1:30,y=(1:30)^2,verbose=FALSE,dkmethod=2)
```



7 ⟨*circle* 7⟩ ≡

```
⟨define bagplot 29⟩
n<-100;bagplot(x=cos((1:n)/n*2*pi),y=sin((1:n)/n*2*pi),
   precision=1,verbose=FALSE,dkmethod=2,debug.plot=FALSE)$center
```

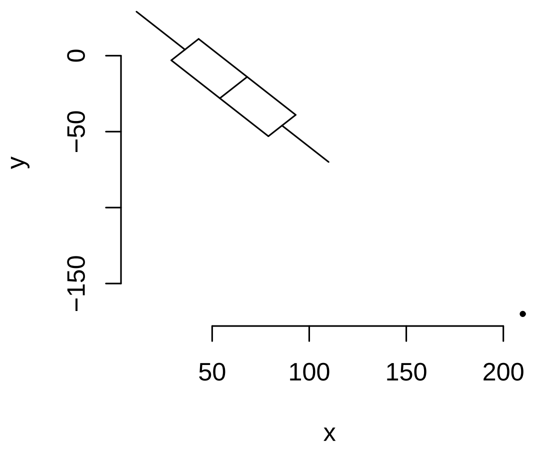


1.6 Degenerated data sets

What happens if all the data points lie in a one dimensional subspace?

8 ⟨*onedim* 8⟩ ≡

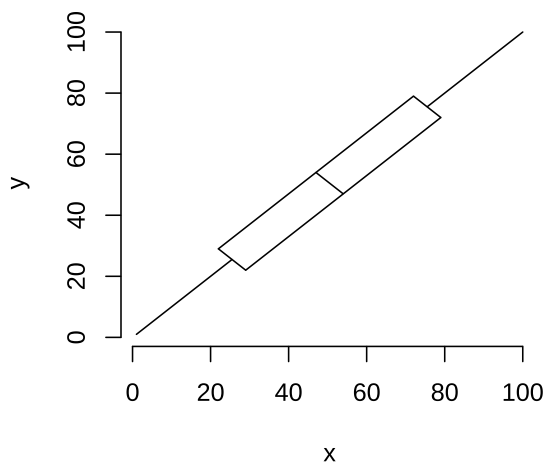
```
bagplot(x=10+c(1:100,200),y=30-c(1:100,200),verbose=FALSE)
```



Here is a second one dim data set.

9 ⟨*one dim test* 9⟩ ≡

```
bagplot(x=(1:100),y=(1:100),verbose=FALSE)
```



1.7 Data set from the mail of M. Maechler

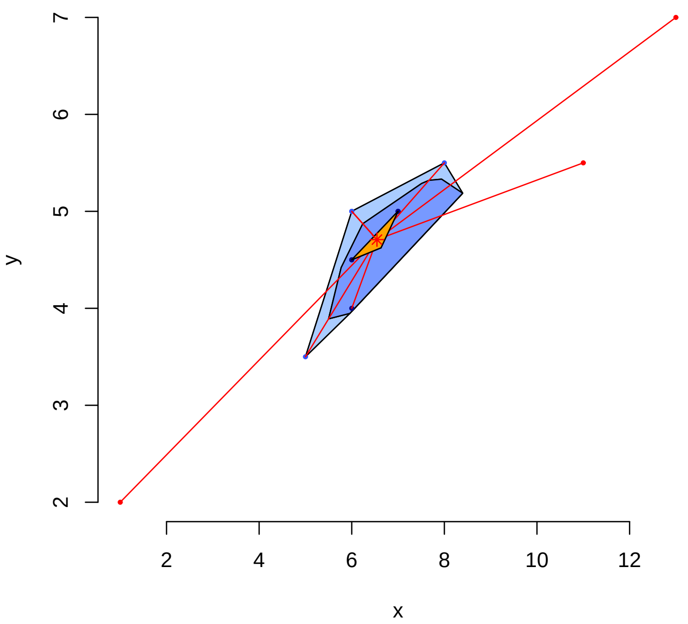
The data set of M. Maechler is discussed within R-help. Decide of yourself if our bagplot is acceptable. Maybe this doesn't matter because mostly a data set is *in regular position* (Rousseeuw, Ruts 1998) and there are no identical coordinates. But it may happen, e.g. in the car data set there are two points that are identical.
M. Maechler wrote in a reply concerning a bagplot question that the correct Tukey median is (6.75 , 4.875) and not (6.544480, 4.708483) that is computed by our bagplot procedure.

10 ⟨*data set of Martin Maechler* 10⟩ ≡

```
⟨define bagplot 29⟩
⟨assing data set of Martin Maechler to x0 and y0 11⟩
bagplot(x0,y0,show.baghull=TRUE,show.loophull=TRUE,
        create.plot=TRUE,show.whiskers=TRUE,factor=3,
        debug.plots="notall",dkmethod=2,verbose=FALSE,
        precision=1)$center
#abline(h=4.85,v=6.75)
```

11 ⟨*assing data set of Martin Maechler to* `x0` *and* `y0` 11⟩ ≡

```
x0<-c(1,5,  6,6,  6,  6,6,7,7,8,  11, 13) #; x0 <- c(x0, 8)
y0<-c(2,3.5,4,4.5,4.5,5,5,5,5,5.5,5.5, 7) #; y0 <- c(y0, 7)
```

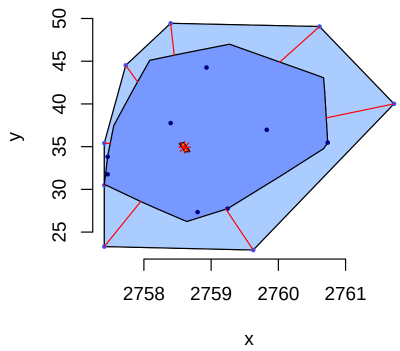


1.8 Data sets of Wouter Meuleman, running in an error with version 09/2005

An old bagplot version runs into errors with following data set. During the computation of ⟨*find* `hull.bag` 65⟩ some NaN values occured.

12 ⟨*data set 2 of Wouter Meuleman 12*⟩ ≡

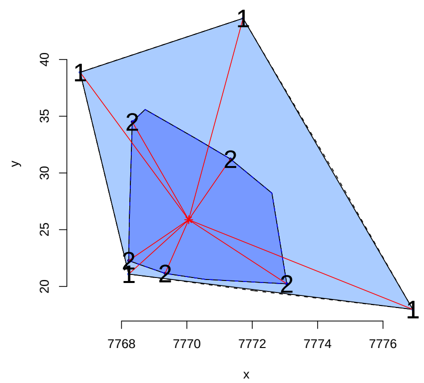
```
a<-gsub("\n"," ",c("3  2759.626 22.90411 6  2757.461 31.75789 13 2758.931 44.25797
15 2757.411 30.47785 16 2761.720 40.01067 18 2759.827 36.97118 19 2758.398 49.43611
21 2757.411 23.30404 26 2757.461 33.81379 27 2758.398 37.75841 28 2759.244 27.74002
32 2757.411 35.40853 34 2760.734 35.47206 38 2760.612 49.05950 39 2757.730 44.51406
40 2758.798 27.33595"))
a<-unlist(strsplit(paste(a,collapse=""),"  "))
a<-as.numeric(a[a!=""]); a<-matrix(a,ncol=3,byrow=TRUE)
⟨define bagplot 29⟩
bagplot(a[,2],a[,3],verbose=FALSE)
```



On 2006/02/17 some lines of code have been changed to remove the NaN values.

13 ⟨*data set 1 of Wouter Meuleman 13*⟩ ≡

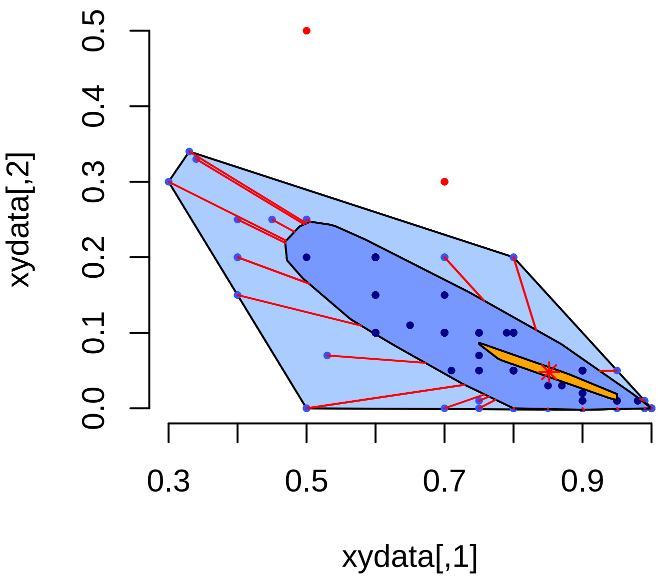
```
a<-gsub("\n"," ",c("1  7766.734 38.86814 2  7768.329 34.50661 3  7769.335
21.14797 4  7768.221 21.08619 5  7776.913 17.97344 6  7768.221 22.27727 8
7771.719 43.62978 9  7773.056 20.22909 12 7771.334 31.22399"))
a<-unlist(strsplit(paste(a,collapse=""),"  "))
a<-as.numeric(a[a!=""])
a<-matrix(a,ncol=3,byrow=TRUE)
⟨define bagplot 29⟩
bagplot(a[,2],a[,3],verbose=TRUE,dkmethod=2)
```



The following data set was proposed by Ben Greiner in January 2007.

14 ⟨*test: data set of Ben Greiner* 14⟩ ≡

```
⟨define bagplot 29⟩
greiner.data<-cbind(c( 1,1,1,0.7,0.8,0.98,0.9,0.85,1,1,0.7,1,0.65,
0.8,0.5,0.7,0.95,0.7,0.8,0.8,0.75,1,0.95,0.7,0.95,0.8,0.75,0.7,0.85,
0.8,0.8,1,0.5,0.9,0.7,0.8,0.6,0.9,0.98,1,0.5,0.45,0.95,1,0.9,0.9,
0.7,1,1,0.7,1,0.4,0.9,0.85,0.75,1,0.5,0.9,0.4,0.95,0.8,0.95,0.99,
1,0.34,0.6,1,0.9,0.6,0.7,0.8,0.7,0.95,1,0.6,0.99,0.85,0.78,0.8,1,
0.4,1,0.33,0.99,0.6,0.8,0.85,0.75,0.9,0.9,1,0.9,1,0.8,1,0.9,1,0.71,
0.4,0.8,1,0.7,1,0.8,1,0.6,0.6,1,0.6,1,1,0.7,0.85,1,0.8,1,0.95,0.8,
0.9,0.8,0.6,0.85,1,0.9,0.9,0.8,1,1,0.6,0.9,1,1,0.5,0.75,0.53,0.8,
0.7,0.3,0.8,0.9,0.7,0.8,0.6,0.9,0.8,0.8,0.6,1,0.6,1,1,0.9,0.8,0.7,
0.6,0.8,1,0.5,0.85,1,0.75,1,0.8,1,0.85,1,0.75,0.8,0.7,0.87,1,1,1,
0.7,0.79,0.8,0.6,0.9,0.6,0.8,0.6,0.7,0.8,0.99,0.9,0.75 ),
c( 0,0,0,0.1,0,0.01,0,0.05,0,0,0.1,0,0.11,0.1,0,0.1,0,0.3,0,0,0.07,
0,0.01,0.1,0,0.05,0.05,0.3,0.05,0.1,0,0,0.25,0,0.1,0.05,0.2,0.05,
0.01,0,0.25,0.25,0.05,0,0.05,0.02,0.1,0,0,0.1,0,0.25,0.03,0.05,0.1,
0,0.2,0.01,0.2,0,0.1,0.01,0,0,0.33,0.1,0,0.05,0.15,0.1,0.1,0.1,0.01,
0,0.1,0,0.05,0.07,0.1,0,0.15,0,0.34,0,0.15,0.1,0.03,0,0,0.05,0,0.05,
0,0.2,0,0.01,0,0.05,0.2,0.05,0,0.15,0,0.05,0,0.2,0.2,0,0.2,0,0,0.2,
0.05,0,0.05,0,0.01,0,0.05,0.05,0.1,0.05,0,0.05,0.05,0.05,0,0,0.15,
0.05,0,0,0,0.05,0.07,0.05,0.1,0.3,0.05,0,0.1,0.1,0.2,0.02,0.05,0.2,
0.2,0,0.1,0,0,0.05,0.05,0.1,0.2,0.1,0,0.5,0,0,0.1,0,0.05,0,0.05,0,
0.01,0.05,0.1,0.03,0,0,0,0,0.1,0.05,0.1,0.05,0.1,0,0.2,0.2,0,0.01,0,0.1))
bagplot(greiner.data)
```

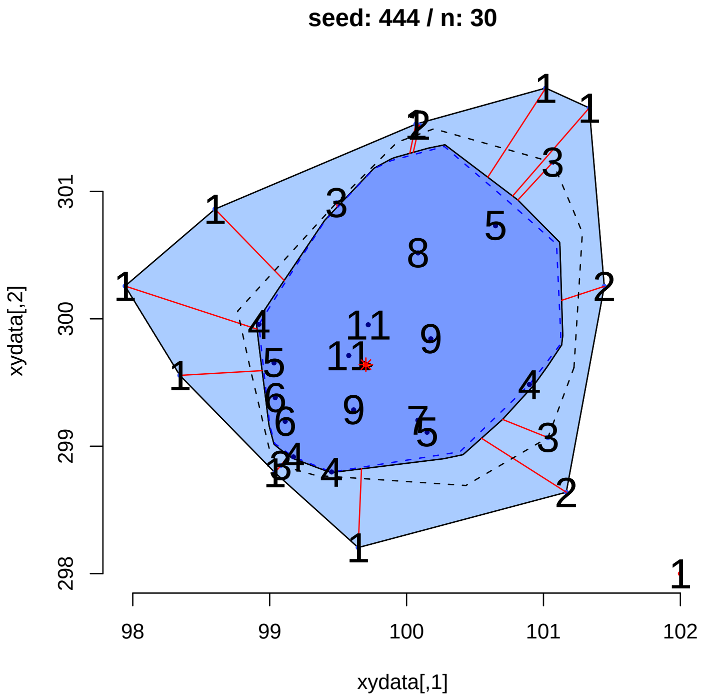


1.9 Bagplot with additional graphical supplements

Verbose computation of bagplot of a sample of 100 rnorm points and an outlier is performed by the following code chunk. With the verbose option the h-depths of the data points are shown in the plot and some of the intermediate results are printed during the the computation.

15 ⟨*verbosetest 15*⟩ ≡

