**Graphical functions for R**
**http://r-project.org**

# User functions in R

Dipl. Biol. Andreas Plank

Version: March 29, 2010

http://www.chironomidaeproject.com



## Contents

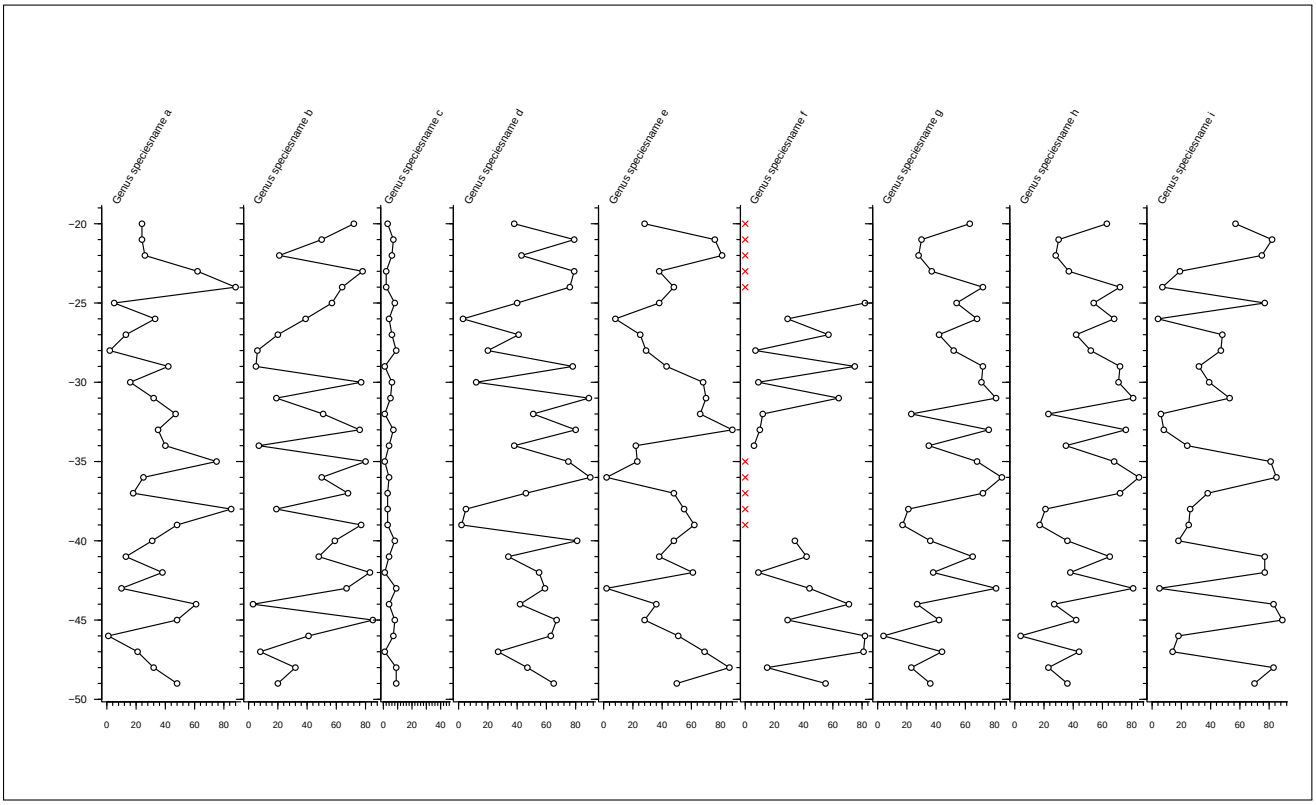


## User functions

# 1 Plotting depth profiles

Default graph of depth profiles to plot depth profiles with many column data sets (up to 50).

