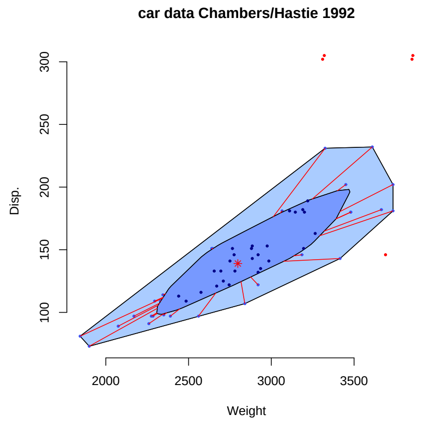
# A rough R Impementation of the Bagplot

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

April 13, 2006



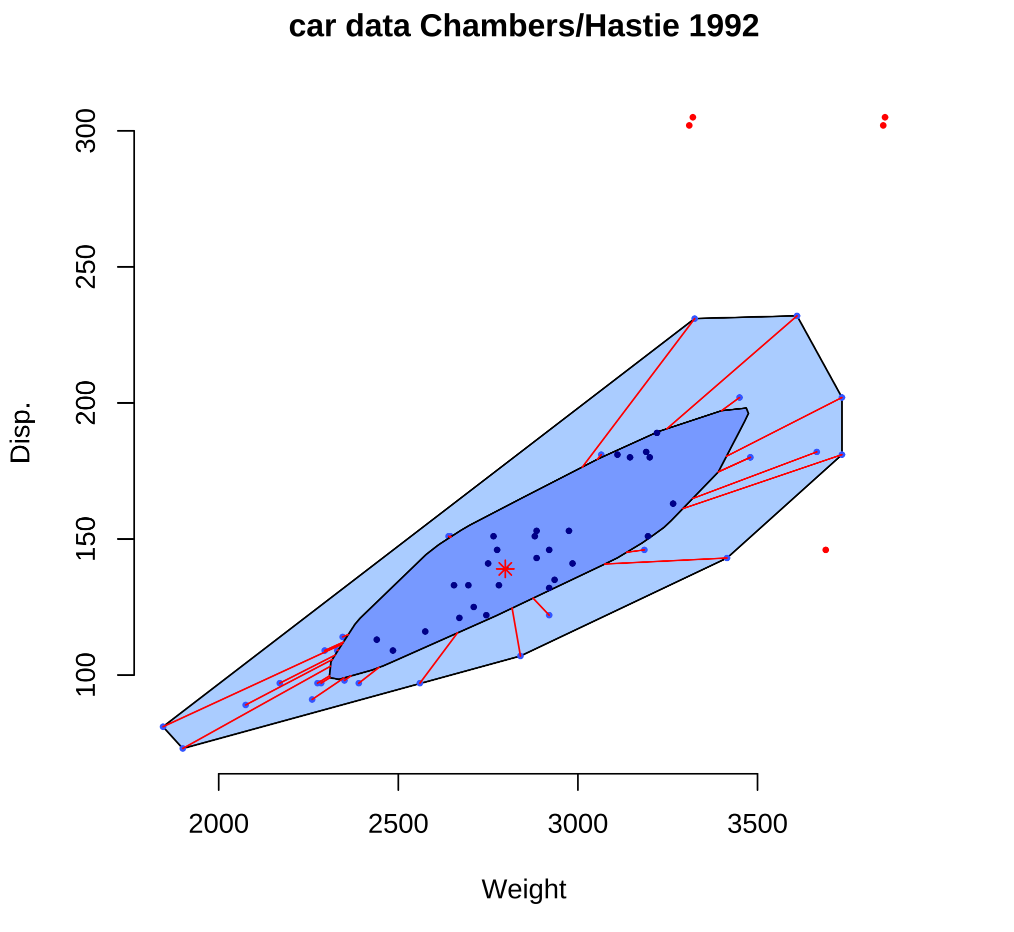

## Contents

# 1 Examples

## 1.1 Example: car data (Chambers / Hastie 1992)

1 ⟨*cardata* 1⟩≡

```
⟨define bagplot 16⟩
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)
#title("car data Chambers/Hastie 1992")
```




By the way Splus computes the Tukey median as 2806.63 139.513. In contrast our center is: 2801.4000 , 139.2667. In difference to Rousseeuw et al. our bagplot as well as the bagplot computed by Splus the data point of Nissan Van 4 is classified as outlier. To get the Splus result you have to download `bagplot*`, the car data and ...