```
seed<-444; n<-30
⟨define rnorm data data, seed: seed, size: n 76⟩
datan<-rbind(data,c(102,298))
bagplot(datan,factor=2.5,create.plot=TRUE,approx.limit=300,
   show.outlier=TRUE,show.looppoints=TRUE,show.bagpoints=TRUE,dkmethod=2,
   show.whiskers=TRUE,show.loophull=TRUE,show.baghull=TRUE,verbose=TRUE)
title(paste("seed:",seed,"/ n:",n))
```

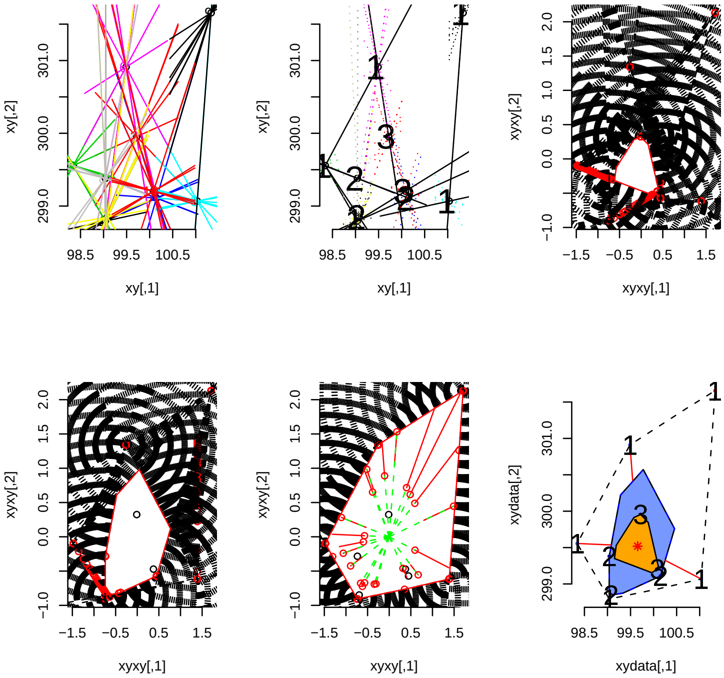


1.10 Debugging plots with additional elements

Here is an example of plots generated with option `debug.plots="all"`. This option has been helpful during debugging and now the plots can be classified as R art.

16 ⟨*debugplot* 16⟩ ≡

```
seed<-444; n<-30
⟨define rnorm data data, seed: seed, size: n 76⟩
datan<-data[1:10,] #datan<-cbind(c(1:100,200),c(1:100,200))
par(mfrow=c(2,3))
bagplot(datan,factor=2.5,create.plot=TRUE,approx.limit=300,
   show.outlier=TRUE,show.looppoints=TRUE,show.bagpoints=TRUE,
   show.whiskers=TRUE,show.loophull=FALSE,show.baghull=TRUE,dkmethod=2,
   debug.plots="all",verbose=TRUE); par(mfrow=c(1,1))
```



2 Bagplots by an alternative approach, proposed by Rousseeuw, Ruts and Tukey

As mentioned above there is a solution using a fortran procedure for generating bagplots, see:

http://www.statistik.tuwien.ac.at/public/filz/students/edavis/ws0607/skriptum/page134.html.

To get the procedure work you have to perform the following steps:

- fetch the fortran code by downloading

  `$ get ftp://ftp.win.ua.ac.be/pub/software/agoras/newfiles/bagplot.tar.gz`

  – this link has been found on the web page: http://www.agoras.ua.ac.be/Locdept.htm

- unzip and unpack the tar.gz-file

  `$ gunzip bagplot.tar.gz; tar -xvf bagplot.tar`

- translate the fortran program `bagplot.f` and generate the object file `bagplot.so`

  `$ R CMD SHLIB -o bagplot.so bagplot.f`

- download bagplot-R-function

  `$ get http://www.statistik.tuwien.ac.at/public/filz/students/edavis/ws0607/skriptum/bagplot.R`

- start R and load so-file

17 ⟨* 17⟩ ≡

```
dyn.load("Tukey/bagplot.so")
```

- source bagplot function; to avoid conflicts in the names we change the name of the bagplot function of Rousseeuw, Ruts, and Tukey to `BAGPLOT`.

18 ⟨* 17⟩+ ≡

```
BAGPLOT<-readLines("Tukey/BAGPLOT.R")
eval(parse(text=sub("^.bagplot","\"BAGPLOT",BAGPLOT))); "ok"
args(BAGPLOT)
```

  Here are the arguments of `BAGPLOT()`:

```
Wed Aug 29 15:19:43 2007
function (x, y, plotinbag = T, plotoutbag = T, ident = T, drawfence = F,
    drawloop = T, truncxmin = NULL, truncxmax = NULL, truncymin = NULL,
    truncymax = NULL, xlab = "x", ylab = "y", ...)
```

- compute an example bagplot.

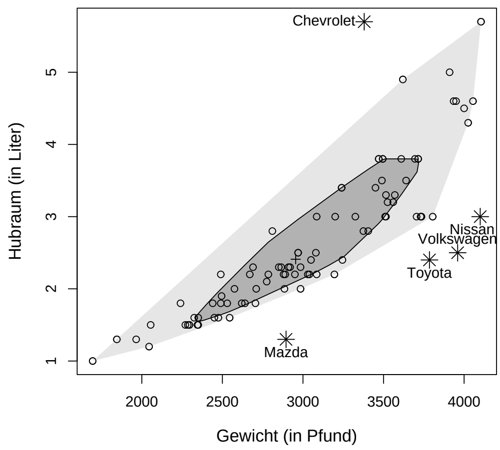
19 ⟨* 17⟩+ ≡

```
library(MASS)
data(Cars93); x<-Cars93[,"Weight"]; y<-Cars93[,"EngineSize"]
names(x)<-dimnames(Cars93)[[1]]; names(y)<-dimnames(Cars93)[[1]]
# par(mar=c(4,4,1,1))
BAGPLOT(x,y,xlab="Gewicht (in Pfund)",ylab="Hubraum (in Liter)",
##      main="Press Mouse BUTTON!",
        cex.lab=1.2)
text(3380,5.7,"Chevrolet",pos=2); text(2895,1.3,"Mazda",pos=1)
text(4050,3.0,"Nissan",pos=1);    text(3785,2.4,"Toyota",pos=1)
text(3960,2.5,"Volkswagen",pos=3)
```

  Here is the numerical result

```
[1] The coordinates of the Tukey median are ( 2954.84 , 2.40962 ).
```

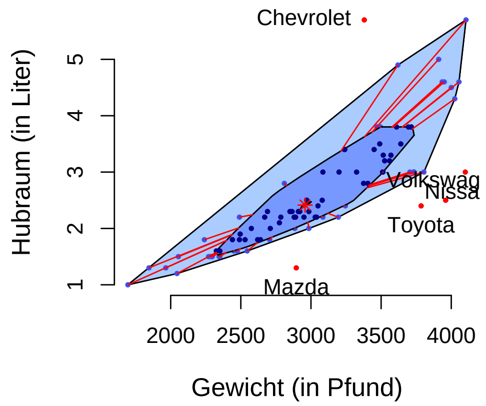
and the bagplot:



A reconstruction of this plot can be done by our bagplot function. For a suitable loop you have to set `factor=2.8`.

20 ⟨* 17⟩+ ≡

```
library(MASS)
data(Cars93); x<-Cars93[,"Weight"]; y<-Cars93[,"EngineSize"]
names(x)<-dimnames(Cars93)[[1]]; names(y)<-dimnames(Cars93)[[1]]
center<-bagplot(x,y,factor=2.8,xlab="Gewicht (in Pfund)",
                ylab="Hubraum (in Liter)",cex.lab=1.2)$center
text(3380,5.7,"Chevrolet",pos=2); text(2895,1.3,"Mazda",pos=1)
text(4050,3.0,"Nissan",pros=1);    text(3785,2.4,"Toyota",pos=1)
text(3960,2.5,"Volkswagen",pos=3)
center
```



We find the center:

```
Tue Aug 28 19:06:10 2007
[1] 2958.051384    2.413979
```

The difference may be caused by numerical difficulties.

Test of data set of Martin Maechler. As a second example we check the bagplot functions by the data set of Martin Maecher.

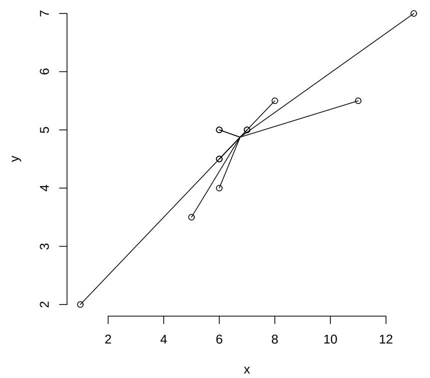
21 ⟨*BAGPLOT of data set of Martin Maechler* 21⟩ ≡
⟨*assing data set of Martin Maechler to* `x0` *and* `y0` 11⟩

```
BAGPLOT(x0,y0)
```

We get the numerical result ...

```
[1] The coordinates of the Tukey median are ( 6.75 , 4.875 ).
```

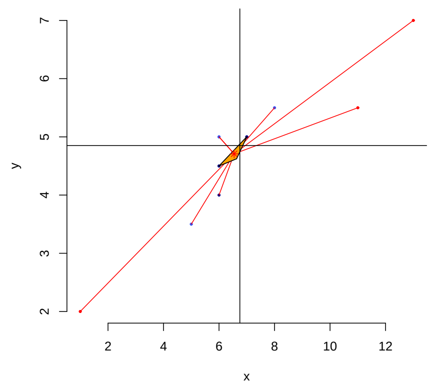
and the following plot



Our procedure will compute slightly different results:

22 ⟨*bagplot of data set of Martin Maechler* 22⟩ ≡
⟨*assing data set of Martin Maechler to* `x0` *and* `y0` 11⟩

```
center<-bagplot(x0,y0,show.baghull=FALSE,show.loophull=FALSE,
                create.plot=TRUE,show.whiskers=TRUE,factor=3,
                dkmethod=2,precision=1)$center
abline(h=4.85,v=6.75); center
```



The two lines mark the Tukey median computed by BAGPLOT. Our median is:

```
[1] 6.544480 4.708483
```

Now we check the stability of the functions by exchanging the variables.

23 ⟨*BAGPLOT of data set of Martin Maechler, exchanged variables* 23⟩ ≡

```
 ⟨assing data set of Martin Maechler to x0 and y0 11⟩
 BAGPLOT(y0,x0)
```

```
[1] The coordinates of the Tukey median are ( 4.84231 , 6.68461 ).
```

There is a difference! The relative difference is:

24 ⟨*BAGPLOT of data set of Martin Maechler, relative difference* 24⟩ ≡

```
 abs((c(6.75 ,4.875)-c(4.84231,6.68461))/c(6.75,4.875))
```

```
[1] 0.2826207 0.3712021
```

How will our function master the test?

25 ⟨*bagplot of data set of Martin Maechler, exchanged variables* 25⟩ ≡

```
 center.ex<-bagplot(y0,x0,show.baghull=FALSE,show.loophull=FALSE,
                  create.plot=TRUE,show.whiskers=TRUE,factor=3,
                  dkmethod=2,precision=1)$center
```

```
[1] 4.708846 6.540282
```

The relative difference is approximately 0.07%. If we increase the precision by `precision=5` the difference is reduced as we like it:

26 ⟨*bagplot of data set of Martin Maechler, difference if precision is increased* 26⟩ ≡

```
 center    <-bagplot(x0,y0,create.plot=FALSE,factor=3,precision=6)$center
 center.ex<-bagplot(y0,x0,create.plot=FALSE,factor=3,precision=6)$center
 print(center)
 abs(center-center.ex[2:1])/center
```

The results seems to be identical for we get:

```
[1] 6.541650 4.708349
[1] 0 0
```

By analyzing the scatterplot we find that the area of the points with h-depth 4 is a triangle. The corners of this triangle are: (6, 4.5), (7,5) and (6.625, 14.125). The center of its gravity is equal to the mean of the three points and we get the Tukey median (6.541666, 4.7083333). Our bagplot function finds a result that is very near to the one computed by hand.

Memory faults. There are some other problems with the implementation via the fortran procedure because we got some memory faults during testing `BAGPLOT`. These errors killed the R process and some of the computed result got lost. But it was not difficult to reconstruct them ... by relax.

3 Arguments and output of `bagplot`, the help page and some links

A summary of the arguments can be found by `args()`.

27 ⟨*args* 27⟩ ≡