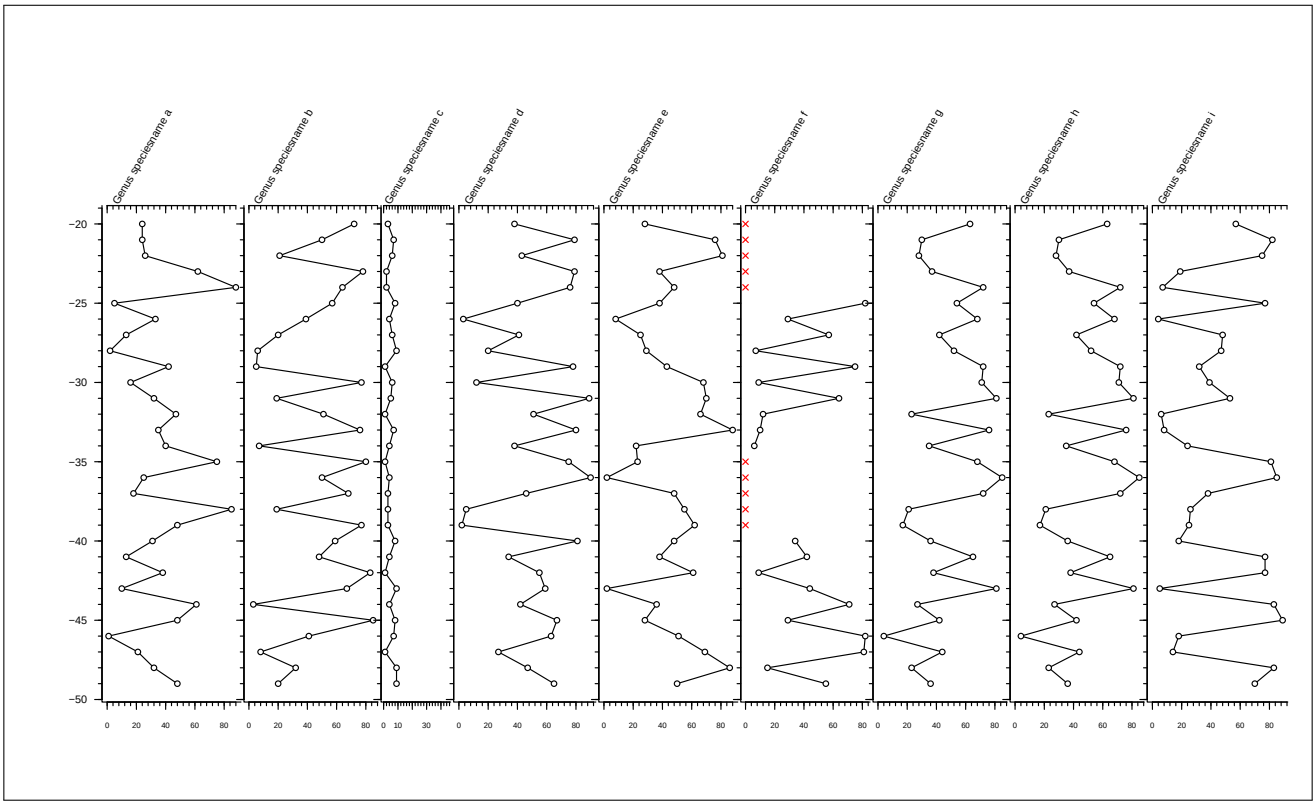
```
# create random dataset
test <- data.frame( #
  "depth"= depth <- 0:(-29)-20,
  a <- sample(90, 30, replace = TRUE),
  b <- sample(90, 30, replace = TRUE),
  c <- sample( 9, 30, replace = TRUE),
  d <- sample(90, 30, replace = TRUE),
  e <- sample(90, 30, replace = TRUE),
    # data with NA
  f <- c(
    rep(NA, 5), # 5 x NA
    sample(90, 10, replace = TRUE),
    rep(NA, 5), # 5 x NA
    sample(90, 10, replace = TRUE)),
  g <- sample(90, 30, replace = TRUE),
  h <- g,
  i <- sample(90, 30, replace = TRUE)
)
# add column names
(colnames(test)[2:ncol(test) ] <- paste("Genus speciesname",letters[1:(ncol(test)-1)]))

# plot default
par(las=1) # labels on axis
  plot.depth(test)
```




