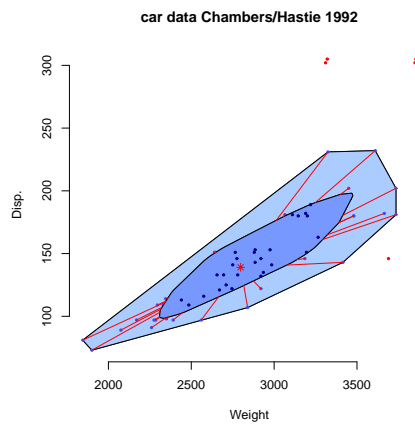


A rough R Impementation of the Bagplot

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

September 21, 2005



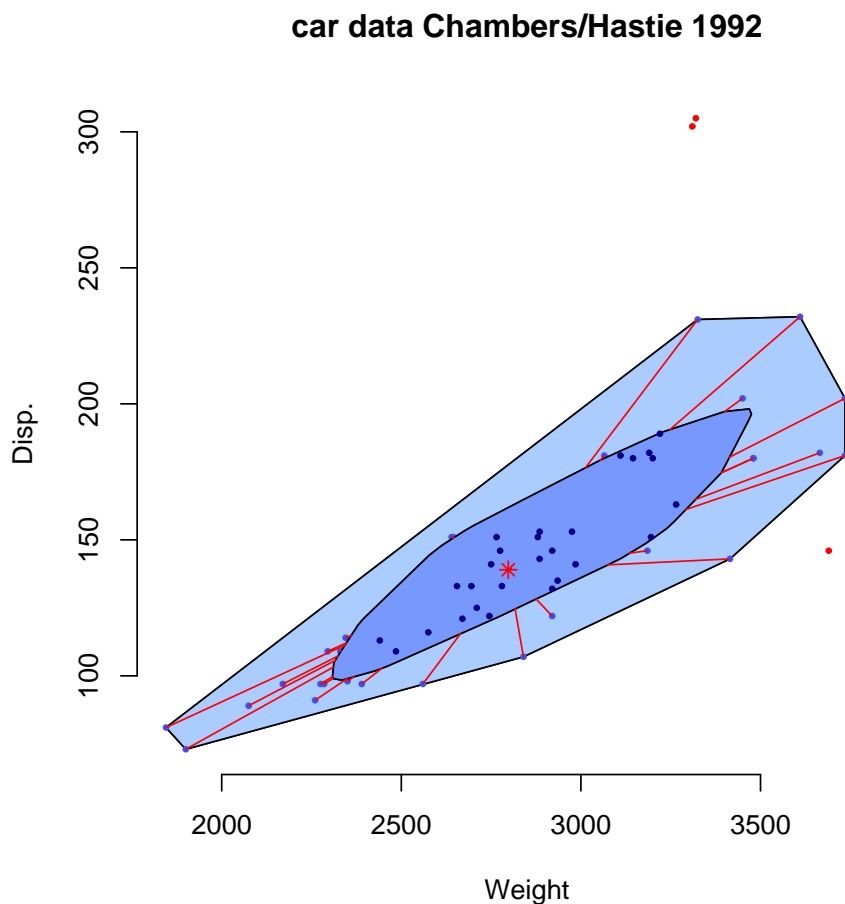
Contents

1	Examples	2
1.1	Example: car data (Chambers / Hastie 1992)	2
1.2	The normal case	3
1.3	Large data sets	4
1.4	Size of data set	5
1.5	Depth one data sets	6
1.6	Degenerated data sets	6
1.7	Data set from the mail of M. Maechler	7
1.8	Bagplot with additional graphical supplements	8
1.9	Debugging plots with additional elements	9
2	Arguments of bagplot	10
3	Links	10
4	The definition of bagplot	10
5	Random data set	23
6	Definitionn of bagplot on start	23
7	Extracting of function bagplot	23