```
args(bagplot)
```

```
function (x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
    show.outlier = TRUE, show.whiskers = TRUE, show.looppoints = TRUE,
    show.bagpoints = TRUE, show.loophull = TRUE, show.baghull = TRUE,
    create.plot = TRUE, add = FALSE, pch = 16, cex = 0.4, dkmethod = 2,
    precision = 1, verbose = FALSE, debug.plots = "no",
    col.loophull = "#aaccff", col.looppoints = "#3355ff",
    col.baghull = "#7799ff", col.bagpoints = "#000088",
    transparency = FALSE, ...)
```

The output of `bagplot` is a list of the relevant quantities of the constructed bagplot. To identify singular points, use `identify()`. Here is a short description of the return values:

`center`	Tukey median
`hull.loop`	set of points of polygon that defines the loop
`hull.bag`	set of points of polygon that defines the bag
`hull.center`	region of points with maximal ldepth
`pxy.outlier`	outlier
`pxy.outer`	outer points
`pxy.bag`	points in bag
`hdepth`	location depth of data points in xy
`is.one.dim`	is `TRUE` if data set is one dimensional
`prdata`	result of PCA
`random.seed`	random.seed that is set by bagplot
`xydata`	data set
`xy`	sample of data set

The help page is defined as a code chunk.

28 ⟨*define help of* `bagplot` 28⟩ ≡

```
\name{bagplot}
\alias{bagplot}
\alias{compute.bagplot}
\alias{plot.bagplot}
\title{ bagplot, a bivariate boxplot }
\description{
  \code{compute.bagplot()} computes an object
  describing a bagplot of a bivariate data set.
  \code{plot.bagplot()} plots a bagplot object.
  \code{bagplot()} computes and plots a bagplot.
}
\usage{
bagplot(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
       show.outlier = TRUE, show.whiskers = TRUE,
       show.looppoints = TRUE, show.bagpoints = TRUE,
       show.loophull = TRUE, show.baghull = TRUE,
       create.plot = TRUE, add = FALSE, pch = 16, cex = 0.4,
       dkmethod = 2, precision = 1, verbose = FALSE,
       debug.plots = "no",   col.loophull="#aaccff",
```

```
       col.looppoints="#3355ff", col.baghull="#7799ff",
       col.bagpoints="#000088", transparency=FALSE, ...
)

compute.bagplot(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
       dkmethod=2,precision=1,verbose=FALSE,debug.plots="no")

plot.bagplot(x,
       show.outlier = TRUE, show.whiskers = TRUE,
       show.looppoints = TRUE, show.bagpoints = TRUE,
       show.loophull = TRUE, show.baghull = TRUE,
       add = FALSE, pch = 16, cex = 0.4, verbose = FALSE,
       col.loophull="#aaccff", col.looppoints="#3355ff",
       col.baghull="#7799ff", col.bagpoints="#000088",
       transparency=FALSE,...)
}

\arguments{
  \item{x}{ x values of a data set;
     in \code{bagplot}: an object of class \code{bagplot}
     computed by \code{compute.bagplot} }
  \item{y}{ y values of the data set }
  \item{factor}{ factor defining the loop }
  \item{na.rm}{ if TRUE 'NA' values are removed otherwise exchanged by mean}
  \item{approx.limit}{ if the number of data points exceeds
          \code{approx.limit} a sample is used to compute
          some of the quantities; default: 300 }
  \item{show.outlier}{ if TRUE outlier are shown }
  \item{show.whiskers}{ if TRUE whiskers are shown }
  \item{show.looppoints}{ if TRUE loop points are plottet }
  \item{show.bagpoints}{ if TRUE bag points are plottet }
  \item{show.loophull}{ if TRUE the loop is plotted }
  \item{show.baghull}{ if TRUE the bag is plotted }
  \item{create.plot}{ if FALSE no plot is created }
  \item{add}{ if TRUE the bagplot is added to an existing plot }
  \item{pch}{ sets the plotting character }
  \item{cex}{ sets characters size}
  \item{dkmethod}{ 1 or 2, there are two method of
     approximating the bag, method 1 is very rough }
  \item{precision}{ precision of approximation, default: 1 }
  \item{verbose}{ automatic commenting of calculations
 }
  \item{debug.plots}{ if TRUE additional plots describing
                     intermediate results are constructed }
  \item{col.loophull}{ color of loop hull }
  \item{col.looppoints}{ color of the points of the loop }
  \item{col.baghull}{ color of bag hull }
  \item{col.bagpoints}{ color of the points of the bag }
  \item{transparency}{ see section details }
  \item{\dots}{ additional graphical parameters }
}
\details{
A bagplot is a bivariate generalization of the well known
boxplot. It has been proposed by Rousseeuw, Ruts, and Tukey.
In the bivariate case the box of the boxplot changes to a
convex polygon, the bag of bagplot. In the bag are 50 percent
of all points. The fence separates points within the fence from
points outside. It is computed by increasing the
```

```
the bag. The loop is defined as the convex hull containing
all points inside the fence.
If all points are on a straight line you get a classical
boxplot.
\code{bagplot()} plots bagplots that are very similar
to the one described in Rousseeuw et al.
Remarks:
The two dimensional median is approximated.
For large data sets the error will be very small.
On the other hand it is not very wise to make a (graphical)
summary of e.g. 10 bivariate data points.

In case you want to plot multiple (overlapping) bagplots,
you may want plots that are semi-transparent. For this
you can use the \code{transparency} flag.
If \code{transparency==TRUE} the alpha layer is set to ’99’ (hex).
This causes the bagplots to appear semi-transparent,
but ONLY if the output device is PDF and opened using:
\code{pdf(file="filename.pdf", version="1.4")}.
For this reason, the default is \code{transparency==FALSE}.
This feature as well as the arguments
to specify different colors has been proposed by Wouter Meuleman.
}
\value{
  \code{compute.bagplot} returns an object of class
  \code{bagplot} that could be plotted by
  \code{plot.bagplot()}.
}
\references{ P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999):
    The bagplot: a bivariate boxplot, The American
    Statistician, vol. 53, no. 4, 382--387 }
\author{ Peter Wolf }
\note{
  Version of bagplot: 08/2007 }
\seealso{ \code{\link[graphics]{boxplot}} }
\examples{
  # example: 100 random points and one outlier
  dat<-cbind(rnorm(100)+100,rnorm(100)+300)
  dat<-rbind(dat,c(105,295))
  bagplot(dat,factor=2.5,create.plot=TRUE,approx.limit=300,
     show.outlier=TRUE,show.looppoints=TRUE,
     show.bagpoints=TRUE,dkmethod=2,
     show.whiskers=TRUE,show.loophull=TRUE,
     show.baghull=TRUE,verbose=FALSE)
  # example of Rousseeuw et al., see R-package rpart
  cardata <- structure(as.integer( c(2560,2345,1845,2260,2440,
   2285, 2275, 2350, 2295, 1900, 2390, 2075, 2330, 3320, 2885,
   3310, 2695, 2170, 2710, 2775, 2840, 2485, 2670, 2640, 2655,
   3065, 2750, 2920, 2780, 2745, 3110, 2920, 2645, 2575, 2935,
   2920, 2985, 3265, 2880, 2975, 3450, 3145, 3190, 3610, 2885,
   3480, 3200, 2765, 3220, 3480, 3325, 3855, 3850, 3195, 3735,
   3665, 3735, 3415, 3185, 3690, 97, 114, 81, 91, 113, 97, 97,
   98, 109, 73, 97, 89, 109, 305, 153, 302, 133, 97, 125, 146,
   107, 109, 121, 151, 133, 181, 141, 132, 133, 122, 181, 146,
   151, 116, 135, 122, 141, 163, 151, 153, 202, 180, 182, 232,
   143, 180, 180, 151, 189, 180, 231, 305, 302, 151, 202, 182,
   181, 143, 146, 146)), .Dim = as.integer(c(60, 2)),
   .Dimnames = list(NULL, c("Weight", "Disp.")))
```

```
  bagplot(cardata,factor=3,show.baghull=TRUE,
    show.loophull=TRUE,precision=1,dkmethod=2)
  title("car data Chambers/Hastie 1992")
  # points of y=x*x
  bagplot(x=1:30,y=(1:30)^2,verbose=FALSE,dkmethod=2)
  # one dimensional subspace
  bagplot(x=1:100,y=1:100)
}
\keyword{ misc }
```

Here are some important links:

```
http://www.cim.mcgill.ca/~lsimard/Pattern/TheBag.htm
http://www.math.yorku.ca/SCS/Gallery/bright-ideas.html
http://maven.smith.edu/~streinu/Research/LocDepth/algorithm.html
http://www.agoras.ua.ac.be/abstract/Bagbiv97.htm
http://www.agoras.ua.ac.be/Locdept.htm
http://article.gmane.org/gmane.comp.lang.r.general/25235
http://finzi.psych.upenn.edu/R/Rhelp02a/archive/45106.html
http://delivery.acm.org/10.1145/370000/365565/
 p690-miller.pdf?key1=365565&key2=9093786211&coll=GUIDE&
 dl=GUIDE&CFID=53086693&CFTOKEN=38519152
http://www.cs.tufts.edu/research/geometry/half_space/
http://www.statistik.tuwien.ac.at/public/filz/students/edavis/
  ws0607/skriptum/page134.html
```

4 The definition of bagplot

The funciton `bagplot` is a container that calls the two functions `compute.bagplot` and `plot.bagplot`. The first one generates an object of class `bagplot` and the second one is called by the generic plot function.

29 ⟨*define* `bagplot` 29⟩ ≡

```
⟨define compute.bagplot 30⟩
⟨define plot.bagplot 69⟩
bagplot<-function(x,y,
   factor=3, # expanding factor for bag to get the loop
   na.rm=FALSE, # should 'NAs' values be removed or exchanged
   approx.limit=300, # limit
   show.outlier=TRUE,# if TRUE outlier are shown
   show.whiskers=TRUE, # if TRUE whiskers are shown
   show.looppoints=TRUE, # if TRUE points in loop are shown
   show.bagpoints=TRUE, # if TRUE points in bag are shown
   show.loophull=TRUE, # if TRUE loop is shown
   show.baghull=TRUE, # if TRUE bag is shown
   create.plot=TRUE, # if TRUE a plot is created
   add=FALSE, # if TRUE graphical elements are added to actual plot
   pch=16,cex=.4, # some graphical parameters
   dkmethod=2, # in 1:2; there are two methods for approximating the bag
   precision=1, # controls precision of computation
   verbose=FALSE,debug.plots="no", # tools for debugging
   col.loophull="#aaccff", # Alternatives: #ccffaa, #ffaacc
   col.looppoints="#3355ff", # Alternatives: #55ff33, #ff3355
   col.baghull="#7799ff", # Alternatives: #99ff77, #ff7799
   col.bagpoints="#000088", # Alternatives: #008800, #880000
   transparency=FALSE, ... # to define further parameters of plot
){
```

```
  ⟨version of bagplot 1⟩
  bo<-compute.bagplot(x=x,y=y,factor=factor,na.rm=na.rm,
        approx.limit=approx.limit,dkmethod=dkmethod,
        precision=precision,verbose=verbose,debug.plots=debug.plots)
  if(create.plot){
    plot(bo,
     show.outlier=show.outlier,
     show.whiskers=show.whiskers,
     show.looppoints=show.looppoints,
     show.bagpoints=show.bagpoints,
     show.loophull=show.loophull,
     show.baghull=show.baghull,
     add=add,pch=pch,cex=cex,...,
     verbose=verbose,
     col.loophull=col.loophull,
     col.looppoints=col.looppoints,
     col.baghull=col.baghull,
     col.bagpoints=col.bagpoints,
     transparency=transparency
    )
  }
  invisible(bo)
}
```

`compute.bagplot` computes the relevant values to allow `plot.bagplot` to draw the bagplot.

30 ⟨*define* `compute.bagplot` 30⟩ ≡

```
compute.bagplot<-function(x,y,
   factor=3, # expanding factor for bag to get the loop
   na.rm=FALSE, # should NAs removed or exchanged
   approx.limit=300, # limit
   dkmethod=2, # in 1:2; method 2 is recommended
   precision=1, # controls precision of computation
   verbose=FALSE,debug.plots="no" # tools for debugging
){
  ⟨version of bagplot 1⟩
  ⟨body of compute.bagplot 31⟩
}
```

4.1 The body of `compute.bagplot`

Here you see the main steps of the construction of a bagplot.

31 ⟨*body of* `compute.bagplot` 31⟩ ≡

```
  ⟨init 33⟩
  ⟨check and handle linear case 46⟩
  ⟨standardize data and compute: xyxy, xym, xysd 47⟩
  ⟨compute angles between points 48⟩
  ⟨compute hdepths 49⟩
  ⟨find k 50⟩
  ⟨compute hdepths of test points to find center 51⟩
  if(dkmethod==1){
    ⟨method one: find hulls of D_k and D_{k-1} 55⟩
  }else{
    ⟨method two: find hulls of D_k and D_{k-1} 56⟩
```