Example 2: axis top

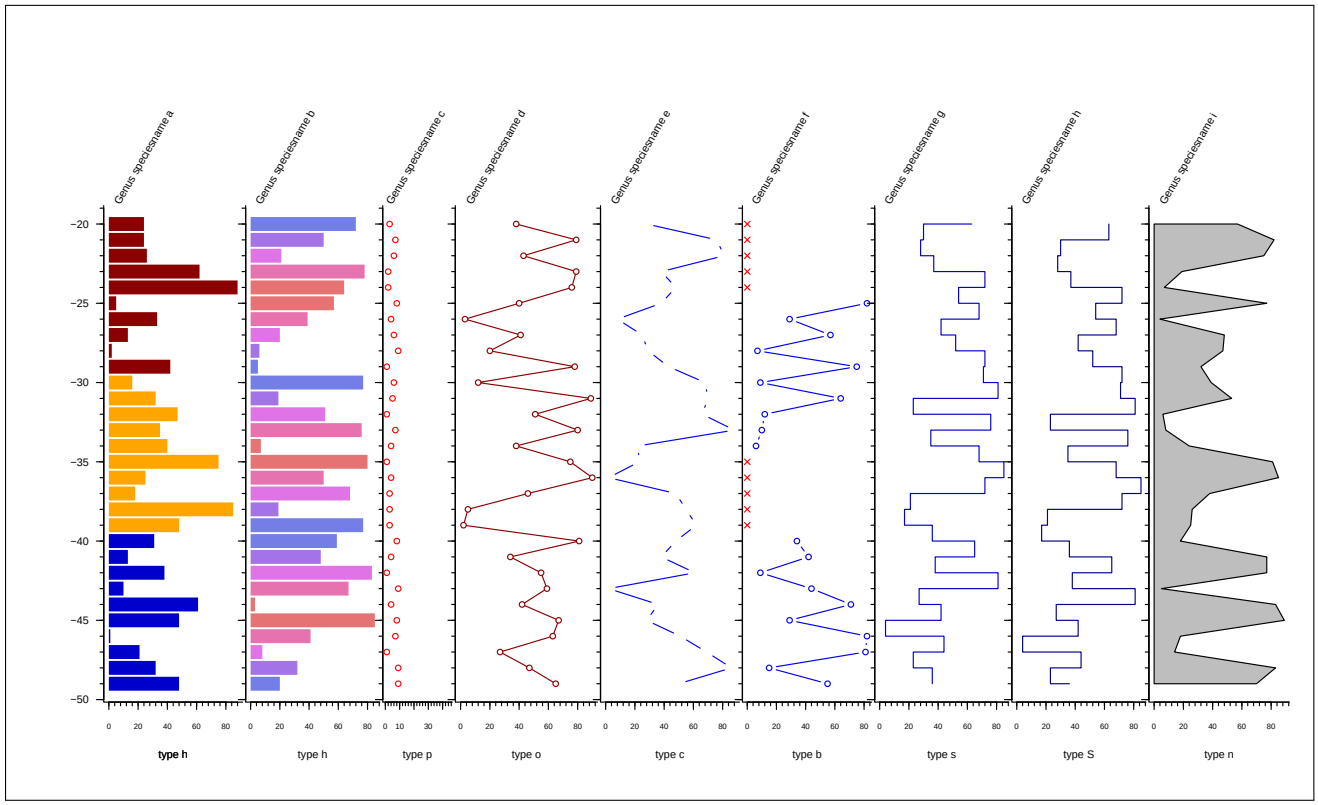
```
par(las=1) # labels on axis
```

```
  plot.depth(test, axis.top=list(c(T,F)))
```