1 Examples

1.1 Example: car data (Chambers / Hastie 1992)

```
1 <cardata 1> ≡  
<define bagplot 12>  
  library(rpart); cardata<-car.test.frame[,6:7]; par(mfrow=c(1,1))  
  bagplot(cardata,verbose=F,factor=3,show.baghull=T,dkmethod=2,  
  show.loophull=T,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 ...

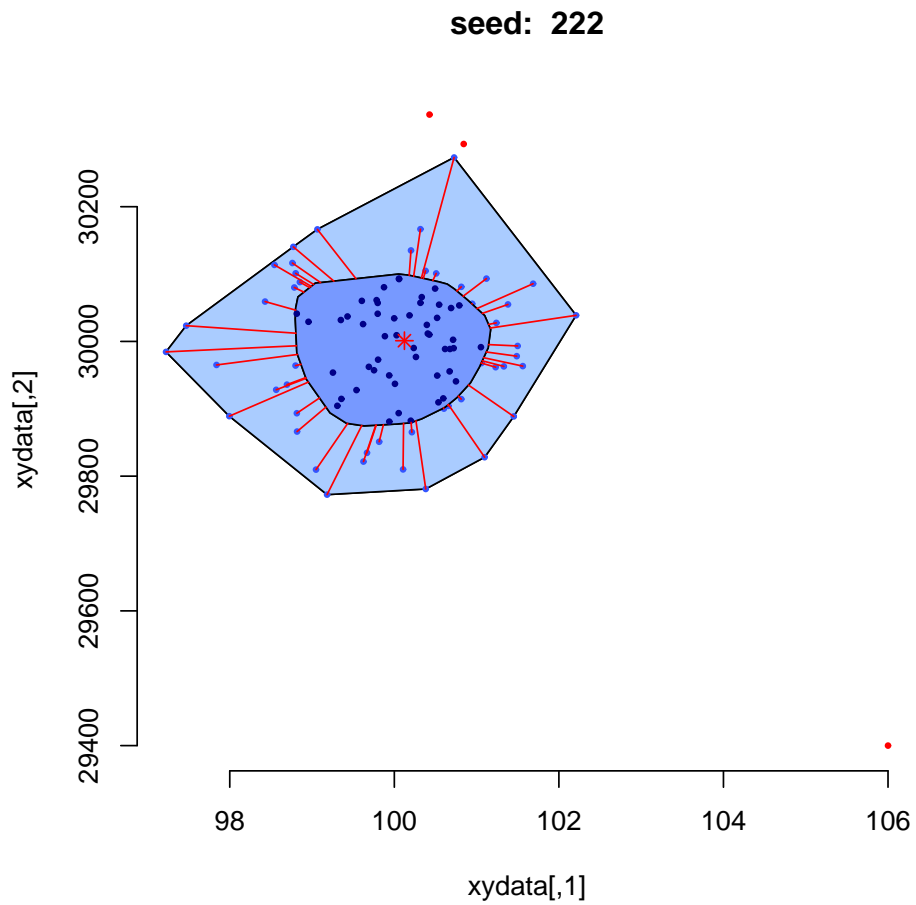
```
Splus CHAPTER bagplot.f  
Splus make  
Splus ...  
> dyn.open("S.so"); source("bagplot.s")  
> postscript("hello.ps"); bagplot(cardata[,1],cardata[,2]); dev.off()
```

1.2 The normal case

A bagplot of an `rnorm` sample plus one heavy outlier

2