```
}
⟨find value of lambda 64⟩
⟨find hull.bag 65⟩
⟨find hull.loop 66⟩
⟨find points outside of bag but inside loop 67⟩
⟨find hull of loop 68⟩
⟨output result 32⟩
```

4.2 Output of `bagplot`

The following table of output values of `bagplot` is copy from section 2:

`center`	Tukey median
`hull.loop`	set of points of polygon that defines the loop
`hull.bag`	set of points of polygon that defines the bag
`hull.center`	region of points with maximal ldepth
`pxy.outlier`	outlier
`pxy.outer`	outer points
`pxy.bag`	points in bag
`hdepth`	location depth of data points in xy
`is.one.dim`	is `TRUE` if data set is one dimensional
`prdata`	result of PCA
`random.seed`	random.seed that is set by bagplot
`xydata`	data set
`xy`	sample of data set

These elements are return as a list.

32 ⟨*output result* 32⟩ ≡

```
res<-list(
 center=center,
 hull.center=hull.center,
 hull.bag=hull.bag,
 hull.loop=hull.loop,
 pxy.bag=pxy.bag,
 pxy.outer=if(length(pxy.outer)>0) pxy.outer else NULL,
 pxy.outlier=if(length(pxy.outlier)>0) pxy.outlier else NULL,
 hdepths=hdepth,
 is.one.dim=is.one.dim,
 prdata=prdata,
 random.seed=random.seed,
 xy=xy,xydata=xydata
)
if(verbose) res<-c(res,list(exp.dk=exp.dk,exp.dk.1=exp.dk.1,hdepth=hdepth))
class(res)<-"bagplot"
return(res)
```

4.3 Initilization of `bagplot`

Points with identical coordinates may result in numerical problem. Therefore, some noise may be added to the data – for this feature the comment signs have to be deleted.

33 ⟨*init* 33⟩ ≡

```
# define some functions
```

```
⟨define function win 34⟩
⟨define function out.of.polygon 35⟩
⟨define function cut.z.pg 36⟩
⟨define function find.cut.z.pg 37⟩
⟨define function hdepth.of.points 39⟩
⟨define function expand.hull 40⟩
⟨define function cut.p.sl.p.sl 38⟩
⟨define function pos.to.pg 45⟩
⟨define find.polygon.center 54⟩
# check input
xydata<-if(missing(y)) x else cbind(x,y)
if(is.data.frame(xydata)) xydata<-as.matrix(xydata)
if(any(is.na(xydata))){
  if(na.rm){ xydata<-xydata[!apply(is.na(xydata),1,any),,drop=FALSE]
    print("Warning: NA elements have been removed!!")
  }else{
    xy.means<-colMeans(xydata,na.rm=TRUE)
    for(j in 1:ncol(xydata)) xydata[is.na(xydata[,j]),j]<-xy.means[j]
    print("Warning: NA elements have been set to mean values!!")
  }
}
if(nrow(xydata)<3) {print("not enough data points"); return()}
# select sample in case of a very large data set
very.large.data.set<-nrow(xydata)>approx.limit
set.seed(random.seed<-13)
if(very.large.data.set){
  ind<-sample(seq(nrow(xydata)),size=approx.limit)
  xy<-xydata[ind,]
} else xy<-xydata
n<-nrow(xy)
points.in.bag<-floor(n/2)
# if jittering is needed
# the following two lines can be activated
#xy<-xy+cbind(rnorm(n,0,.0001*sd(xy[,1])),
#             rnorm(n,0,.0001*sd(xy[,2])))
if(verbose) cat("end of initialization")
```

4.4 Some local functions to find intersection points

win: After a lot of experiments the function `atan2` is found to compute the gradient in fastest way.

34 ⟨*define function* `win` 34⟩ ≡

```
win<-function(dx,dy){  atan2(y=dy,x=dx) }
```

out.of.polygon: The function `out.of.polygon` checks if the points of `xy` are within the polygon `pg` (return value FALSE) or not (return value TRUE).

35 ⟨*define function* `out.of.polygon` 35⟩ ≡

```
out.of.polygon<-function(xy,pg){
  if(nrow(pg)==1)  return(xy[,1]==pg[1] & xy[,2]==pg[2])
  m<-nrow(xy<-matrix(xy,ncol=2)); n<-nrow(pg)
  limit<--abs(1E-10*diff(range(pg)))
  pgn<-cbind(diff(c(pg[,2],pg[1,2])),-diff(c(pg[,1],pg[1,1])))
```

```
   S<-matrix(colMeans(xy),1,2)
   dxy<-cbind(S[1]-pg[,1],S[2]-pg[,2])
   S.in.pg<-all(limit<apply(dxy*pgn,1,sum))
   if(!all(limit<apply(dxy*pgn,1,sum))){
     pg<-pg[n:1,]; pgn<--pgn[n:1,]
   }
   in.pg<-rep(TRUE,m)
   for(j in 1:n){
     dxy<-xy-matrix(pg[j,],m,2,byrow=TRUE)
     in.pg<-in.pg & limit<(dxy%*%pgn[j,])
   }
   return(!in.pg)
 }
```

This version of `out.of.polygon` is based on the following algorithm:

1. compute the orthogonal vectors of the sides of the polygon pointing to the interior
2. compute the vectors which starts in the corners of the polygon and ends in a point to be tested
3. check if all the angles between the pairs of associated vectors lie between $-\pi/2$ and $\pi/2$ which is equivalent to get positive signs of the inner products of the associated vectors only.

For more than 2000 test points the new version is 100 times faster than the old one.

cut.z.pg: `cut.z.pg` finds the intersection points of lines defined by `p1x,p1y,p2x,p2y` and lines that contains `zx,zy` and origin.

36 ⟨*define function* `cut.z.pg` 36⟩ ≡

```
 cut.z.pg<-function(zx,zy,p1x,p1y,p2x,p2y){
   a2<-(p2y-p1y)/(p2x-p1x); a1<-zy/zx
   sx<-(p1y-a2*p1x)/(a1-a2); sy<-a1*sx
   sxy<-cbind(sx,sy)
   h<-any(is.nan(sxy))||any(is.na(sxy))||any(Inf==abs(sxy))
   if(h){
   if(!exists("verbose")) verbose<-FALSE
     if(verbose) cat("special")
     # points on line defined by line segment
     h<-0==(a1-a2) & sign(zx)==sign(p1x)
        sx<-ifelse(h,p1x,sx); sy<-ifelse(h,p1y,sy)
     h<-0==(a1-a2) & sign(zx)!=sign(p1x)
        sx<-ifelse(h,p2x,sx); sy<-ifelse(h,p2y,sy)
     # line segment vertical
     #   & center NOT ON line segment
     h<-p1x==p2x & zx!=p1x & p1x!=0
        sx<-ifelse(h,p1x,sx); sy<-ifelse(h,zy*p1x/zx,sy)
     #   & center ON line segment
     h<-p1x==p2x & zx!=p1x & p1x==0
        sx<-ifelse(h,p1x,sx); sy<-ifelse(h,0,sy)
     #   & center ON line segment & point on line
     h<-p1x==p2x & zx==p1x & p1x==0 & sign(zy)==sign(p1y)
        sx<-ifelse(h,p1x,sx); sy<-ifelse(h,p1y,sy)
     h<-p1x==p2x & zx==p1x & p1x==0 & sign(zy)!=sign(p1y)
        sx<-ifelse(h,p1x,sx); sy<-ifelse(h,p2y,sy)
     #  points identical to end points of line segment
     h<-zx==p1x & zy==p1y; sx<-ifelse(h,p1x,sx); sy<-ifelse(h,p1y,sy)
     h<-zx==p2x & zy==p2y; sx<-ifelse(h,p2x,sx); sy<-ifelse(h,p2y,sy)
```

```
    # point of z is center
    h<-zx==0 & zy==0; sx<-ifelse(h,0,sx); sy<-ifelse(h,0,sy)
    sxy<-cbind(sx,sy)
  } # end of special cases
  #if(verbose){ print(rbind(a1,a2));print(cbind(zx,zy,p1x,p1y,p2x,p2y,sxy))}
  if(!exists("debug.plots")) debug.plots<-"no"
  if(debug.plots=="all"){
    segments(sxy[,1],sxy[,2],zx,zy,col="red")
    segments(0,0,sxy[,1],sxy[,2],type="l",col="green",lty=2)
    points(sxy,col="red")
  }
  return(sxy)
}
```

find.cut.z.pg: `find.cut.z.pg` finds the intersection points of the lines defined by `z` and center `center` and polygon `pg`.

37 ⟨*define function* `find.cut.z.pg` 37⟩ ≡

```
find.cut.z.pg<-function(z,pg,center=c(0,0),debug.plots="no"){
  if(!is.matrix(z)) z<-rbind(z)
  if(1==nrow(pg)) return(matrix(center,nrow(z),2,TRUE))
  n.pg<-nrow(pg); n.z<-nrow(z)
  # center z and pg
  z<-cbind(z[,1]-center[1],z[,2]-center[2])
  pgo<-pg; pg<-cbind(pg[,1]-center[1],pg[,2]-center[2])
  if(!exists("debug.plots")) debug.plots<-"no"
  if(debug.plots=="all"){
           plot(rbind(z,pg,0),bty="n"); points(z,pch="p")
           lines(c(pg[,1],pg[1,1]),c(pg[,2],pg[1,2]))}
  # find angles of pg und z
  apg<-win(pg[,1],pg[,2])
  apg[is.nan(apg)]<-0; a<-order(apg); apg<-apg[a]; pg<-pg[a,]
  az<-win(z[,1],z[,2])
  # find line segments
  segm.no<-apply((outer(apg,az,"<")),2,sum)
  segm.no<-ifelse(segm.no==0,n.pg,segm.no)
  next.no<-1+(segm.no %% length(apg))
  # compute cut points
  cuts<-cut.z.pg(z[,1],z[,2],pg[segm.no,1],pg[segm.no,2],
                           pg[next.no,1],pg[next.no,2])
  # rescale
  cuts<-cbind(cuts[,1]+center[1],cuts[,2]+center[2])
  return(cuts)
}
```

cut.p.sl.p.sl: `cut.p.sl.p.sl` finds the intersection point of two lines. Both lines are described by a point and its slope. Remember:

$$y = y_1 + m_1(x - x_1)$$

If both slopes are identical an `inf`-value will be returned.

38 ⟨*define function* `cut.p.sl.p.sl` 38⟩ ≡

```
cut.p.sl.p.sl<-function(xy1,m1,xy2,m2){
  sx<-(xy2[2]-m2*xy2[1]-xy1[2]+m1*xy1[1])/(m1-m2)
```

```
    sy<-xy1[2]-m1*xy1[1]+m1*sx
    if(!is.nan(sy)) return( c(sx,sy) )
    if(abs(m1)==Inf) return( c(xy1[1],xy2[2]+m2*(xy1[1]-xy2[1])) )
    if(abs(m2)==Inf) return( c(xy2[1],xy1[2]+m1*(xy2[1]-xy1[1])) )
  }
```

4.5 A function to compute the h-depths of data points

hdepth.of.points: `hdepth.of.points` computes the h-depths of test points `tp`. local variable `ident` stores the number of identical points. Algorithmical aspects of finding the h-depth are later discussed in: ⟨*find kkk-hull:* `pg` 57⟩

39 ⟨*define function* `hdepth.of.points` 39⟩ ≡

```
  hdepth.of.points<-function(tp,n){
    n.tp<-nrow(tp)
    tphdepth<-rep(0,n.tp); dpi<-2*pi-0.000001
    minusplus<-c(rep(-1,n),rep(1,n))
    for(j in 1:n.tp) {
      dx<-tp[j,1]-xy[,1]; dy<-tp[j,2]-xy[,2]
      a<-win(dx,dy)+pi; h<-a<10;a<-a[h]; ident<-sum(!h)
      init<-sum(a < pi); a.shift<-(a+pi) %% dpi
      minusplus<-c(rep(-1,length(a)),rep(1,length(a))) ## 070824
      h<-cumsum(minusplus[order(c(a,a.shift))])
      tphdepth[j]<-init+min(h)+1 # +1 because of the point itself!!
      # tphdepth[j]<-init+min(h)+ident; cat("SUMME",ident)
    }
    tphdepth
  }
```

4.6 A function to expand the hull

expand.hull: `expand.hull` expands polygon `pk` without changing the depth of its points. `k` is the depth and `resolution` is the number of points to be checked during expandation.

40 ⟨*define function* `expand.hull` 40⟩ ≡

```
  expand.hull<-function(pg,k){
    ⟨find end points of line segments: center → pg → pg0 41⟩
    ⟨search for points with critical hdepth 42⟩
    ⟨find additional points between the line segments 43⟩
    ⟨compute hull pg.new 44⟩
  }
```

At first we search the intersection points of the hull of the data set with the lines beginning in the center and running through the points of `pg`. Then test points on the segments defined by these intersection points and the points of `pg` will be generated by using a vector `lam`.

41 ⟨*find end points of line segments: center* → `pg` → `pg0` 41⟩ ≡