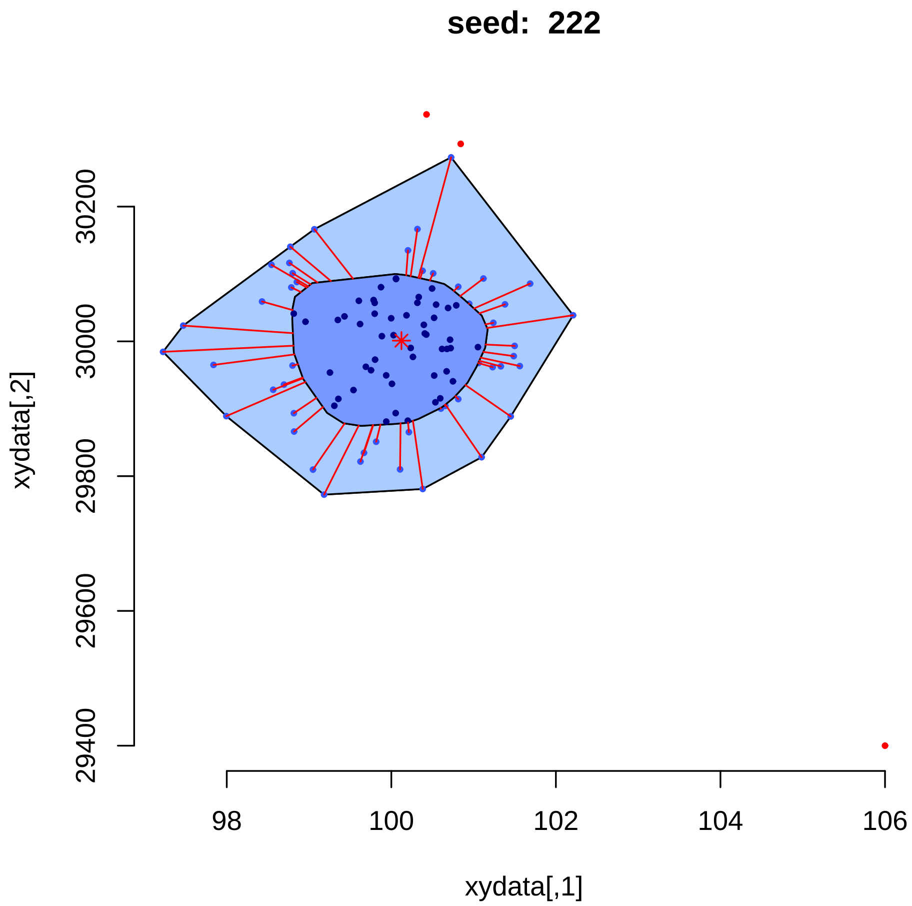
```
Splus CHAPTER bagplot.f
Splus make
Splus ...
> dyn.open("S.so"); source("bagplot.s")
> postscript("hello.ps"); bagplot(cardata[,1],cardata[,2]); dev.off()
```

## 1.2 The normal case

A bagplot of an rnorm sample plus one heavy outlier

2 ⟨*rnorm* 2⟩≡

```
lll<-221
⟨define data xy 58⟩
datan<-rbind(data,c(106,294)); par(mfrow=c(1,1))
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: ",lll))
```

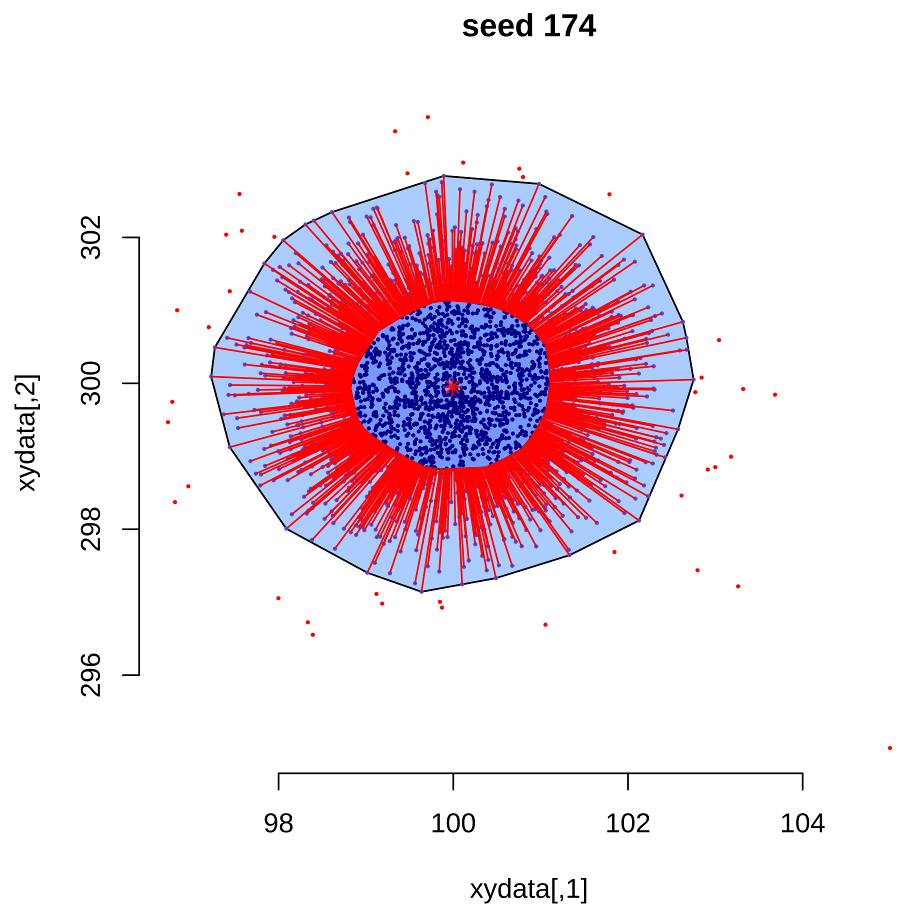


## 1.3 Large data sets

What about large data sets?