```
<rnorm 2> ≡  
  111<-221  
<define data xy 50>  
  datan<-rbind(data,c(106,294)); par(mfrow=c(1,1))  
  datan[,2]<-datan[,2]*100  
  bagplot(datan,factor=3,create.plot=T,approx.limit=300,  
          show.outlier=T,show.looppoints=T,show.bagpoints=T,  
          show.whiskers=T,show.loophull=T,show.baghull=T,verbose=F)  
  title(paste("seed: ",111))
```

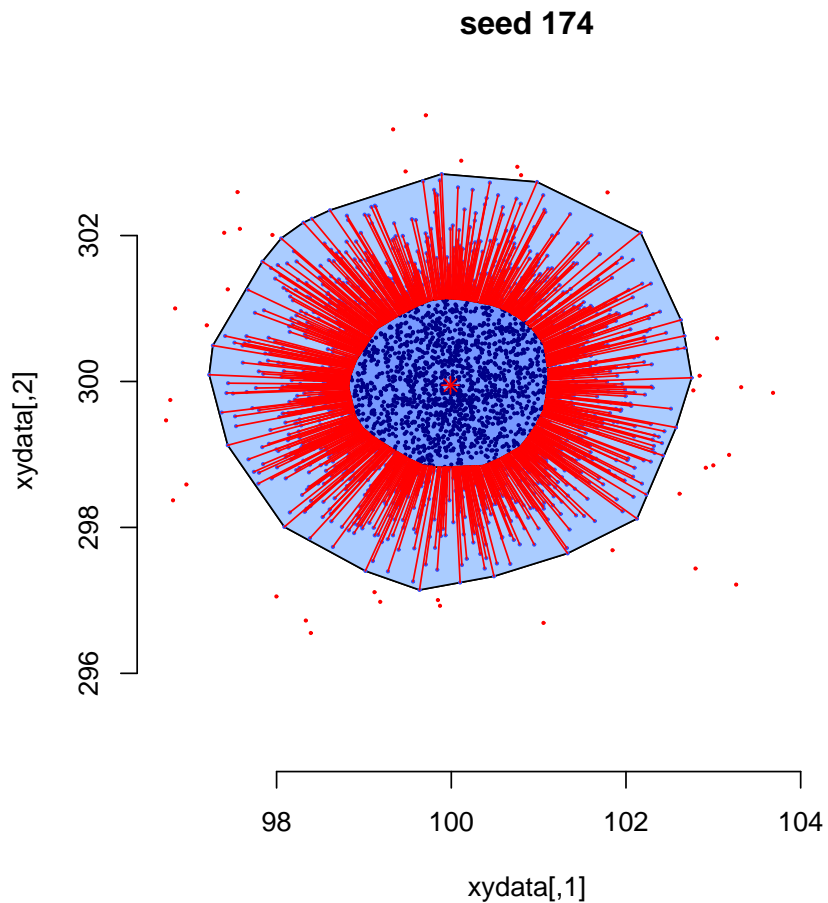


1.3 Large data sets

What about large data sets?

3

```
<large 3> ≡  
l11<-173  
if(!exists("l11")) l11<-75  
set.seed(l11<-l11+1); print(l11)  
n<-3000; datan<-cbind(rnorm(n)+100, rnorm(n)+300)  
print(l11)  
datan<-rbind(datan, c(105, 295))  
par(mfrow=c(1,1)) #; par(mfrow=2:3)  
bagplot(datan, factor=2.5, create.plot=T, approx.limit=1000, cex=0.2,  
        show.outlier=T, show.looppoints=T, show.bagpoints=T, dkmeth=2,  
        show.loophull=T, show.baghull=T, verbose=F, debug.plots="no")  
title(paste("seed", l11))
```



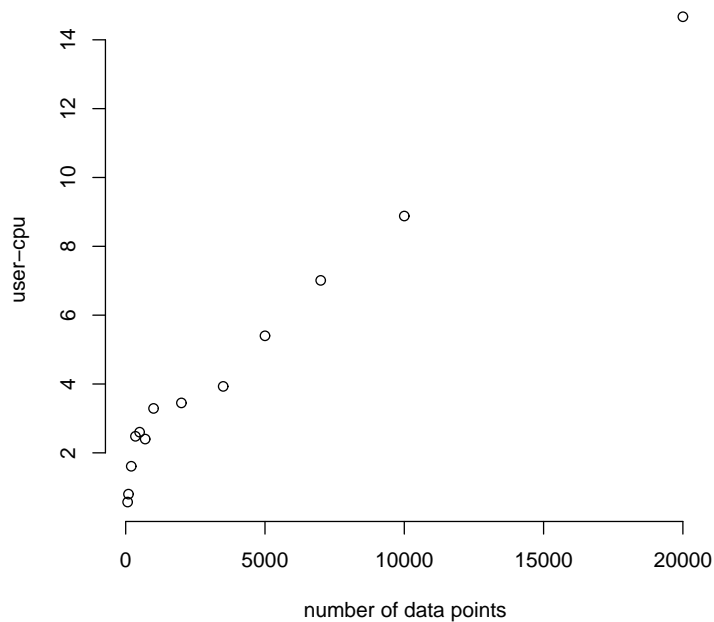
1.4 Size of data set

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)+ ≡
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=F,approx.limit=300,
      show.outlier=T,show.looppoints=T,show.bagpoints=T,
      show.whiskers=T,show.loophull=T,show.baghull=T,verbose=F)
  )[1]
}
plot(nn,result,bty="n",ylab="user-cpu",xlab="number of data points")
names(result)<-nn; result
```

Tue Sep 20 09:23:31 2005

70	100	200	350	500	700	1000	2000	3500	5000	7000	10000	20000
0.74	0.98	2.10	2.70	2.67	2.36	3.16	3.28	3.95	5.49	7.33	8.87	14.86