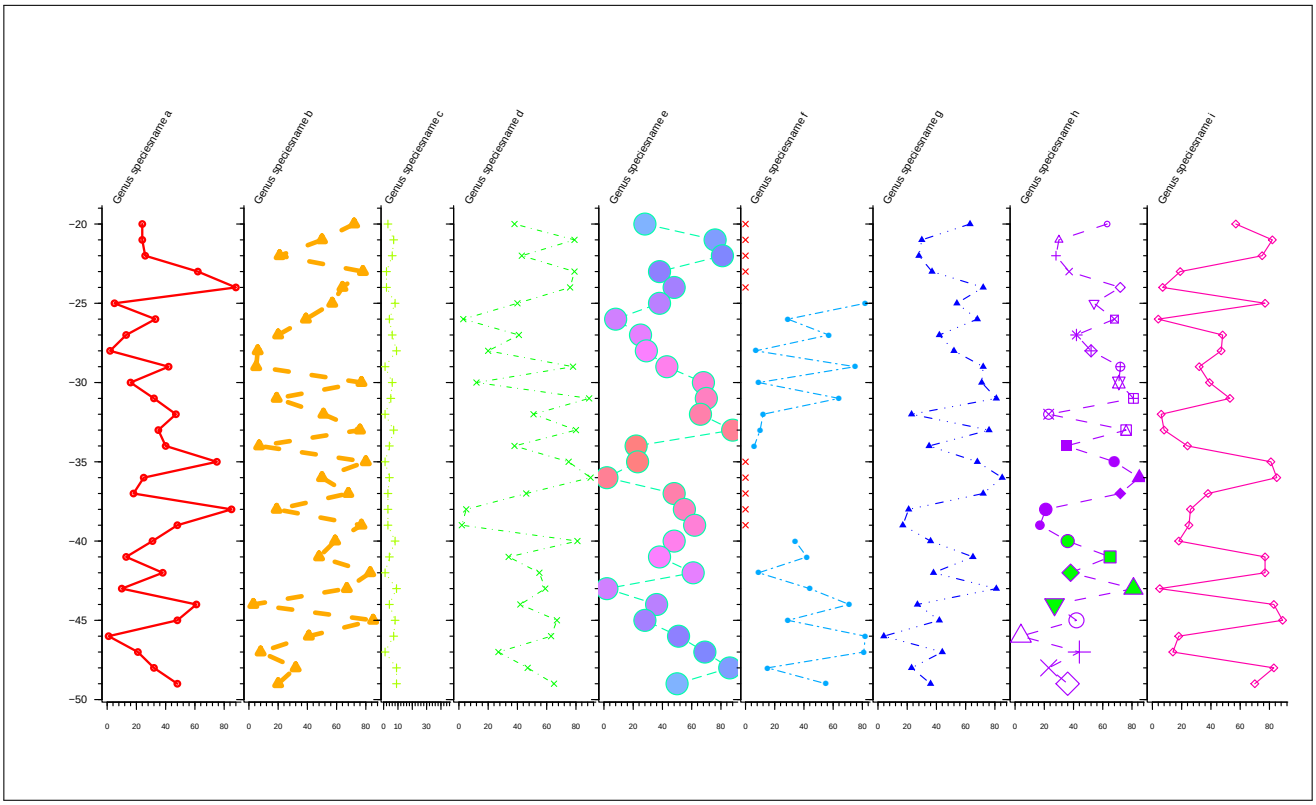

Example 3: colors and types

```
(color <- rep(c("darkred","orange","blue3"), each = 10))
bluetored <- rainbow(5, s=0.5, v=0.9, start=0.65, end=1)
color.coldwarm <- c(bluetored, bluetored[5:1], bluetored, bluetored[5:1], bluetored, bluetored[5:1])
par(las=1) # labels on axis
  plot.depth(test,
    plot.type = c("h","h","p","o","c","b", "s", "S", "n") -> type,
    polygon = c(rep(FALSE, 8), TRUE),
    xlabel = paste("type ",type, sep = ""), # Info
    l.width = c(12,12,1,1,1,1,1,1,1), # line widths
    lp.color = list(color, # plot 1
      color.coldwarm, # plot 2; 30 colors
      "red", # plot 3
      "darkred", # plot 4
      "blue1", # plot 5
      "blue2", # plot 6
      "blue3", # plot 7
      "blue4", # plot 8
      "white" # plot 9
    ) # End list color
  ) # End plot.depth()
```