3 ⟨*large* 3⟩≡

```
lll<-173
if(!exists("lll")) lll<-75
set.seed(lll<-lll+1); print(lll)
n<-3000;datan<-cbind(rnorm(n)+100,rnorm(n)+300)
print(lll)
datan<-rbind(datan,c(105,295))
par(mfrow=c(1,1)) #; par(mfrow=2:3)
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",lll))
```

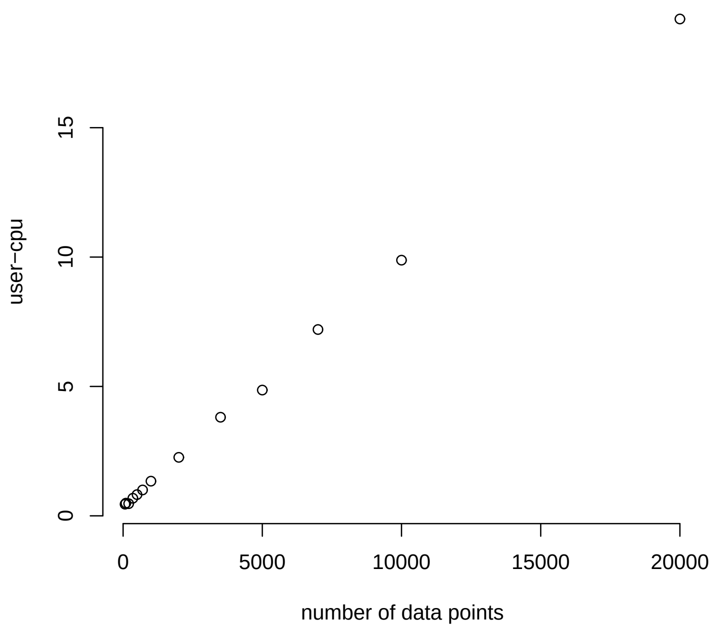


## 1.4 Size of data set

The time for computation increase with the number of observations. To get an imagination of the time needed look at the following experiment: We measure the times rnorm data sets of different sizes and plot the result.

4 ⟨*rnorm* 2⟩+≡

```
⟨define bagplot 16⟩
nn<-c(35,50,70,100,200); nn<-c(nn,10*nn,100*nn);nn<-nn[-(1:2)]
result<-1:length(nn)
for(j in seq(along=nn)){
  lll<-111; set.seed(lll); 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
```

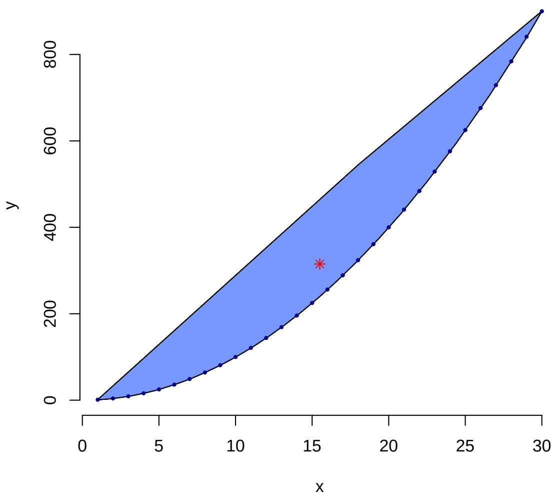



```
Thu Sep 22 16:53:14 2005
   70   100   200   350   500   700  1000  2000  3500  5000  7000 10000 20000
 0.45  0.49  0.47  0.68  0.82  1.00  1.34  2.26  3.81  4.86  7.20  9.88 19.20
```

## 1.5 Depth one data sets

What happens if all points are of depth 1?

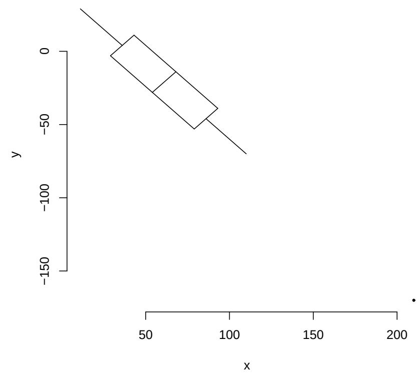
5 ⟨*quadratic* 5⟩≡
```
  ⟨define bagplot 16⟩
  bagplot(x=1:30,y=(1:30)^2,verbose=TRUE,dkmethod=2)
```




## 1.6 Degenerated data sets

What happens if the data are in a one dim subspace?