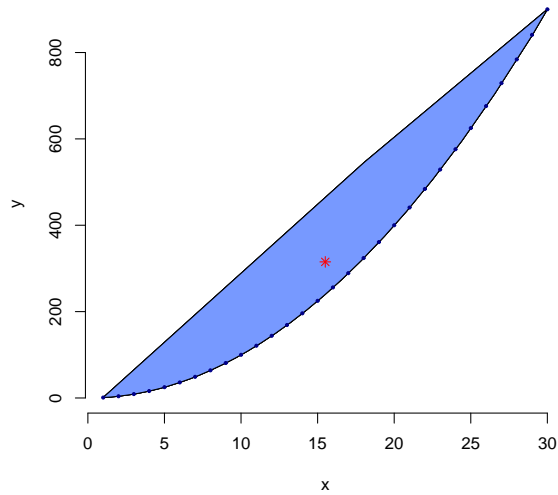
1.5 Depth one data sets

What happens if all points are of depth 1?

5

`<quadratic 5> ≡`

```
bagplot(x=1:30,y=(1:30)^2,verbose=F,dkmethod=2)
```



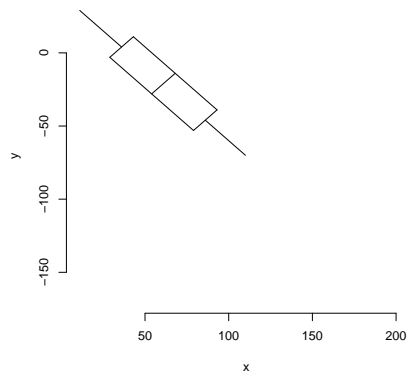
1.6 Degenerated data sets

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=F)
```



Here is a second one dim data set.

7

`<one dim test 7> ≡`

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

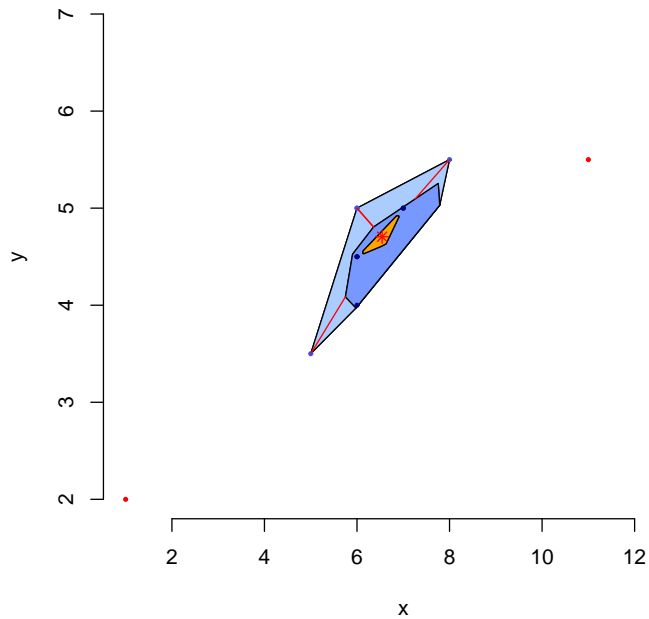
1.7 Data set from the mail of M. Maechler

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

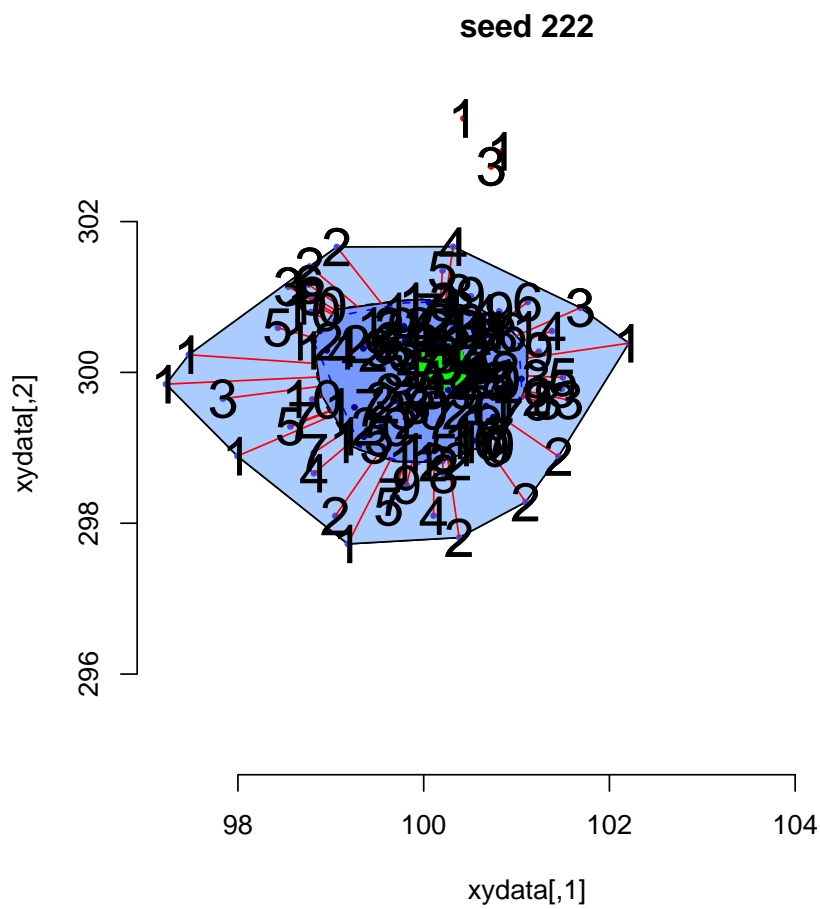
```
<mm 8> ≡  
# hallo  
<define bagplot 12>  
x0<-c(1,5, 6,6, 6, 6,6,7,7,8, 11, 13) # x0 <- c(x0, 8)  
y0<-c(2,3.5,4,4.5,4.5,5,5,5,5,5.5,5.5, 7) # y0 <- c(y0, 7)  
par(mfrow=c(1,1))  
bagplot(x0,y0,show.baghull=T,show.loophull=T,create.plot=T,  
        show.whiskers=T,factor=3,debug.plots="notall",  
        dkmethod=2,verbose=F,precision=1)$center  
#abline(h=4.85,v=6.75)
```



1.8 Bagplot with additional graphical supplements

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