```
  resolution<-floor(20*precision)
  pg0<-xy[hdepth==1,]
  pg0<-pg0[chull(pg0[,1],pg0[,2]),]
  end.points<-find.cut.z.pg(pg,pg0,center=center,debug.plots=debug.plots)
  lam<-((0:resolution)^1)/resolution^1
```

The test is performed in two stages. In the interval from start point to end point `resolution` test points are tested concerning their h-depth. The critical interval is divided again to find a better limit.

42 ⟨*search for points with critical hdepth* 42⟩ ≡
```
pg.new<-pg
for(i in 1:nrow(pg)){
  tp<-cbind(pg[i,1]+lam*(end.points[i,1]-pg[i,1]),
            pg[i,2]+lam*(end.points[i,2]-pg[i,2]))
  hd.tp<-hdepth.of.points(tp,nrow(xy))
  ind<-max(sum(hd.tp>=k),1)
  if(ind<length(hd.tp)){  # hd.tp[ind]>k &&
    tp<-cbind(tp[ind,1]+lam*(tp[ind+1,1]-tp[ind,1]),
              tp[ind,2]+lam*(tp[ind+1,2]-tp[ind,2]))
    hd.tp<-hdepth.of.points(tp,nrow(xy))
    ind<-max(sum(hd.tp>=k),1)
  }
  pg.new[i,]<-tp[ind,]
}
pg.new<-pg.new[chull(pg.new[,1],pg.new[,2]),]
# cat("depth pg.new", hdepth.of.points(pg.new,n))
```

Between the spurts we interpolated additional directions and find additional limits by the same procedure.

43 ⟨*find additional points between the line segments* 43⟩ ≡
```
pg.add<-0.5*(pg.new+rbind(pg.new[-1,],pg.new[1,]))
# end.points<-find.cut.z.pg(pg,pg0,center=center)
end.points<-find.cut.z.pg(pg.add,pg0,center=center) ## 070824
for(i in 1:nrow(pg.add)){
  tp<-cbind(pg.add[i,1]+lam*(end.points[i,1]-pg.add[i,1]),
            pg.add[i,2]+lam*(end.points[i,2]-pg.add[i,2]))
  hd.tp<-hdepth.of.points(tp,nrow(xy))
  ind<-max(sum(hd.tp>=k),1)
  if(ind<length(hd.tp)){ # hd.tp[ind]>k &&
    tp<-cbind(tp[ind,1]+lam*(tp[ind+1,1]-tp[ind,1]),
              tp[ind,2]+lam*(tp[ind+1,2]-tp[ind,2]))
    hd.tp<-hdepth.of.points(tp,nrow(xy))
    ind<-max(sum(hd.tp>=k),1)
  }
  pg.add[i,]<-tp[ind,]
}
# cat("depth pg.add", hdepth.of.points(pg.add,n))
```

Finally the hull of the limits is computed and our numerical solution of hull(d_k). `pg.new` is the output of `expand.hull`.

44 ⟨*compute hull* `pg.new` 44⟩ ≡
```
pg.new<-rbind(pg.new,pg.add)
pg.new<-pg.new[chull(pg.new[,1],pg.new[,2]),]
```

4.7 A function to find the position of points respectively to a polygon

pos.to.pg: `pos.to.pg` finds the position of points `z` relative to a polygon `pg` If a point is below the polygon `"lower"` is returned otherwise `"upper"`.

45 ⟨*define function* pos.to.pg 45⟩ ≡

```
pos.to.pg<-function(z,pg,reverse=FALSE){
  if(reverse){
    int.no<-apply(outer(pg[,1],z[,1],">="),2,sum)
    zy.on.pg<-pg[int.no,2]+pg[int.no,3]*(z[,1]-pg[int.no,1])
  }else{
    int.no<-apply(outer(pg[,1],z[,1],"<="),2,sum)
    zy.on.pg<-pg[int.no,2]+pg[int.no,3]*(z[,1]-pg[int.no,1])
  }
  ifelse(z[,2]<zy.on.pg, "lower","higher")
}
```

4.8 Check if data set is one dimensional

Now the local functions are ready for usage. To detect a data set being one dimensional we apply `prcomp()`. In the one dimensional case we construct a boxplot by hand.

46 ⟨*check and handle linear case* 46⟩ ≡

```
prdata<-prcomp(xydata)
is.one.dim<-(min(prdata[[1]])/max(prdata[[1]]))<0.0001
if(is.one.dim){
  if(verbose) cat("data set one dimensional")
  center<-colMeans(xydata)
  res<-list(xy=xy,xydata=xydata,prdata=prdata,
            is.one.dim=is.one.dim,center=center)
  class(res)<-"bagplot"
  return(res)
}
if(verbose) cat("data not linear")
```

4.9 Standardize data and compute h-depths of points

For numerical reasons we standardize the data set: `xyxy`. Some computations takes place on the standardized copy `xyxy` of `xy`.

47 ⟨*standardize data and compute:* xyxy, xym, xysd 47⟩ ≡

```
xym<-apply(xy,2,mean); xysd<-apply(xy,2,sd)
xyxy<-cbind((xy[,1]-xym[1])/xysd[1],(xy[,2]-xym[2])/xysd[2])
```

For each data point we compute the directions to all the points and determine the angles of the directions. This information helps us to find the h-depths of the points. For friends of complexity: the angles between all pair of points are computed in $O(n^2 \log n)$ time because of sorting the columns of a $(n \times n)$-matrix. The angle between identical points is coded by entry 1000.

48 ⟨*compute angles between points* 48⟩ ≡

```
dx<-(outer(xy[,1],xy[,1],"-"))
dy<-(outer(xy[,2],xy[,2],"-"))
alpha<-atan2(y=dy,x=dx); diag(alpha)<-1000
for(j in 1:n) alpha[,j]<-sort(alpha[,j])
alpha<-alpha[-n,] ; m<-n-1
## quick look inside, just for check
if(debug.plots=="all"){
  plot(xy,bty="n"); xdelta<-abs(diff(range(xy[,1]))); dx<-xdelta*.3
  for(j in 1:n) {
```

```
    p<-xy[j,]; dy<-dx*tan(alpha[,j])
    segments(p[1]-dx,p[2]-dy,p[1]+dx,p[2]+dy,col=j)
    text(p[1]-xdelta*.02,p[2],j,col=j)
  }
}
if(verbose) print("end of computation of angles")
```

We compute the h-depths in $O(n^2 \log(n))$. The `NaN` angles are extracted because they indicate points with identical coordinates. For each point we find the h-depth by the following algorithm: At first we count the number of angles of the actual point within interval $[0, \pi)$. This is equivalent to the number of points above the actual point. Then we rotate the $y = 0$-line counterclockwise and increment the initial counter if an additional point emerges and we decrement the counter if a point / angle leaves the halve plain.
The median is defined as the gravity center of all points with maximal h-depth. As usually some problems were induced by equality of angles. One reaction was to add some shift to compare with slightly modified π-values.

49 ⟨*compute hdepths* 49⟩ ≡
```
hdepth<-rep(0,n); dpi<-2*pi-0.000001; mypi<-pi-0.000001
minusplus<-c(rep(-1,m),rep(1,m))
for(j in 1:n) {
  a<-alpha[,j]+pi; h<-a<10; a<-a[h]; init<-sum(a < mypi) # hallo
  a.shift<-(a+pi) %% dpi
  minusplus<-c(rep(-1,length(a)),rep(1,length(a))) ## 070824
  h<-cumsum(minusplus[order(c(a,a.shift))])
  hdepth[j]<-init+min(h)+1 # or do we have to count identical points?
# hdepth[j]<-init+min(h)+sum(xy[j,1]==xy[,1] & xy[j,2]==xy[,2])
}
if(verbose){print("end of computation of hdepth:"); print(hdepth)}
## quick look inside, just for a check
if(debug.plots=="all"){
  plot(xy,bty="n")
  xdelta<-abs(diff(range(xy[,1]))); dx<-xdelta*.1
  for(j in 1:n) {
    a<-alpha[,j]+pi; a<-a[a<10]; init<-sum(a < pi)
    a.shift<-(a+pi) %% dpi
    minusplus<-c(rep(-1,length(a)),rep(1,length(a))) ## 070824
    h<-cumsum(minusplus[ao<-(order(c(a,a.shift)))])
    no<-which((init+min(h)) == (init+h))[1]
    p<-xy[j,]; dy<-dx*tan(alpha[,j])
    segments(p[1]-dx,p[2]-dy,p[1]+dx,p[2]+dy,col=j,lty=3)
    dy<-dx*tan(c(sort(a),sort(a))[no])
    segments(p[1]-5*dx,p[2]-5*dy,p[1]+5*dx,p[2]+5*dy,col="black")
    text(p[1]-xdelta*.02,p[2],hdepth[j],col=1,cex=2.5)
  }
}
```

We determine the h-depth k so that the following condition holds:
(the number of points of h-depth greater or equal k is lower or equal to the number of data points staying in the bag) and (the number of points of h-depth greater equal $k - 1$ is greater $n/2$):

$$\#D_k \leq \text{points in the bag:} n/2 < \#D_{k-1}.$$

50 ⟨*find* k 50⟩ ≡
```
hd.table<-table(sort(hdepth))
d.k<-cbind(dk=rev(cumsum(rev(hd.table))),
           k =as.numeric(names(hd.table)))
```

```
k.1<-sum(points.in.bag<d.k[,1])

# if(nrow(d.k)>1){ # version 09/2005, error in data set 1 of Meuleman
if(nrow(d.k)>k.1){ # instead of >2 now >k.1 # 070827
  k<-d.k[k.1+1,2]
} else {
  k<-d.k[k.1,2]
}
if(verbose){cat("numbers of members of dk:"); print(hd.table)}
if(verbose){cat("end of computation of k, k=",k)}
```

4.10 Find the center of the data set

The two dimensional median is the center of gravity of the polygon of the points (not data points) with maximal h-depths.
We check some inner test points to find the maximal h-depth. Then we look for the boundery of the area of points with this h-depth.
This procedure has been tested with the Ben Greiner data using: ⟨*test: data set of Ben Greiner* 14⟩

51 ⟨*compute hdepths of test points to find center* 51⟩ ≡

```
center<-apply(xy[which(hdepth==max(hdepth)),,drop=FALSE],2,mean)
hull.center<-NULL
if(5<nrow(xy)&&length(hd.table)>2){
  n.p<-floor(c(32,16,8)[1+(n>50)+(n>200)]*precision)
  h<-cands<-xy[rev(order(hdepth))[1:6],]
  cands<-cands[chull(cands[,1],cands[,2]),]; n.c<-nrow(cands)
  if(is.null(n.c))cands<-h
  ⟨check some points to find the maximal h-depth 52⟩
  ⟨find the polygon of points of maximal h-depth 53⟩
  if(verbose){cat("hull.center",hull.center); print(table(tphdepth)) }
}
if(verbose) cat("center depth:",hdepth.of.points(rbind(center),n)-1)
if(verbose){print("end of computation of center"); print(center)}
```

52 ⟨*check some points to find the maximal h-depth* 52⟩ ≡

```
xyextr<-rbind(apply(cands,2,min),apply(cands,2,max))
xydel<-2*(xyextr[2,]-xyextr[1,])/n.p
h1<-seq(xyextr[1,1],xyextr[2,1],length=n.p)
h2<-seq(xyextr[1,2],xyextr[2,2],length=n.p)
tp<-cbind(matrix(h1,n.p,n.p)[1:n.p^2],
          matrix(h2,n.p,n.p,TRUE)[1:n.p^2])
tphdepth<-max(hdepth.of.points(tp,n))-1
```

For finding the area of maximal h-depth we use an algorithm that has been implemented for finding the bag, see below. ⟨*find kkk-hull:* `pg` 57⟩

53 ⟨*find the polygon of points of maximal h-depth* 53⟩ ≡

```
⟨initialize angles, ang 60⟩
kkk<-tphdepth
if(verbose){cat("max-hdepth found:"); print(kkk)}
⟨find kkk-hull: pg 57⟩
hull.center<-cbind(pg[,1]*xysd[1]+xym[1],pg[,2]*xysd[2]+xym[2])
center<-find.polygon.center(hull.center)
```

A function to compute the center of gravity of a polygon. The function `find.polygon.center` determines the center of gravity of a polygon.

54 ⟨*define* `find.polygon.center` 54⟩ ≡

```
find.polygon.center<-function(xy){
  ## if(missing(xy)){n<-50;x<-rnorm(n);y<-rnorm(n); xy<-cbind(x,y)}
  ## xy<-xy[chull(xy),]
  if(length(xy)==2) return(xy[1:2])
  ## partition polygon into triangles
  n<-length(xy[,1]); mxy<-colMeans(xy)
  xy2<-rbind(xy[-1,],xy[1,]); xy3<-cbind(rep(mxy[1],n),mxy[2])
  ## determine areas and centers of gravity of triangles
  S<-(xy+xy2+xy3)/3
  F2<-abs((xy[,1]-xy3[,1])*(xy2[,2]-xy3[,2])-
        (xy[,2]-xy3[,2])*(xy2[,1]-xy3[,1]))
  ## compute center of gravity of polygon
  lambda<-F2/sum(F2)
  SP<-colSums(cbind(S[,1]*lambda,S[,2]*lambda))
  return(SP)
}
```

We compute the convex hull of D_k: polygon `pdk` and the hull of D_{k-1}: polygon `pdk.1`.
`pdk` represents inner polygon and `pdk.1` outer one.
Then polygon `pdk` and `pdk.1` are enlarged without changing its h-depth: `exp.dk`, `exp.dk.1`-

55 ⟨*method one: find hulls of D_k and D_{k-1}* 55⟩ ≡