6 ⟨*onedim* 6⟩≡
```
  bagplot(x=10+c(1:100,200),y=30-c(1:100,200),verbose=FALSE)
```




Here is a second one dim data set.

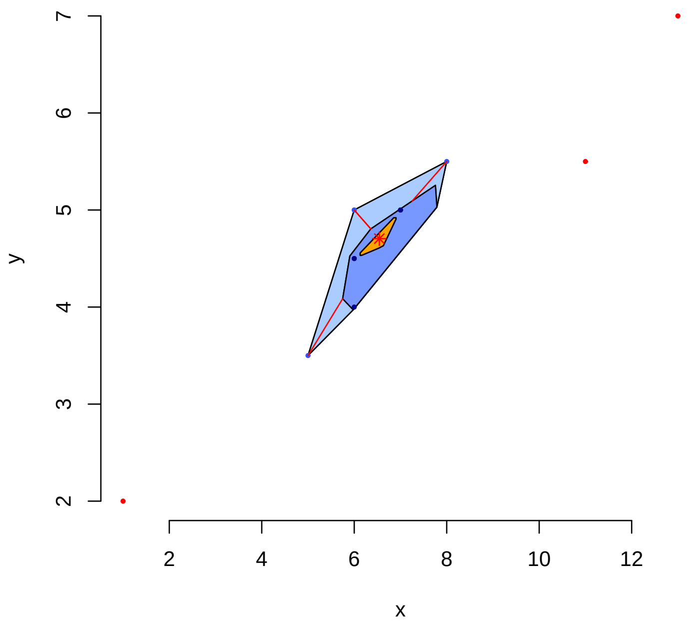
7 ⟨*one dim test* 7⟩≡
```
  bagplot(x=(1:100),y=(1:100),verbose=FALSE)
```

## 1.7 Data set from the mail of M. Maechler

The data set of M. Maechler is discussed within R-help. We are not shure if our boxplot is an approximation that is good enough. Maybe this doesn't matter because usually a data set is *in regular position* (Rousseeuw, Ruts 1998) that is we have no problems with identical coordinates. (In the car data set there are two points which are identical.)
M. Maechler wrote in a reply concerning a bagplot questiion that the correct Tukey median is (6.75 , 4.875 ) and not (6.542816, 4.707176) that is computed by our bagplot procedure.

8 ⟨*mm* 8⟩≡

```
#       hallo
⟨define bagplot 16⟩
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)
par(mfrow=c(1,1))
hallo<-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)
```

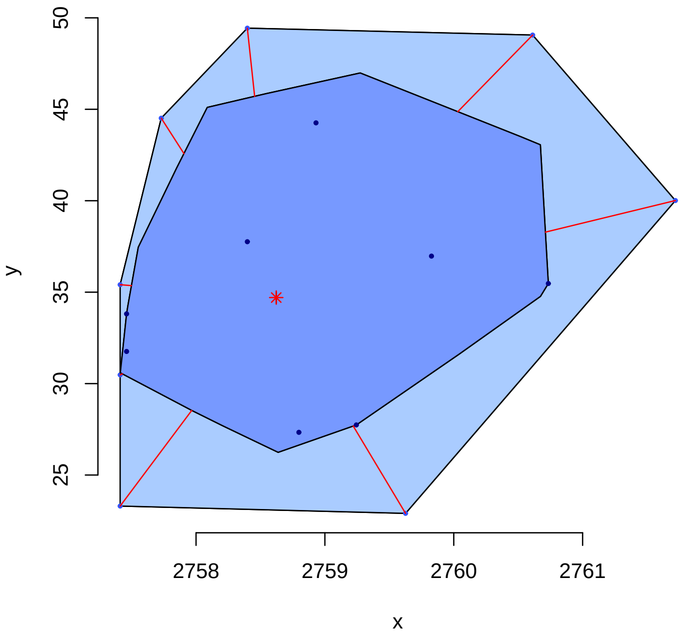


## 2 Data sets of Wouter Meuleman, running in an error with version 09/2005

The following data sets runs in an error because of some NaN values occured in computation *find* `hull.bag`.