```
9 <verbose>test 9) ≡  
  l11<-221  
<define data xy 50)  
  datan<-rbind(data,c(105,295))  
  bagplot(datan,factor=2.5,create.plot=T,approx.limit=300,  
    show.outlier=T,show.looppoints=T,show.bagpoints=T,dkmethod=2,  
    show.whiskers=T,show.loophull=T,show.baghull=T,verbose=T)  
  title(paste("seed",l11))
```



1.9 Debugging plots with additional elements

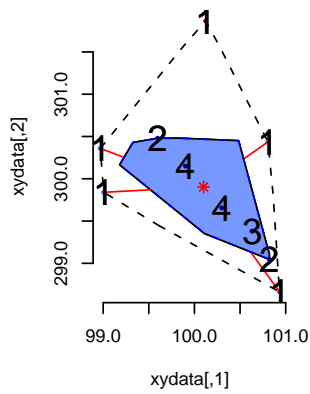
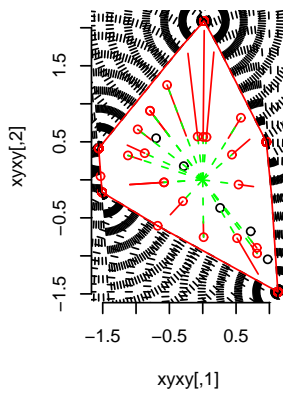
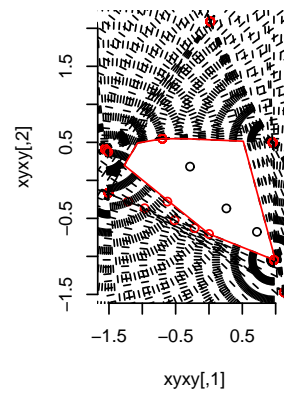
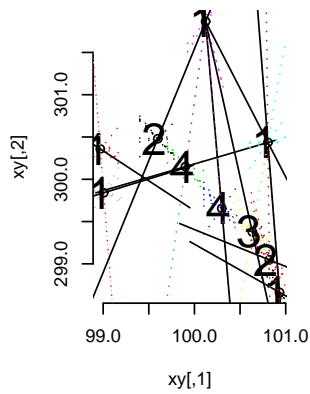
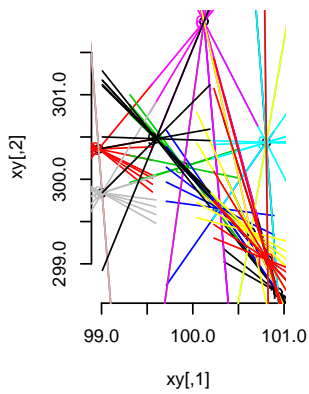
Here is an example of plots generated with option `debug.plots="all"`.

10