```
# inner hull of bag
xyi<-xy[hdepth>=k,,drop=FALSE]
pdk<-xyi[chull(xyi[,1],xyi[,2]),,drop=FALSE]
# outer hull of bag
xyo<-xy[hdepth>=(k-1),,drop=FALSE]
pdk.1<-xyo[chull(xyo[,1],xyo[,2]),,drop=FALSE]
if(verbose)cat("hull computed:")
#; if(verbose){print(pdk); print(pdk.1) }
if(debug.plots=="all"){
  plot(xy,bty="n")
  h<-rbind(pdk,pdk[1,]); lines(h,col="red",lty=2)
  h<-rbind(pdk.1,pdk.1[1,]);lines(h,col="blue",lty=3)
  points(center[1],center[2],pch=8,col="red")
}
exp.dk<-expand.hull(pdk,k)
exp.dk.1<-expand.hull(exp.dk,k-1) # pdk.1,k-1,20)
```

The new approach to find the hull works as follows:
For a given k we move lines with different slopes from outside of the cloud to the center and stop if k points are crossed. To keep things simple we rotate the data points so that we have only move a vertical line.

1. define directions / angles for hdepth search
2. standardize data set to get appropiate directions
3. computation of D_k polygon and restandardization
4. computation of D_{k-1} polygon and restandardization

We determine the hulls on the base of the standardized data `xyxy`.

56 ⟨*method two: find hulls of D_k and D_{k-1}* 56⟩ ≡
```
⟨initialize angles, ang 60⟩
# standardization of data set xyxy is used
kkk<-k
⟨find kkk-hull: pg 57⟩
exp.dk<-cbind(pg[,1]*xysd[1]+xym[1],pg[,2]*xysd[2]+xym[2])
if(kkk>1) kkk<-kkk-1
⟨find kkk-hull: pg 57⟩
exp.dk.1<-cbind(pg[,1]*xysd[1]+xym[1],pg[,2]*xysd[2]+xym[2])
```

The polygon for h-depth *kkk* is constructed in a loop. In each step we consider one direction / angle.

57 ⟨*find kkk-hull:* `pg` 57⟩ ≡
```
⟨initialize loop of directions 59⟩
for(ia in seq(angles)[-1]){
  ⟨body of loop of directions 58⟩
}
⟨combination of lower and upper polygon 61⟩
```

At first we search the limiting points for every direction by rotating the data set and then we determine the quantiles $x_{k/n}$ and $x_{(n+1-k)/n}$. With this points we construct a upper polygon `pg` and a lower one `pgl` that limit the hull we are looking for. To update a polygon we have to find the line segments of the polygon that are cut by the lines of slope `a` through the limiting points as well as the intersection points.

58 ⟨*body of loop of directions* 58⟩ ≡
```
# determine critical points pnew and pnewl of direction a
### cat("ia",ia)
a<-angles[ia]; angtan<-ang[ia]; xyt<-xyxy%*%c(cos(a),-sin(a)); xyto<-order(xyt)
ind.k <-xyto[kkk]; ind.kk<-xyto[n+1-kkk]; pnew<-xyxy[ind.k,]; pnewl<-xyxy[ind.kk,]
if(debug.plots=="all") points(pnew[1],pnew[2],col="red")
# new limiting lines are defined by pnew / pnewl and slope a
# find segment of polygon that is cut by new limiting line and cut
if(abs(angtan)>1e10){  ###  cat("y=c case")
    pg.no<-sum(pg[,1]<pnew[1])
    cutp<-c(pnew[1],pg[pg.no,2]+pg[pg.no,3]*(pnew[1]-pg[pg.no,1]))
    pg.nol<-sum(pgl[,1]>=pnewl[1])
    cutpl<-c(pnewl[1],pgl[pg.nol,2]+pgl[pg.nol,3]*(pnewl[1]-pgl[pg.nol,1]))
}else{    ### cat("normal case")
    pg.inter<-pg[,2]-angtan*pg[,1]; pnew.inter<-pnew[2]-angtan*pnew[1]
    pg.no<-sum(pg.inter<pnew.inter)
    cutp<-cut.p.sl.p.sl(pnew,ang[ia],pg[pg.no,1:2],pg[pg.no,3])
    pg.interl<-pgl[,2]-angtan*pgl[,1]; pnew.interl<-pnewl[2]-angtan*pnewl[1]
    pg.nol<-sum(pg.interl>pnew.interl)
    cutpl<-cut.p.sl.p.sl(pnewl,angtan,pgl[pg.nol,1:2],pgl[pg.nol,3])
}
# update pg, pgl
pg<-rbind(pg[1:pg.no,],c(cutp,angtan),c(cutp[1]+dxy, cutp[2]+angtan*dxy,NA))
pgl<-rbind(pgl[1:pg.nol,],c(cutpl,angtan),c(cutpl[1]-dxy, cutpl[2]-angtan*dxy,NA))
⟨debug: plot within for loop 62⟩
```

To initialize the loop we construct the first polygons (upper one: `pg`, lower one: `pgl`) by vertical lines. `dxdy` is a step that is larger than the range of the standardized data set.

59 ⟨*initialize loop of directions* 59⟩ ≡
```
ia<-1; a<-angles[ia]; xyt<-xyxy%*%c(cos(a),-sin(a)); xyto<-order(xyt)
# initial for upper part
```

```
ind.k <-xyto[kkk]; cutp<-c(xyxy[ind.k,1],-10)
dxy<-diff(range(xyxy))
pg<-rbind(c(cutp[1],-dxy,Inf),c(cutp[1],dxy,NA))
# initial for lower part
ind.kk<-xyto[n+1-kkk]; cutpl<-c(xyxy[ind.kk,1],10)
pgl<-rbind(c(cutpl[1],dxy,Inf),c(cutpl[1],-dxy,NA))
⟨debug: plot ini 63⟩
```

It is necessary to initialize the angles of the directions. If the data set is very large we will check fewer directions than in case of a small data set. If the data set is small the choice of the angles may be improved by using the observed angles defined by the slopes of lines running through the pairs of the points.

60 ⟨*initialize* `angles`, `ang` 60⟩ ≡

```
# define direction for hdepth search
num<-floor(c(417,351,171,85,67,43)[sum(n>c(1,50,100,150,200,250))]*precision)
num.h<-floor(num/2); angles<-seq(0,pi,length=num.h)
ang<-tan(pi/2-angles)
```

The combination of the polygons is a little bit complicated because sometimes at the right and at left margin an additional intersection point has to be computed and integrated. `l` in front of a variable name indicates the left margin whereas the right one is coded by `r`. Letter `l` (`u`) at the end of a name is short for lower and upper.

61 ⟨*combination of lower and upper polygon* 61⟩ ≡

```
## plot(xyxy[,1:2],xlim=c(-.5,+.5),ylim=c(-.5,.50))
## lines(pg,type="b",col="red")
## lines(pgl,type="b",col="blue")
# remove first and last points and multiple points
limit<-1e-10
pg<-pg[c(TRUE,(abs(diff(pg[,1]))>limit)|(abs(diff(pg[,2]))>limit)),]
pgl<-pgl[c(TRUE,(abs(diff(pgl[,1]))>limit)|(abs(diff(pgl[,2]))>limit)),]
pg<-pg[-nrow(pg),][-1,,drop=FALSE]
pgl<-pgl[-nrow(pgl),][-1,,drop=FALSE]
# determine position according to the other polygon
#   cat("relative position: lower polygon")
indl<-pos.to.pg(pgl,pg)
#   print(cbind(signif(pgl,3),indl))
#   cat("relative position: upper polygon")
indu<-pos.to.pg(pg,pgl,TRUE)
#   print(cbind(signif(pg,3),indu))
sr<-sl<-NULL
# right region
if(indu[(npg<-nrow(pg))]=="lower" & indl[1]=="higher"){
  # cat("in if of right region: the upper polynom is somewhere lower")
  #  checking from the right: last point of lower polygon that is NOT ok
  rnuml<-which(indl=="lower")[1]-1
  #  checking from the left: last point of upper polygon that is ok
  rnumu<-npg+1-which(rev(indu=="higher"))[1]
  #  special case all points of lower polygon are upper
  if(is.na(rnuml)) rnuml<-sum(pg[rnumu,1]<pgl[,1])
  #  special case all points of upper polygon are lower
  if(is.na(rnumu)) rnumu<-sum(pg[,1]<pgl[rnuml,1])
  xyl<-pgl[rnuml,]; xyu<-pg[rnumu,]
  ## cat("right"); print(rnuml); print(xyl)
  ## cat("right"); print(rnumu); print(xyu)
  sr<-cut.p.sl.p.sl(xyl[1:2],xyl[3],xyu[1:2],xyu[3])
}
# left region
```

```
if(indl[(npgl<-nrow(pgl))]=="higher"&indu[1]=="lower"){
  # cat("in if of left region: the upper polynom is somewhere lower")
  #  checking from the right: last point of lower polygon that is ok
  lnuml<-npgl+1-which(rev(indl=="lower"))[1]
  #  checking from the left: last point of upper polygon that is NOT ok
  lnumu<-which(indu=="higher")[1]-1
  #  special case all points of lower polygon are upper
  if(is.na(lnuml)) lnuml<-sum(pg[lnumu,1]<pgl[,1])
  #  special case all points of upper polygon are lower
  if(is.na(lnumu)) lnumu<-sum(pg[,1]<pgl[lnuml,1])
  xyl<-pgl[lnuml,]; xyu<-pg[lnumu,]
  ## cat("left"); print(lnuml); print(xyl)
  ## cat("left"); print(lnumu); print(xyu)
  sl<-cut.p.sl.p.sl(xyl[1:2],xyl[3],xyu[1:2],xyu[3])
}
pg<-rbind(pg [indu=="higher",1:2,drop=FALSE],sr,
          pgl[indl=="lower", 1:2,drop=FALSE],sl)
  ## print(pg)
if(debug.plots=="all") lines(rbind(pg,pg[1,]),col="red")
pg<-pg[chull(pg[,1],pg[,2]),]
```

62 ⟨*debug: plot within for loop* 62⟩ ≡

```
#########################################
#### cat("angtan",angtan,"pg.no",pg.no,"pkt:",pnew)
# if(ia==stopp) lines(pg,type="b",col="green")
if(debug.plots=="all"){
  points(pnew[1],pnew[2],col="red")
  hx<-xyxy[ind.k,c(1,1)]; hy<-xyxy[ind.k,c(2,2)]
  segments(hx,hy,c(10,-10),hy+ang[ia]*(c(10,-10)-hx),lty=2)
#  text(hx+rnorm(1,,.1),hy+rnorm(1,,.1),ia)
# print(pg)
# if(ia==stopp) lines(pgl,type="b",col="green")
  points(cutpl[1],cutpl[2],col="red")
  hx<-xyxy[ind.kk,c(1,1)]; hy<-xyxy[ind.kk,c(2,2)]
  segments(hx,hy,c(10,-10),hy+ang[ia]*(c(10,-10)-hx),lty=2)
#  text(hx+rnorm(1,,.1),hy+rnorm(1,,.1),ia)
# print(pgl)
}
```

63 ⟨*debug: plot ini* 63⟩ ≡

```
if(debug.plots=="all"){ plot(xyxy,type="p",bty="n")
# text(xy,,1:n,col="blue")
# hx<-xy[ind.k,c(1,1)]; hy<-xy[ind.k,c(2,2)]
# segments(hx,hy,c(10,-10),hy+ang[ia]*(c(10,-10)-hx),lty=2)
# text(hx+rnorm(1,,.1),hy+rnorm(1,,.1),ia)
}
```

4.11 Finding of the bag

To find the bag the function `expand.hull` computes not an exact solution but a numerical approximation. `k.1` indicates the polygon (`exp.dk.1`) with h-depth $(k-1)$. `k.1+1` will usually point to h-depth k (polygon: `exp.dk`), to the inner polygon.

In computing λ we follow Miller et al. (1999). They define λ as the relative distance from the bag to the inner contour and they compute it *by* $\lambda = (50 - J)/(L - J)$, *where* D_k *contains* $J\%$ *of the original points and* D_{k-1} *contains* $L\%$ *of the original points:*

$$\lambda = \frac{50 - J}{L - J} = \frac{n/2 - \#D_k}{\#D_{k-1} - \#D_k} = \frac{\text{number in bag} - \text{number in inner contour}}{\text{number in outer contour} - \text{number in inner contour}}$$

If bag and inner contour is identical then $\lambda \leftarrow 0$.
`k.1` is the number of the rows of `dk` that represent points within the bag / h-depths greater $n/2$.

64 ⟨*find value of* `lambda` 64⟩ ≡
```
if(nrow(d.k)==k.1||nrow(d.k)==1)  lambda<-0 else {
  lambda<-(n/2-d.k[k.1+1,1])/(d.k[k.1,1]-d.k[k.1+1,1])
}
if(verbose) cat("lambda",lambda)
```

The bag is constructed by lambda * outer polygon + (1-lambda)* inner polygon.
In former versions it happened that some lines of `h` get `NaN` values because `nrow(d.k)==2` and `k.1==2` (e.g. example of Wouter Meuleman). This problem doesn't occur no longer but to be sure we have an additional look at `h`.

65 ⟨*find* `hull.bag` 65⟩ ≡
```
cut.on.pdk.1<-find.cut.z.pg(exp.dk,  exp.dk.1,center=center)
cut.on.pdk  <-find.cut.z.pg(exp.dk.1,exp.dk,  center=center)
# expand inner polgon exp.dk
h1<-(1-lambda)*exp.dk+lambda*cut.on.pdk.1
# shrink outer polygon exp.dk.1
h2<-(1-lambda)*cut.on.pdk+lambda*exp.dk.1
h<-rbind(h1,h2);
h<-h[!is.nan(h[,1])&!is.nan(h[,2]),]
hull.bag<-h[chull(h[,1],h[,2]),]
if(verbose)cat("bag completed:") #if(verbose) print(hull.bag)
if(debug.plots=="all"){   lines(hull.bag,col="red") }
```

4.12 Computation of the loop

The loop is found by expanding `hull.bag` by factor `factor`.

66 ⟨*find* `hull.loop` 66⟩ ≡
```
hull.loop<-cbind(hull.bag[,1]-center[1],hull.bag[,2]-center[2])
hull.loop<-factor*hull.loop
hull.loop<-cbind(hull.loop[,1]+center[1],hull.loop[,2]+center[2])
if(verbose) cat("loop computed")
```

Now we identify the points of the bag, the outliers and the outer points. Remark: There may be some points of h-depth $(k-1)$ that are members of the bag. If the data set is very large we will not check whether the h-depth $(k-1)$ points are in the bag.

67 ⟨*find points outside of bag but inside loop* 67⟩ ≡
```
if(!very.large.data.set){
  pxy.bag    <-xydata[hdepth>= k   ,,drop=FALSE]
  pkt.cand   <-xydata[hdepth==(k-1),,drop=FALSE]
  pkt.not.bag<-xydata[hdepth< (k-1),,drop=FALSE]
  if(length(pkt.cand)>0){
    outside<-out.of.polygon(pkt.cand,hull.bag)
    if(sum(!outside)>0)
      pxy.bag    <-rbind(pxy.bag,     pkt.cand[!outside,])
```