9 ⟨*data set 2 of Wouter Meuleman* 9⟩≡

```
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 16⟩
bagplot(a[,2],a[,3],verbose=TRUE)
```




10 ⟨*data set 1 of Wouter Meuleman* 10⟩≡

```
a<-gsub("\n"," ",c("1  7766.734 38.86814 2  7768.329 34.50661 3  7769.335 21.14797 4  77
5  7776.913 17.97344 6  7768.221 22.27727 8  7771.719 43.62978 9  7773.056 20.22909 12 7
a<-unlist(strsplit(paste(a,collapse=""),"   "))
a<-as.numeric(a[a!=""])
a<-matrix(a,ncol=3,byrow=TRUE)
```

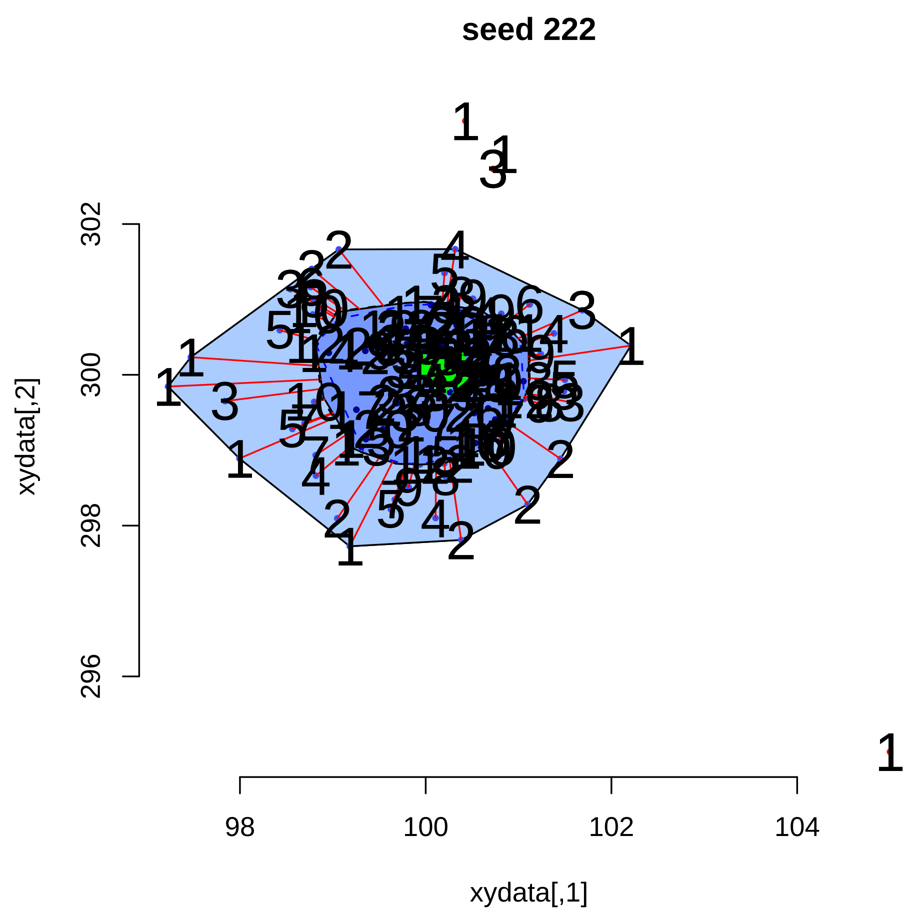
On 2006/02/17 some lines of code are changed so that no error occur. However, we have to check the resulting bagplot analyse, how to

11 ⟨*errorasdf* 11⟩≡
```
⟨define bagplot 16⟩
⟨data set 1 of Wouter Meuleman 10⟩
bagplot(a[,2],a[,3],verbose=TRUE,dkmethod=2)
```

## 2.1 Bagplot with additional graphical supplements

Verbose bagplot of a sample of 100 rnorm points and an outlier

12 ⟨*verbosetest* 12⟩≡
```
lll<-221
⟨define data xy 58⟩
datan<-rbind(data,c(105,295))
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",lll))
```

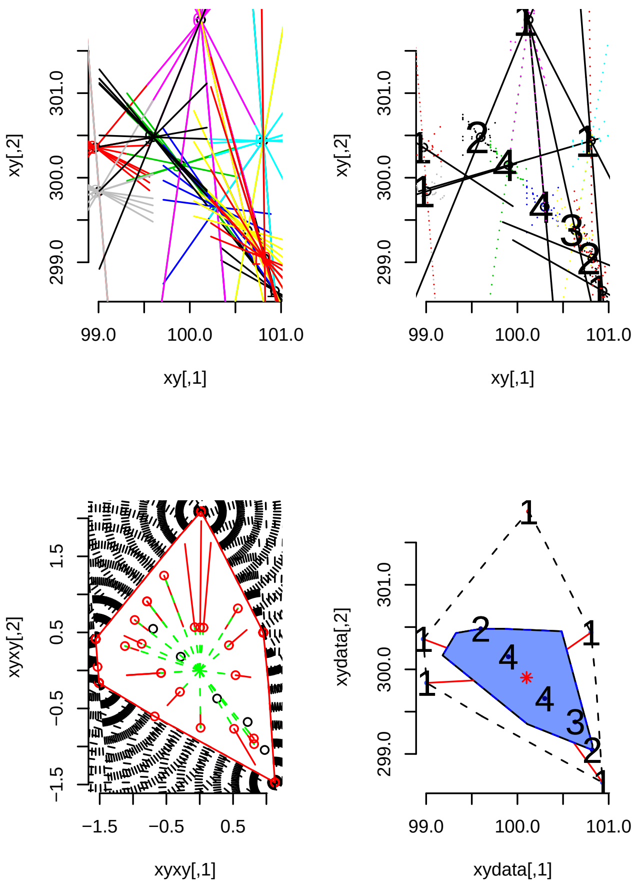


## 2.2 Debugging plots with additional elements

Here is an example of plots generated with option `debug.plots="all"`.