```

(debugplot 10) ≡
(define data xy 50)
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=T,approx.limit=300,
        show.outlier=T,show.looppoints=T,show.bagpoints=T,
        show.whiskers=T,show.loophull=F,show.baghull=T,dkmethod=2,
        debug.plots="all",verbose=T)

```



2 Arguments of bagplot

11 $\langle args\ 11 \rangle \equiv$
args(bagplot)

Wed Sep 21 15:48:55 2005

```
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, ..., dkmethode = 2, precision = 1,
 verbose = FALSE, debug.plots = "")
NULL
```

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

3 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/
```

4 The definition of bagplot

12 $\langle define\ bagplot\ 12 \rangle \equiv$
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,..., # to define further parameters of plot
 dkmethode=2, # in 1:2; there are two methods for approximating the bag
 precision=1, # controls precision of computation
 verbose=FALSE,debug.plots="" # tools for debugging
) {
 $\langle body\ of\ bagplot\ 13 \rangle$
}

```

13  <body of bagplot 13> ≡
      #pwolf 050921
      <init 15>
      <check and handle linear case 28>
      <compute angles between points 29>
      <compute hdepths 30>
      <find k 31>
      <compute hdepths of test points to find center 32>
      if (dkmethod==1) {
        <method one: find hulls of  $D_k$  and  $D_{k-1}$  35>
      } else {
        <method two: find hulls of  $D_k$  and  $D_{k-1}$  36>
      }
      <find value of lambda 43>
      <find hull.bag 44>
      <find hull.loop 45>
      <find points outside of bag but inside loop 46>
      <find hull of loop 47>
      <create plot 48>
      <output result 14>

```

Output of `bagplot` is a list of essential components of the computation. To identify singular points, use `identify()`.

```

14  <output result 14> ≡
      assign(".Random.seed", save.seed, env=.GlobalEnv)
      return(invisible(list(
        center=center, bag=hull.bag, loop=hull.loop,
        outlier=if(length(pkt.outlier)>0) pkt.outlier else NULL,
        hdepths=hdepth
      )))

```

Points with identical coordinates may result in numerical problem. Therefore, some noise is added to the data.

```

15  <init 15> ≡
      # define some functions
      <define function win 16>
      <define function out.of.polygon 17>
      <define function cut.z.pg 18>
      <define function find.cut.z.pg 19>
      <define function hdepth.of.points 20>
      <define function expand.hull 21>
      <define function cut.p.sl.p.sl 26>
      <define function pos.to.pg 27>
      # 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

```

16 <define function win 16> ≡
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).

```

17 <define function out.of.polygon 17> ≡
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 plx,ply,p2x,p2y and lines that contains zx,zy and origin.

```

18 <define function cut.z.pg 18> ≡
cut.z.pg<-function(zx,zy,plx,ply,p2x,p2y){
  a2<-(p2y-ply)/(p2x-plx); a1<-zy/zx
  sx<-(ply-a2*plx)/(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(verbose) cat("special")
    # points on line defined by line segment
    h<-0==(a1-a2) & sign(zx)==sign(plx)
    sx<-ifelse(h,plx,sx); sy<-ifelse(h,ply,sy)
    h<-0==(a1-a2) & sign(zx)!=sign(plx)
    sx<-ifelse(h,p2x,sx); sy<-ifelse(h,p2y,sy)
    # line segment vertical
    # & center NOT ON line segment
    h<-plx==p2x & zx!=plx & plx!=0
    sx<-ifelse(h,plx,sx); sy<-ifelse(h,zy*plx/zx,sy)
    # & center ON line segment
    h<-plx==p2x & zx!=plx & plx==0
    sx<-ifelse(h,plx,sx); sy<-ifelse(h,0,sy)
    # & center ON line segment & point on line
    h<-plx==p2x & zx==plx & plx==0 & sign(zy)==sign(ply)
    sx<-ifelse(h,plx,sx); sy<-ifelse(h,ply,sy)
    h<-plx==p2x & zx==plx & plx==0 & sign(zy)!=sign(ply)
    sx<-ifelse(h,plx,sx); sy<-ifelse(h,p2y,sy)
    # points identical to end points of line segment
    h<-zx==plx & zy==ply; sx<-ifelse(h,plx,sx); sy<-ifelse(h,ply,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,plx,ply,p2x,p2y,sxy))}
  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.

```

19 (define function find.cut.z.pg 19) ≡
  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(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.