```
    if(sum( outside)>0)
      pkt.not.bag<-rbind(pkt.not.bag, pkt.cand[ outside,])
  }
}else {
  extr<-out.of.polygon(xydata,hull.bag)
  pxy.bag    <-xydata[!extr,]
  pkt.not.bag<-xydata[extr,,drop=FALSE]
}
if(length(pkt.not.bag)>0){
  extr<-out.of.polygon(pkt.not.bag,hull.loop)
  pxy.outlier<-pkt.not.bag[extr,,drop=FALSE]
  if(0==length(pxy.outlier)) pxy.outlier<-NULL
  pxy.outer<-pkt.not.bag[!extr,,drop=FALSE]
}else{
  pxy.outer<-pxy.outlier<-NULL
}
if(verbose) cat("points of bag, outer points and outlier identified")
```

The points of the hull of the loop are stored in `hull.loop`.

68 ⟨*find hull of loop* 68⟩ ≡

```
hull.loop<-rbind(pxy.outer,hull.bag)
hull.loop<-hull.loop[chull(hull.loop[,1],hull.loop[,2]),]
if(verbose) cat("end of computation of loop")
```

4.13 The definition of `plot.bagplot`

Finally we have to draw the bagplot. This job is managed by a new plot method.

69 ⟨*define* `plot.bagplot` 69⟩ ≡

```
plot.bagplot<-function(x,
   show.outlier=TRUE,# if TRUE outlier are shown
   show.whiskers=TRUE, # if TRUE whiskers are shown
   show.looppoints=TRUE, # if TRUE points in loop are shown
   show.bagpoints=TRUE, # if TRUE points in bag are shown
   show.loophull=TRUE, # if TRUE loop is shown
   show.baghull=TRUE, # if TRUE bag is shown
   add=FALSE, # if TRUE graphical elements are added to actual plot
   pch=16,cex=.4, # to define further parameters of plot
   verbose=FALSE, # tools for debugging
   col.loophull="#aaccff", # Alternatives: #ccffaa, #ffaacc
   col.looppoints="#3355ff", # Alternatives: #55ff33, #ff3355
   col.baghull="#7799ff", # Alternatives: #99ff77, #ff7799
   col.bagpoints="#000088", # Alternatives: #008800, #880000
   transparency=FALSE,...
){
 ⟨version of bagplot 1⟩
 # transparency flag and color flags have been proposed by wouter
    if (transparency==TRUE) {
      col.loophull = paste(col.loophull, "99", sep="")
      col.baghull = paste(col.baghull, "99", sep="")
    }
 ⟨define function win 34⟩
 ⟨define function cut.z.pg 36⟩
 ⟨define function find.cut.z.pg 37⟩
 bagplotobj<-x
 for(i in seq(along=bagplotobj))
    eval(parse(text=paste(names(bagplotobj)[i],"<-bagplotobj[[",i,"]]")))
```

```
  if(is.one.dim){
      ⟨construct plot for one dimensional case and return 71⟩
  }
  ⟨construct bagplot as usual 70⟩
}
```

The following elements allows us to draw the bagplot: `xydata` (data set), `xy` (sample of data set), `hdepth` (location depth of data points in xy), `hull.loop` (points of polygon that define the loop), `hull.bag` (points of polygon that define the bag), `hull.center` (region of points with maximal ldepth), `pxy.outlier` (outlier), `pxy.outer` (outer points), `pxy.bag` (points in bag), `center` (Tukey median), `is.one.dim` is TRUE if data set is one dimensional, `prdata` result of PCA

70 ⟨*construct bagplot as usual* 70⟩ ≡

```
if(!add) plot(xydata,type="n",pch=pch,cex=cex,bty="n",...)
if(verbose) text(xy[,1],xy[,2],paste(as.character(hdepth)),cex=2)
# loop: --------------------------------------------------
if(show.loophull){ # fill loop
    h<-rbind(hull.loop,hull.loop[1,]); lines(h[,1],h[,2],lty=1)
    polygon(hull.loop[,1],hull.loop[,2],col=col.loophull)
}
if(show.looppoints && length(pxy.outer)>0){ # points in loop
    points(pxy.outer[,1],pxy.outer[,2],col=col.looppoints,pch=pch,cex=cex)
}
# bag: ---------------------------------------------------
if(show.baghull){ # fill bag
    h<-rbind(hull.bag,hull.bag[1,]); lines(h[,1],h[,2],lty=1)
    polygon(hull.bag[,1],hull.bag[,2],col=col.baghull)
}
if(show.bagpoints && length(pxy.bag)>0){ # points in bag
    points(pxy.bag[,1],pxy.bag[,2],col=col.bagpoints,pch=pch,cex=cex)
}
# whiskers
if(show.whiskers && length(pxy.outer)>0){
    debug.plots<-"not"
    if((n<-length(xy[,1]))<15){
      segments(xy[,1],xy[,2],rep(center[1],n),rep(center[2],n),
               col="red")
    }else{
      pkt.cut<-find.cut.z.pg(pxy.outer,hull.bag,center=center)
      segments(pxy.outer[,1],pxy.outer[,2],pkt.cut[,1],pkt.cut[,2],
               col="red")
    }
}
# outlier: -----------------------------------------------
if(show.outlier && length(pxy.outlier)>0){ # points in loop
      points(pxy.outlier[,1],pxy.outlier[,2],col="red",pch=pch,cex=cex)
}
# center:
if(exists("hull.center")&&length(hull.center)>2){
    h<-rbind(hull.center,hull.center[1,]); lines(h[,1],h[,2],lty=1)
    polygon(hull.center[,1],hull.center[,2],col="orange")
}
  points(center[1],center[2],pch=8,col="red")
if(verbose){
   h<-rbind(exp.dk,exp.dk[1,]); lines(h,col="blue",lty=2)
   h<-rbind(exp.dk.1,exp.dk.1[1,]); lines(h,col="black",lty=2)
   if(exists("tphdepth"))
      text(tp[,1],tp[,2],as.character(tphdepth),col="green")
```

```
    text(xy[,1],xy[,2],paste(as.character(hdepth)),cex=2)
    points(center[1],center[2],pch=8,col="red")
  }
  "bagplot plottet"
```

71 ⟨*construct plot for one dimensional case and return* 71⟩ ≡

```
   if(verbose) cat("data set one dimensional")
   prdata<-prdata[[2]];
   trdata<-xydata%*%prdata; ytr<-mean(trdata[,2])
   boxplotres<-boxplot(trdata[,1],plot=FALSE)
   dy<-0.1*diff(range(stats<-boxplotres$stats))
   dy<-0.05*mean(c(diff(range(xydata[,1])),
                   diff(range(xydata[,2]))))
   segtr<-rbind(cbind(stats[2:4],ytr-dy,stats[2:4],ytr+dy),
                cbind(stats[c(2,2)],ytr+c(dy,-dy),
                      stats[c(4,4)],ytr+c(dy,-dy)),
                cbind(stats[c(2,4)],ytr,stats[c(1,5)],ytr))
   segm<-cbind(segtr[,1:2]%*%t(prdata),
               segtr[,3:4]%*%t(prdata))
   if(!add) plot(xydata,type="n",bty="n",pch=16,cex=.2,...)
   extr<-c(min(segm[6,3],segm[7,3]),max(segm[6,3],segm[7,3]))
   extr<-extr+c(-1,1)*0.000001*diff(extr)
   xydata<-xydata[xydata[,1]<extr[1] |
                  xydata[,1]>extr[2],,drop=FALSE]
   if(0<nrow(xydata))points(xydata[,1],xydata[,2],pch=pch,cex=cex)
   segments(segm[,1],segm[,2],segm[,3],segm[,4],)
   return("one dimensional boxplot plottet")
```

In case of problems some additional plottings may be helpful.

72 ⟨*additional graphical comments if necessary* 72⟩ ≡

```
  # points(exp.dk[,1],exp.dk[,2],type="b",col="red")
  # points(exp.dk[,1],exp.dk[,2],type="b",col="green")
  # points(exp.dk.1[,1],exp.dk.1[,2],type="b",col="blue")
```

4.14 Some technical leftovers

4.14.1 Definition of `bagplot` on start

73 ⟨*start* 73⟩ ≡
⟨*define* `bagplot` 29⟩

4.14.2 Extracting the R code file `bagplot.R`

74 ⟨*some functions for generating bagplots* 74⟩ ≡
⟨*define* `bagplot` 29⟩

75 ⟨*call* `tangleR` *to extract tangle function* `bagplot()` 75⟩ ≡

```
  tangleR("bagplot.rev",expand.roots="some functions for generating bagplots",
          expand.root.start=FALSE)
```

5 Appendix

5.1 Some further examples – usefull for testing

76 ⟨*define rnorm data* `data`*, seed:* `seed`*, size:* `n` 76⟩ ≡
```
if(!exists("seed")) seed<-75 # 267 81 115
set.seed(seed)
#data<-matrix(sample(1:10000,size=2000),1000,2)
#data<-matrix(sample(1:10000,size=300),50,2)
if(!exists("n")) n<-100
data<-cbind(rnorm(n)+100,rnorm(n)+300)
par(mfrow=c(1,1))
```

77 ⟨*error* 77⟩ ≡
```
⟨define bagplot 29⟩
⟨data set 1 of Wouter Meuleman 13⟩
bagplot(a[,2],a[,3],verbose=TRUE,debug.plots="all",dkmethod=2)
```

78 ⟨* 17⟩+ ≡
```
seed<-222; n<-100
⟨define rnorm data data, seed: seed, size: n 76⟩
datan<-rbind(data,c(106,294)); par(mfrow=c(1,1))
datan[,2]<-datan[,2]*100
⟨define bagplot 29⟩
bagplot(datan,verbose=TRUE)
```

79 ⟨*quadratic* 6⟩+ ≡
```
⟨define bagplot 29⟩
bagplot(x=1:30,y=(1:30)^2,verbose=TRUE,dkmethod=2)
```

80 ⟨*error* 77⟩+ ≡
```
⟨define bagplot 29⟩
⟨data set 1 of Wouter Meuleman 13⟩
bagplot(a[,2],a[,3],verbose=TRUE,debug.plots="all",dkmethod=2)
```

81 ⟨* 17⟩+ ≡
```
par(mfrow=2:3)
⟨define bagplot 29⟩
bagplot(x=cos((1:100)/100*2*pi),y=-sin((1:100)/100*2*pi),
  precision=1,verbose=TRUE,dkmethod=2,debug.plot="all"); "ok"
```

5.2 Some old code chunks for comparison

82 ⟨*old version of the combination of lower and upper polygon* 82⟩ ≡
```
pg<-pg[-nrow(pg),][-1,,drop=FALSE]; pgl<-pgl[-nrow(pgl),][-1,,drop=FALSE]
indl<-pos.to.pg(pgl,pg); indu<-pos.to.pg(pg,pgl,TRUE)
npg<-nrow(pg); npgl<-nrow(pgl)
rnuml<-rnumu<-lnuml<-lnumu<-0; sl<-pg[1,1:2]; sr<-pgl[1,1:2]
# right region
if(indl[1]=="higher"&indu[npg]=="lower"){
  rnuml<-which(indl=="lower")[1]-1; xyl<-pgl[rnuml,] #
  rnumu<-which(rev(indu=="higher"))[1]; xyu<-pg[npg+1-rnumu,] #
  sr<-cut.p.sl.p.sl(xyl[1:2],xyl[3],xyu[1:2],xyu[3])
}
# left region
if(indl[npgl]=="higher"&indu[1]=="lower"){
  lnuml<-which(rev(indl=="lower"))[1]; xyl<-pgl[npgl+1-lnuml,] #
  lnumu<-which(indu=="higher")[1]-1; xyu<-pg[lnumu,] #?
  sl<-cut.p.sl.p.sl(xyl[1:2],xyl[3],xyu[1:2],xyu[3])
}
pgl<-pgl[(rnuml+1):(npgl-lnuml),1:2,drop=FALSE]
pg <-pg [(lnumu+1):(npg -rnumu),1:2,drop=FALSE]
pg<-rbind(pg,sr,pgl,sl)
pg<-pg[chull(pg[,1],pg[,2]),]
if(debug.plots=="all") lines(rbind(pg,pg[1,]),col="red")
```

In the old version of `out.of.polygon` the angles between the lines that are defined by a point of [[xy] and the vertices of `pg` are computed. If maximal angle $> 2\pi$ than the point is not an inner point of the polygon:

83 ⟨*old version: define function* `out.of.polygon` 83⟩ ≡
```
out.of.polygon<-function(xy,pg){
  if(nrow(pg)==1) return(pg)
  pgcenter<-apply(pg,2,mean) # not necessary
  pg<-cbind(pg[,1]-pgcenter[1],pg[,2]-pgcenter[2])# not necessary
  xy<-cbind(xy[,1]-pgcenter[1],xy[,2]-pgcenter[2])# not necessary
  extr<-rep(FALSE,nrow(xy))
  for(i in seq(nrow(xy))){
    alpha<-sort((win(xy[i,1]-pg[,1],xy[i,2]-pg[,2]))%%(2*pi))
    extr[i]<-pi<max(diff(alpha)) |
             pi<(alpha[1]+2*pi-alpha[length(alpha)])
  }
  extr
}
```

This was an alternative approach to find the center but the brute force method seems to be better.

84 ⟨*beta* 84⟩ ≡
```
#  lam<-matrix(runif(n.c*n.p),n.p,n.c)
  set.seed(13);  n.p.beta<-10*n.p
```