13 ⟨*debugplot* 13⟩≡

```
⟨define data xy 58⟩
datan<-rbind(data,c(120,280))
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)
```



# 3 Arguments of `bagplot`

... can be found by:

14 ⟨*args* 14⟩≡

```
args(bagplot)
```

A very short description can be found in the header of the function.

# 4 Links

Here are some 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/
```

# 5 The help page of bagplot

```
function (x, y, factor = 3, 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 = "")
```

15 ⟨*define help of* `bagplot` 15⟩≡

```
  \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, 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, 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{approx.limit}{ precision of approximation, 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,
currently under construction}
  \item{precision}{ precision of approximation, default: 1 }
  \item{verbose}{ automatic commenting of calculations
 }
  \item{debug.plots}{ developers' tool for debugging }
  \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 hull, the bag of bagplot. In the bag are 50 percent
of all points. The fence separates points in the fence from
points outside. It is computed by increasing the
the bag. The loop is defined as the convex polygon 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.
There are known difficulties with small data sets
(But I think it is not wise to make a (graphical)
summary of e.g. 10 points.)

In case people want to plot multiple (overlappIng) bagplots, it is convenient if the plo
the \code{transparency} flag has been added to the bagplot
command.
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 PD
\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{
  The development of the function has not been finished.
```

```
  Version 02/2006 }
\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 }
```

# 6 The definition of bagplot

The funciton`bagplot` is a container that calls the two function `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.

16 ⟨*define* `bagplot` 16⟩≡

```
⟨define compute.bagplot 17⟩
⟨define plot.bagplot 54⟩
bagplot<-function(x,y,
   factor=3, # expanding factor for bag to get the loop
   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 precisionn 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
){
  bo<-compute.bagplot(x=x,y=y,factor=factor,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
    )
  }
}
```

`compute.bagplot` computes the neccessary values to allow `plot.bagplot` plot the bagplot.

17 ⟨*define* `compute.bagplot` 17⟩≡

```
compute.bagplot<-function(x,y,
   factor=3, # expanding factor for bag to get the loop
   approx.limit=300, # limit
   dkmethod=2, # in 1:2; there are two methods for approximating the bag
   precision=1, # controls precisionn of computation
   verbose=FALSE,debug.plots="no" # tools for debugging
){
  ⟨body of compute.bagplot 18⟩
}
```

18 ⟨*body of* `compute.bagplot` 18⟩≡

```
#pwolf 050921 / 060217
⟨init 20⟩
⟨check and handle linear case 33⟩
⟨compute angles between points 34⟩
⟨compute hdepths 35⟩
⟨find k 36⟩
⟨compute hdepths of test points to find center 37⟩
if(dkmethod==1){
  ⟨method one: find hulls of D_k and D_{k-1} 40⟩
}else{
  ⟨method two: find hulls of D_k and D_{k-1} 41⟩
}
⟨find value of lambda 48⟩
⟨find hull.bag 50⟩
⟨find hull.loop 51⟩
⟨find points outside of bag but inside loop 52⟩
⟨find hull of loop 53⟩
⟨output result 19⟩
```

Output of `bagplot` is a list of essential components of the computation. To identify singular points, use `identify()`. There is the list that is returned:
`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 T if data set is one dimensional
`prdata` result of PCA
The elements are concatenated in a list and returned.

19 ⟨*output result* 19⟩≡
```
assign(".Random.seed",save.seed,env=.GlobalEnv)
res<-list(
 center=center,
 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,
 hull.center=hull.center,
 hull.bag=hull.bag,
 hull.loop=hull.loop,
 hdepths=hdepth,
 is.one.dim=is.one.dim,
 prdata=prdata,
 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)
```

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

20 ⟨*init* 20⟩≡

```
# define some functions
⟨define function win 21⟩
⟨define function out.of.polygon 22⟩
⟨define function cut.z.pg 23⟩
⟨define function find.cut.z.pg 24⟩
⟨define function hdepth.of.points 25⟩
⟨define function expand.hull 26⟩
⟨define function cut.p.sl.p.sl 31⟩
⟨define function pos.to.pg 32⟩
# check input
xydata<-if(missing(y)) x else cbind(x,y)
if(is.data.frame(xydata)) xydata<-as.matrix(xydata)
# select sample in case of large data set
very.large.data.set<-nrow(xydata)>approx.limit
if(!exists(".Random.seed")) set.seed(13)
save.seed<-.Random.seed
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])))
assign(".Random.seed",save.seed,env=.GlobalEnv)
if(verbose) cat("end of initialization")
```

after a lot of experiments the function `atan2` is found to do the job best

21 ⟨*define function* `win` 21⟩≡

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

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

22 ⟨*define function* `out.of.polygon` 22⟩≡