```

20 (define function hdepth.of.points 20) ≡
  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 expansion.

```
21 <define function expand.hull 21> ≡
    expand.hull<-function(pg,k){
      <find end points of line segments: mean → pg → pg0 22>
      <search for points with critical hdepth 23>
      <find additional points between the line segments 24>
      <compute hull pg.new 25>
    }
```

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.

```
22 <find end points of line segments: mean → pg → pg0 22> ≡
    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.

```
23 <search for points with critical hdepth 23> ≡
    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.

```
24 <find additional points between the line segments 24> ≡
    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 $\text{hull}(d_k)$. `pg.new` is the output of `expand.hull`.

```
25 <compute hull pg.new 25> ≡
    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:

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

```
26 <define function cut.p.sl.p.sl 26> ≡
    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".

```
27 <define function pos.to.pg 27> ≡
    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.

```
28 <check and handle linear case 28> ≡
    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")
      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(seg[1],seg[2],seg[3],seg[4],)
return(apply(matrix(seg[2,],2,2),1,mean))
}
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.

```

29 <compute angles between points 29> ≡
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.

```

30 <compute hdepths 30> ≡
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[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)
    }
  }
}

```

31 We compute the depth k with $\#D_k \leq \text{points.in.bag} < \#D_{k-1}$

```

<find k 31> ≡
hd.table<-table(sort(hdepth))
d.k<-cbind(dk=rev(cumsum(rev(hd.table))),
          k =as.numeric(names(hd.table)))
k.l<-sum(points.in.bag<d.k[,1])
if(nrow(d.k)>1){
  k<-d.k[k.l+1,2]
} else {
  k<-d.k[k.l,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.

```

32 <compute hdepths of test points to find center 32> ≡
center<-apply(xy[which(hdepth==max(hdepth))],,drop=FALSE),2,mean)
if(10<nrow(xy)&&length(hd.table)>2){
  n.p<-floor(c(32,16,8)[1+(n>50)+(n>200)]*precision)
  cand<-xy[rev(order(hdepth))[1:6],]
  cand<-cand[chull(cand[,1],cand[,2]),]; n.c<-nrow(cand)
  <check points on a grid to find center 33>
  if(verbose){cat("center.region",center.region); 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))

```

33 <check points on a grid to find center 33> ≡

```

xyextr<-rbind(apply(cand,2,min),apply(cand,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)
center.region<-tp[which(tphdepth>=(max(tphdepth))),,drop=FALSE]
center<-apply(center.region,2,mean)
cand<-center.region[chull(center.region[,1],center.region[,2]),,drop=F]
xyextr<-rbind(apply(cand,2,min),apply(cand,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)
center.region<-tp[which(tphdepth>=max(tphdepth)),,drop=FALSE]
center<-apply(center.region,2,mean)
center.region<-center.region[chull(center.region[,1],center.region[,2]),]

```

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

```
34 <beta 34> ≡
# 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)
center.region<-tp[which(tphdepth==max(tphdepth)),,drop=FALSE]
center<-apply(center.region,2,mean)
center.region<-center.region[chull(center.region[,1],center.region[,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`

```
35 <method one: find hulls of  $D_k$  and  $D_{k-1}$  35> ≡
# 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 appropriate directions
3. computation of D_k polygon and restandardization
4. computation of D_{k-1} polygon and restandardization

```
36 <method two: find hulls of  $D_k$  and  $D_{k-1}$  36> ≡
# define direction for hdepth search
num<-floor(c(351,171,85)[c(n>200,n<200,n<50)][1]*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 37>
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 37>
```

```
exp.dk.l<-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.

```
37 <find kkk-hull: pg 37> ≡
<initialize loop of directions 39>
for(ia in seq(angles)[-1]){
  <body of loop of directions 38>
}
<combination of lower and upper polygon 40>
```

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.

```
38 <body of loop of directions 38> ≡
# 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]))
  pgl.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])
  pgl.interl<-pgl[,2]-angtan*pgl[,1]; pnew.interl<-pnewl[2]-angtan*pnewl[1]
  pgl.nol<-sum(pgl.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 41>
```