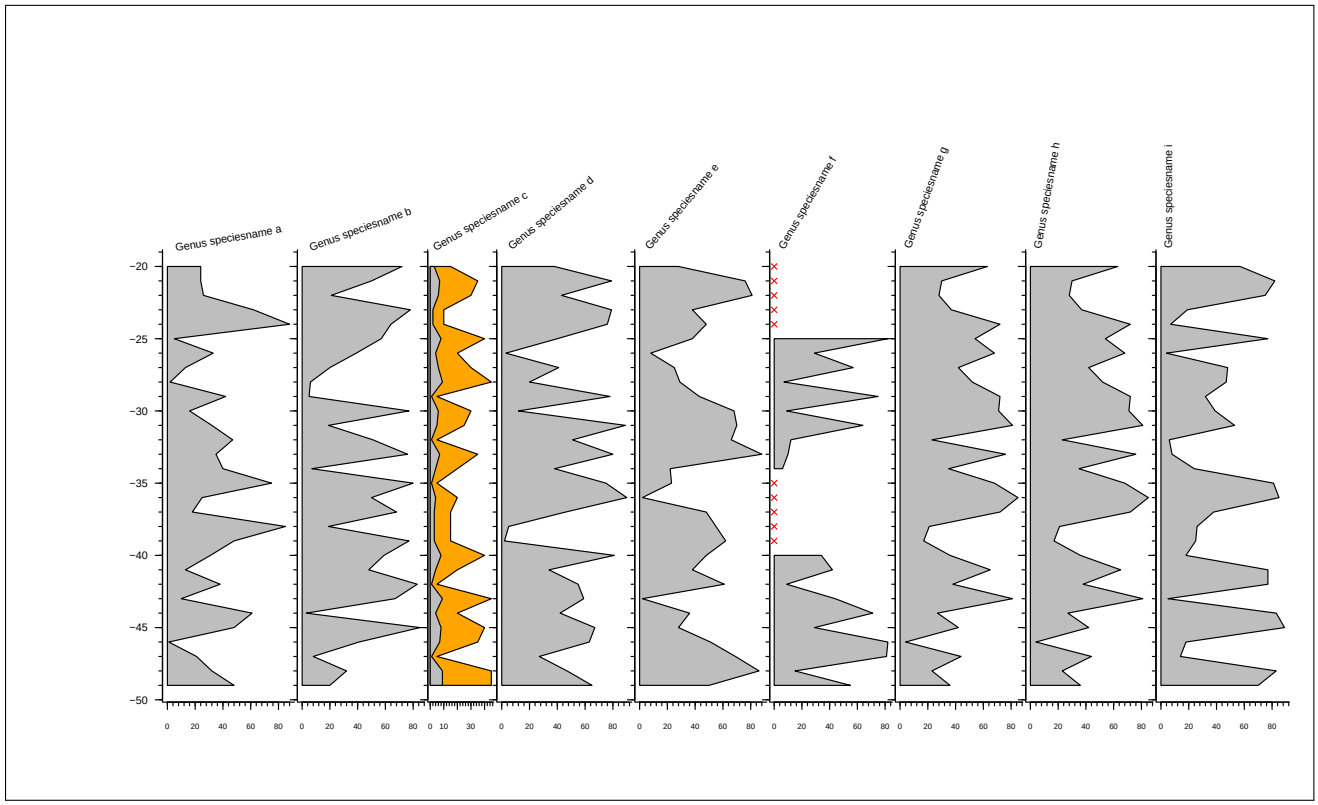

Example 4: lines+points

```
par(las=1) # labels on axis
bluetored <- rainbow(15, s=0.5, start=0.6, end=1)
color.points <- c(bluetored, bluetored[15:1])
plot.depth(test,
  plot.type = "o",
  # points
  p.type = list(1,2,3,4,21,16,17,c(1:25,1:5),5), # as for pch
  p.bgcolor = list("white", "white", "white", "white", # 1, 2, 3, 4th plot
    color.points, # at plot 5
    "white","white", "green", "white" # 6 7 8 9
  ),
  p.size = list(1, 1, 1, 1, 4, 1, 1,  seq(1, 3, length.out=30), 1 ),
  l.width = list(2, 4, 1, 0.5, 1, 1, 1, 1, 1 ),
  # lines: character, numeric, manuell
  l.type = list("solid",2,3,4,5,6,"64141414","88","solid"),
  lp.color = rainbow(9) # line/point-fg color
)
```