```
out.of.polygon<-function(xy,pg){
  if(nrow(pg)==1) return(pg)
  pgcenter<-apply(pg,2,mean)
  pg<-cbind(pg[,1]-pgcenter[1],pg[,2]-pgcenter[2])
  xy<-cbind(xy[,1]-pgcenter[1],xy[,2]-pgcenter[2])
  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
}
```

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

23 ⟨*define function* `cut.z.pg` 23⟩≡

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

24 ⟨*define function* `find.cut.z.pg` 24⟩≡

```
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)
}
```

`hdepth.of.points` computes the hdepths of test points `tp`.

25 ⟨*define function* `hdepth.of.points` 25⟩≡

```
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
    h<-cumsum(minusplus[order(c(a,a.shift))])
    tphdepth[j]<-init+min(h)+1
    # tphdepth[j]<-init+min(h)+ident; cat("SUMME",ident)
  }
  tphdepth
}
```

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

26 ⟨*define function* `expand.hull` 26⟩≡

```
expand.hull<-function(pg,k){
  ⟨find end points of line segments: mean → pg → pg0 27⟩
  ⟨search for points with critical hdepth 28⟩
  ⟨find additional points between the line segments 29⟩
  ⟨compute hull pg.new 30⟩
}
```

At first we search the cut 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 cut points and the points of `pg` will be generated by using a vector `lam`.

27 ⟨*find end points of line segments: mean* $\rightarrow$ `pg` $\rightarrow$ `pg0` 27⟩≡

```
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 form 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.

28 ⟨*search for points with critical hdepth* 28⟩≡

```
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.

29 ⟨*find additional points between the line segments* 29⟩≡

```
pg.add<-0.5*(pg.new+rbind(pg.new[-1,],pg.new[1,]))
end.points<-find.cut.z.pg(pg,pg0,center=center)
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`.

30 ⟨*compute hull* `pg.new` 30⟩≡

```
pg.new<-rbind(pg.new,pg.add)
pg.new<-pg.new[chull(pg.new[,1],pg.new[,2]),]
```

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

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

31 ⟨*define function* `cut.p.sl.p.sl` 31⟩≡

```
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])) )
}
```

`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"`.

32 ⟨*define function* `pos.to.pg` 32⟩≡

```
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")
}
```

Now the local function are ready for usage.
To detect a one dimensional data set we apply prcomp. Then we construct a boxplot by hand.

33 ⟨*check and handle linear case* 33⟩≡

```
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")
```

For friends of complexity: the angles between all pair of points are computed in $O(n^2 \log n)$ time. The angle between identical points is set to 1000.

34 ⟨*compute angles between points* 34⟩≡

```
dx<-(outer(xy[,1],xy[,1],"-"))
dy<-(outer(xy[,2],xy[,2],"-"))
alpha<-atan2(y=dy,x=dx); diag(alpha)<-1200
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 every point we find the hdeep 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 hdeep.

35 ⟨*compute hdepths* 35⟩≡

```
hdepth<-rep(0,n); dpi<-2*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 < pi) # hallo
  a.shift<-(a+pi) %% dpi
  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])# hallo
}
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
    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 compute the depth $k$ with $\#D_k \leq$ `points.in.bag` $< \#D_{k-1}$

36 ⟨*find* k 36⟩≡
```
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
if(nrow(d.k)>2){  # changed in cause of data set 1 of W. Meuleman
  k<-d.k[k.1+1,2]
} else {
  k<-d.k[k.1,2]
}
if(verbose){cat("counts of members of dk:"); print(hd.table)}
if(verbose){cat("end of computation of k, k=",k)}
```

The two dimensional median is the center of gravity of the points (not data points) with maximal h-depths.
We extract some data points with maximal depths and define `tp` as random linear combinations of them. Then we compute their h-depths.

37 ⟨*compute hdepths of test points to find center* 37⟩≡
```
center<-apply(xy[which(hdepth==max(hdepth)),,drop=FALSE],2,mean)
hull.center<-NULL
if(10<nrow(xy)&&length(hd.table)>2){
  n.p<-floor(c(32,16,8)[1+(n>50)+(n>200)]*precision)
  cands<-xy[rev(order(hdepth))[1:6],]
  cands<-cands[chull(cands[,1],cands[,2]),]; n.c<-nrow(cands)
  ⟨check points on a grid to find center 38⟩
  if(verbose){cat("hull.center",hull.center); print(table(tphdepth)) }
}
if(verbose) cat("center depth:",hdepth.of.points(rbind(center),n))
if(verbose){print("end of computation of center"); print(center)}
# cat("HALLO"); print(hdepth.of.points(cbind(6.75,4.85),n))
```