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.

```
39 <initialize loop of directions 39> ≡
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 42>
```

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.

```
40 <combination of lower and upper polygon 40> ≡
pg<-pg[-nrow(pg),][-1,,drop=F]; 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[1]=="higher"&indu[1]=="lower"){
  lnuml<-which(rev(indl=="lower"))[1]; xyl<-pgl[1:2,lnuml,] #
  lnumu<-which(indu=="higher")[1]-1; xyu<-pg[1:2,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")
```

```
41 <debug: plot within for loop 41> ≡
#####
#### 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)
}
```

```
42 <debug: plot ini 42> ≡
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 $k-1$. $k+1$ will usually points to h-depth k , 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:

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

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

```
43 <find value of lambda 43> ≡
  lambda<-if(nrow(d.k)==1) 0.5 else
                                (n/2-d.k[k.1+1,1])/(d.k[k.1,1]-d.k[k.1+1,1])
  if(verbose) cat("lambda",lambda)
```

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

```
44 <find hull.bag 44> ≡
  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 polygon
  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); 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`.

```
45 <find hull.loop 45> ≡
  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 ...

```
46 <find points outside of bag but inside loop 46> ≡
  if(!very.large.data.set){
    pkt.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)
        pkt.bag <-rbind(pkt.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)
    pkt.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)
    pkt.outlier<-pkt.not.bag[extr,,drop=FALSE]
    if(0==length(pkt.outlier)) pkt.outlier<-NULL
    pkt.outer<-pkt.not.bag[!extr,,drop=FALSE]
  }else{
    pkt.outer<-pkt.outlier<-NULL
  }
  if(verbose) cat("points of bag, outer points and outlier identified")
```

and compute the hull of the loop points.

```
47 <find hull of loop 47> ≡  
  hull.loop<-rbind(pkt.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 is constructed.

```
48 <create plot 48> ≡  
  if(create.plot){  
    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="#aaccff")  
    }  
    if(show.looppoints && length(pkt.outer)>0){ # points in loop  
      points(pkt.outer[,1],pkt.outer[,2],col="#3355ff",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="#7799ff")  
    }  
    if(show.bagpoints && length(pkt.bag)>0){ # points in bag  
      points(pkt.bag[,1],pkt.bag[,2],col="#000088",pch=pch,cex=cex)  
    }  
    # whiskers  
    if(show.whiskers && length(pkt.outer)>0){  
      debug.plots<-"not"  
      pkt.cut<-find.cut.z.pg(pkt.outer,hull.bag,center=center)  
      segments(pkt.outer[,1],pkt.outer[,2],pkt.cut[,1],pkt.cut[,2],col="red")  
    }  
    # outlier: -----  
    if(show.outlier && length(pkt.outlier)>0){ # points in loop  
      points(pkt.outlier[,1],pkt.outlier[,2],col="red",pch=pch,cex=cex)  
    }  
    # center:  
    if(exists("center.region")&&length(center.region)>2){  
      h<-rbind(center.region,center.region[1,]); lines(h[,1],h[,2],lty=1)  
      polygon(center.region[,1],center.region[,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")  
  }
```

In case of problems some additional plottings may be helpful.

```
49 <additional graphical comments if necessary 49> ≡  
  # points(exp.dk[,1],exp.dk[,2],type="b",col="red")  
  # points(exp.dk.1[,1],exp.dk.1[,2],type="b",col="green")  
  # points(exp.dk.1[1,1],exp.dk.1[1,2],type="b",col="blue")
```

5 Random data set

```
50 <define data xy 50> ≡  
  if(!exists("l11")) l11<-75  
  #l11<-75 # 267 81 115  
  set.seed(l11<-l11+1); print(l11)  
  #data<-matrix(sample(1:10000,size=2000),1000,2)  
  #data<-matrix(sample(1:10000,size=1000),500,2)  
  #data<-matrix(sample(1:10000,size=300),50,2)  
  n<-100;data<-cbind(rnorm(n)+100,rnorm(n)+300)  
  par(mfrow=c(1,1))
```

6 Definitionn of bagplot on start

```
51 <start 51> ≡  
  <define bagplot 12>
```

7 Extracting of function bagplot

```
52 <call tangleR to extract tangle function bagplot() 52> ≡  
  tangleR("hdeep.rev",expand.roots="define [[bagplot]]")
```