```
    lam<-matrix(rbeta(n.c*n.p.beta,.5,.5),n.p.beta,n.c)
    lam<-lam/matrix(apply(lam,1,sum),n.p.beta,n.c,FALSE)
    tp<-cbind( lam%*%cands[,1],lam%*%cands[,2])
    tphdepth<-hdepth.of.points(tp,n)
    hull.center<-tp[which(tphdepth==max(tphdepth)),,drop=FALSE]
    center<-apply(hull.center,2,mean)
    hull.center<-hull.center[chull(hull.center[,1],hull.center[,2]),]
```

85 ⟨*old version: check points on a grid to find center* 85⟩ ≡

```
    xyextr<-rbind(apply(cands,2,min),apply(cands,2,max))
    xydel<-2*(xyextr[2,]-xyextr[1,])/n.p
    h1<-seq(xyextr[1,1],xyextr[2,1],length=n.p)
    h2<-seq(xyextr[1,2],xyextr[2,2],length=n.p)
    tp<-cbind(matrix(h1,n.p,n.p)[1:n.p^2],
              matrix(h2,n.p,n.p,TRUE)[1:n.p^2])
    tphdepth<-hdepth.of.points(tp,n)
    hull.center<-tp[which(tphdepth>=(max(tphdepth))),,drop=FALSE]
    center<-apply(hull.center,2,mean)
    cands<-hull.center[chull(hull.center[,1],hull.center[,2]),,drop=FALSE]
    xyextr<-rbind(apply(cands,2,min),apply(cands,2,max))
    ##  xydel<-(xyextr[2,]-xyextr[1,])/n.p
    xyextr<-rbind(xyextr[1,]-xydel,xyextr[2,]+xydel)
    h1<-seq(xyextr[1,1],xyextr[2,1],length=n.p)
    h2<-seq(xyextr[1,2],xyextr[2,2],length=n.p)
    tp<-cbind(matrix(h1,n.p,n.p)[1:n.p^2],
              matrix(h2,n.p,n.p,TRUE)[1:n.p^2])
    tphdepth<-hdepth.of.points(tp,n)
    hull.center<-tp[which(tphdepth>=max(tphdepth)),,drop=FALSE]
    center<-apply(hull.center,2,mean)
    hull.center<-hull.center[chull(hull.center[,1],hull.center[,2]),]
```

86 ⟨*old: experiment for finding bag* 86⟩ ≡

```
  critical.angles.of.points<-function(tp){
    n.tp<-nrow(tp)
    tphdepth<-rep(0,n.tp); dpi<-2*pi-0.000001

  minusplus<-c(rep(-1,n),rep(1,n))
    result<-matrix(0,n.tp,4)
    for(j in 1:n.tp) {
      dx<-tp[j,1]-xy[,1]; dy<-tp[j,2]-xy[,2]
      a<-win(dx,dy)+pi; a<-a[a<10]; a<-sort(a)
      a.shift<-(a+pi) %% dpi
      h<-cumsum(minusplus[order(c(a,a.shift))])
      no<-which(min(h)==h); no<-c(no[1],no[length(no)])
      no<-c(no,1+((no-2)%%n))
  # print(no)
     result[j,]<-c(a,a)[no]
    }
  if(debug.plots=="all"){
    plot(xy,type="n")
    # points(xy);
    text(xy,as.character(hdepth))
    h<-rbind(tp,tp[1,]); lines(h)
    points(tp[1,,drop=FALSE],col="red")
    dx<-3; ro<-1
    dy<-dx*tan(result[ro,1])

  #  segments(tp[ro,1]-dx,tp[ro,2]-dy,tp[ro,1]+dx,tp[ro,2]+dy,col="orange")
    dy<-dx*tan(result[ro,3])
    segments(tp[ro,1]-dx,tp[ro,2]-dy,tp[ro,1]+dx,tp[ro,2]+dy,col="red")
    dy<-dx*tan(result[ro,2])
  #  segments(tp[ro,1]-dx,tp[ro,2]-dy,tp[ro,1]+dx,tp[ro,2]+dy,col="green")
    dy<-dx*tan(result[ro,4])
  #  segments(tp[ro,1]-dx,tp[ro,2]-dy,tp[ro,1]+dx,tp[ro,2]+dy,col="blue")
  }
    result
  }
  a.pdk<-critical.angles.of.points(pdk)
```

87 ⟨*old version to find* `lambda` 87⟩ ≡

```
  # old version based on polygon of data points
  if(nrow(d.k)>1){
    lambda<-1-(points.in.bag-d.k[k.1+1,1])/(d.k[k.1,1]-d.k[k.1+1,1])
  } else {
    lambda<-0.5
  }
```

88 ⟨*old* 88⟩ ≡

```
  pdk.1<-pdk.1-matrix(pcenter,nrow(pdk.1),2,byrow=TRUE)

  pcenter<-apply(pdk,2,mean)
  pdk<-pdk-matrix(pcenter,nrow(pdk),2,byrow=TRUE)
  ai<-win(pdk[,1],pdk[,2])
  a<-order(ai); ai<-ai[a]; pdk<-pdk[a,,drop=FALSE]
  ai<-win(pdk[,1],pdk[,2])
  ao<-win(pdk.1[,1],pdk.1[,2])
  a<-order(ao); ao<-ao[a]; pdk.1<-pdk.1[a,,drop=FALSE]
  ao<-win(pdk.1[,1],pdk.1[,2])
  # for display the two polygons in verbose mode we store them
```

89 ⟨*old: find bag* 89⟩ ≡
⟨*find points of outer polygon to be shift* NA⟩
⟨*find points to shift on inner polygon* NA⟩
⟨*shift points on polygons and construct* `hull.bag` NA⟩

Some points of the two polygon will be identical, so the subsets of points has to be moved.

90 ⟨*old: find points of outer polygon to be shift* 90⟩ ≡

```
  h1<-match(pdk.1[,1], pdk[,1]); h2<-match(pdk.1[,2], pdk[,2])
```

```
ind.pdk.1<-seq(along=h1); found<-!is.na(h1)
union.points<-pdk.1[found,2]==pdk[h1[found],2]
union.points<-ind.pdk.1[found][union.points]
ind.o.points.to.shift<-ind.pdk.1[-union.points]
outer.shift.points<-pdk.1[ind.o.points.to.shift,,drop=FALSE]
if(length(ai)==1){
 # inner polygon and center center identical / ai==NaN
 outer.shift.points<-   lambda *outer.shift.points
} else {
  for(i in seq(along=outer.shift.points[,1])){
   # get point
   xy0<-outer.shift.points[i,]
   # get segment of inner polygon
   ind1<-sum(ai<win(xy0[1],xy0[2])); if(ind1==0) ind1<-length(ai)
   ind2<-ind1+1; if(ind2>length(ai)) ind2<-1
   xy1<-pdk[ind1,]; xy2<-pdk[ind2,]
# determinate cut of inner segment and line (0,0) -> point
   lam<-solve(matrix(c(xy0,xy1-xy2),2,2))%*%xy1
   xy.cut<-lam[1]*xy0
   # determinate new position of point of outer polygon
   outer.shift.points[i,]<-   lambda *outer.shift.points[i,]+
                           (1-lambda)*xy.cut
  } # end of for
} # end of if
if(verbose) {cat("outer polygon points have been shifted:") }
```

91 ⟨*old: find points to shift on inner polygon* 91⟩ ≡

```
h1<-match(pdk[,1], pdk.1[,1]); h2<-match(pdk[,2], pdk.1[,2])
ind.i.points.to.shift<-is.na(h1)&is.na(h2)
inner.shift.points<-pdk[ind.i.points.to.shift,,drop=FALSE]
for(i in seq(along=inner.shift.points[,1])){
 # get point
 xy0<-inner.shift.points[i,]
 # get segment of outer polygon
 ind1<-sum(ao<win(xy0[1],xy0[2])); if(ind1==0) ind1<-length(ao)
 ind2<-ind1+1; if(ind2>length(ao)) ind2<-1
 xy1<-pdk.1[ind1,]; xy2<-pdk.1[ind2,]
 # determinate cut of outer segment and line (0,0) -> point
 lam<-solve(matrix(c(xy0,xy1-xy2),2,2))%*%xy1
 xy.cut<-lam[1]*xy0
 # determinate new position of point of inner polygon
 inner.shift.points[i,]<-(1-lambda)*inner.shift.points[i,]+
                             lambda *xy.cut
}
if(verbose) {cat("inner polygon points have been shifted:") }
```

92 ⟨*old: shift points on polygons and construct* `hull.bag` 92⟩ ≡

```
pdk[ind.i.points.to.shift,]<-inner.shift.points
pdk.1[ind.o.points.to.shift,]<-outer.shift.points
hull.bag<-rbind(pdk.1,pdk)
hull.bag<-hull.bag[chull(hull.bag[,1],hull.bag[,2]),,drop=FALSE]
if(verbose){cat("bag completed:"); print(hull.bag) }
```

93 ⟨*old: find value of* `lambda` *– old version* 93⟩ ≡

```
if(nrow(d.k)>1){
  lambda<-1-(points.in.bag-d.k[k.1+1,1])/(d.k[k.1,1]-d.k[k.1+1,1])
} else {
  lambda<-0.5
}
vt<-find.cut.z.pg(xy,pdk,center=center)
vt.1<-find.cut.z.pg(xy,pdk.1,center=center)
h<-cbind(xy[,1]-center[1],xy[,2]-center[2]); lz<-apply(h*h,1,sum)^0.5
h<-cbind(vt[,1]-center[1],vt[,2]-center[2]); lv<-apply(h*h,1,sum)^0.5
h<-cbind(vt.1[,1]-center[1],vt.1[,2]-center[2]); lv.1<-apply(h*h,1,sum)^0.5
lambda.i<-(lz-lv)/(lv.1-lv)
lambda.i<-(lambda.i[!is.na(lambda.i) & ! is.nan(lambda.i)])
# lambda<-median(lambda.i)
# cat("median? lambda",median(lambda.i))
if(lambda<0|lambda>1) lambda<-0.5
if(verbose) cat("lambda",lambda)
#  segm.no[is.na(segm.no)]<-1
#  cut.pkt[is.nan(cut.pkt)]<-z[is.nan(cut.pkt)]
#  cut.pkt<-cbind(cut.pkt[,1]+center[1],cut.pkt[,2]+center[2])
#  h<-is.na(cuts[,1])
#  if(any(h)){cut.pkt[h,1]<-pgo[1,1];cut.pkt[h,2]<-pgo[1,2]}
```

`# car data: lambda==0.6918136`

In this definition it follows: `lambda==1` iff bag is identical with inner polygon `exp.dk`.

94 ⟨*old: find value of* `lambda` 94⟩ ≡

```
vt<-find.cut.z.pg(xy,exp.dk,center=center)
vt.1<-find.cut.z.pg(xy,exp.dk.1,center=center)
h<-cbind(xy[,1]-center[1],xy[,2]-center[2]); lz<-apply(h*h,1,sum)^0.5
h<-cbind(vt[,1]-center[1],vt[,2]-center[2]); lv<-apply(h*h,1,sum)^0.5
h<-cbind(vt.1[,1]-center[1],vt.1[,2]-center[2]); lv.1<-apply(h*h,1,sum)^0.5
lambda.i<-(lz-lv)/(lv.1-lv)
lambda.i<-(lambda.i[!is.na(lambda.i) & ! is.nan(lambda.i)])
lambda<-median(lambda.i)
if(verbose) cat("\nmedian lambda",lambda)
if(lambda<0|lambda>1) lambda<-lambda<-min(1,max(0,median(lambda.i)))
```