38 ⟨*check points on a grid to find center* 38⟩≡
```
  xyextr<-rbind(apply(cands,2,min),apply(cands,2,max))
  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]),]
```

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

39 ⟨*beta* 39⟩≡

```
#  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]),]
```

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`-

40 ⟨*method one: find hulls of $D_k$ and $D_{k-1}$* 40⟩≡

```
# 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

41 ⟨*method two: find hulls of $D_k$ and $D_{k-1}$* 41⟩≡

```
# 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)
# standardization of data set
xym<-apply(xy,2,mean); xysd<-apply(xy,2,sd)
xyxy<-cbind((xy[,1]-xym[1])/xysd[1],(xy[,2]-xym[2])/xysd[2])
kkk<-k
⟨find kkk-hull: pg 42⟩
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 42⟩
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.

42 ⟨*find kkk-hull:* `pg` 42⟩≡

```
⟨initialize loop of directions 44⟩
for(ia in seq(angles)[-1]){
  ⟨body of loop of directions 43⟩
}
⟨combination of lower and upper polygon 45⟩
```

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 `pg` and a lower polygon `pgl` that limits 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 cut points.

43 ⟨*body of loop of directions* 43⟩≡

```
# 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 46⟩
```

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.

44 ⟨*initialize loop of directions* 44⟩≡

```
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 47⟩
```

The combination of the is a little bit complicated because sometimes at the right and at left margin an additional cut 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.

45 ⟨*combination of lower and upper polygon* 45⟩≡
```
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")
```

46 ⟨*debug: plot within for loop* 46⟩≡
```
#########################################
#### 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)
}
```

47 ⟨*debug: plot ini* 47⟩≡
```
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)
}
```

On the way of finding the bag the function `expand.hull` computes not an exact solution but a numerical approximation. `k.1` indicates the polygon with h-depth $k-1$. `k.1+1` will usually points to h-depth $k$, to the inner polygon.
In computing $\lambda$ we follow Miller et al. (1999). They define $\lambda$ 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:*

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

$\lambda == 0$ happens if bag and inner contour is identical.

48 ⟨*find value of* `lambda` 48⟩≡

```
# lambda<-if(nrow(d.k)==1 ) 0.5 else  # version 09/2005
#                    (n/2-d.k[k.1+1,1])/(d.k[k.1,1]-d.k[k.1+1,1])
# data set 1 of W. Meuleman runs in difficulties because nrow(d.k)==2 and k.1==2
if(nrow(d.k)==1)                  lambda<-0.5
if(nrow(d.k)==k.1)                lambda<-0
if(nrow(d.k)!=1 & nrow(d.k)!=k.1) 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)
```

49 ⟨*error* 49⟩≡

⟨*data set 1 of Wouter Meuleman* 10⟩

```
bagplot(a[,2],a[,3],verbose=TRUE,debug.plots="all",dkmethod=1)
```

$\lambda == 0$ happens if bag and inner contour is identical. The bag is constructed by lambda * outer polygon + (1-lambda)* inner polygon

50 ⟨*find* `hull.bag` 50⟩≡

```
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
h1<-(1-lambda)*exp.dk+lambda*cut.on.pdk.1
# shrink outer polygon
h2<-(1-lambda)*cut.on.pdk+lambda*exp.dk.1
h<-rbind(h1,h2);
# some lines of h are NaN in example of Wouter Meuleman # print(h) # short correction
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") }
```

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

51 ⟨*find* `hull.loop` 51⟩≡

```
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 look for the loop ...

52 ⟨*find points outside of bag but inside loop* 52⟩≡

```
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")
```

and compute the hull of the loop points.

53 ⟨*find hull of loop* 53⟩≡

```
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")
```

Finally the plot has to be constructed. For this we write a new plot method.

54 ⟨*define* `plot.bagplot` 54⟩≡

```
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,...
){
 # 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 21⟩
 ⟨define function cut.z.pg 23⟩
 ⟨define function find.cut.z.pg 24⟩
 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 56⟩
 }
 ⟨construct bagplot as usual 55⟩
}
```

We need the following elements to be able to construct 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

55 ⟨*construct bagplot as usual* 55⟩≡
```
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"
    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"
```

56 ⟨*construct plot for one dimensional case and return* 56⟩≡

```
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.

57 ⟨*additional graphical comments if necessary* 57⟩≡

```
# 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")
```

# 7 Random data set

58 ⟨*define data* `xy` 58⟩≡

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

# 8 Definitionn of `bagplot` on start

59 ⟨*start* 59⟩≡

⟨*define* `bagplot` 16⟩

# 9 Extracting of function `bagplot`

60 ⟨*call* `tangleR` *to extract tangle function* `bagplot()` 60⟩≡

```
tangleR("hdeep.rev",expand.roots="define [[bagplot]]")
```