Example 5: scaled rare data

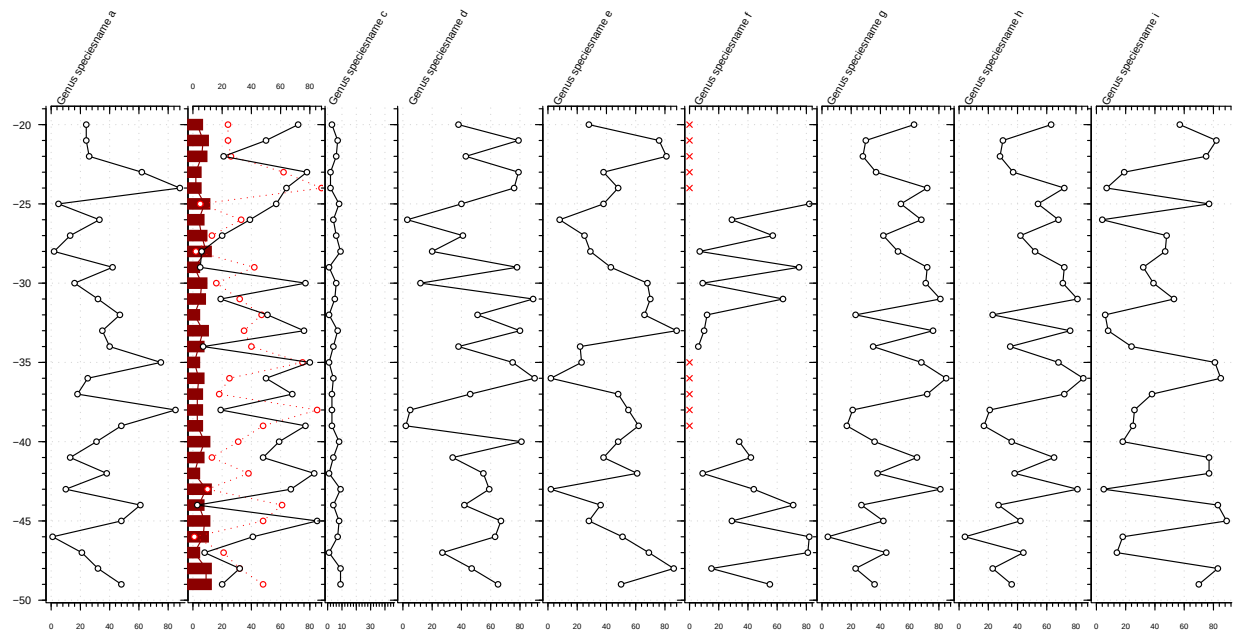
```
plot.depth(test,
  plot.type = "n",
  mar.outer = c(1,10,4,1), # more space at the left
  mar.top = 12, # more palce on top
  polygon = T,
  rotation = c(1:9)*10, # 10 20 30 ... 80
  min.scale.level = 0.12, # 12%-level of maximum
  min.scaling = c(F,F,5,F,F,F,F,F,F), # T - TRUE, F - FALSE
  color.minscale = "orange"   # Farbe
)
```




Example 6: additinal plots

```
plot.depth(test,
  plot.before=expression(c(par(xpd=F),grid())), # add grid, but not in outer region
  plot.after=list(
    NULL, # 1. plot
    expression( # at the 2. plot:
      # data from 4th plot
      segments(0, test[,1], test[,4], test[,1], lend=2, lwd=10, col="darkred"),
      # daten fron 4th Graph
      lines(x=test[,4],y=test[,1], col="darkred", lty="solid", pch=16, type="o"),
      # daten fron 2nd Graph
      lines(x=test[,2],y=test[,1], col="red", pch=21 , lty="dotted", type="o", bg="white")
    ), # end expression(..)
    NULL, NULL, NULL, NULL, NULL, NULL, NULL
    # 3. 4. 5. 6. 7. 8. 9. plot
  ), # end list(..) in Option 'plot.after'
  axis.top=list( # top lables
    c(TRUE, FALSE), # 1. ticks-yes, numbers-no
    c(T, T), # 2.
    c(F, T), # 3.
    c(F, F), # 4.
    c(T, F), # 5.
    c(T, F), # 6.
    c(T, F), # 7.
    c(T, F), # 8.
    c(T, F) # 9.
  ),
  colnames=c( # column names
    T, F, T, T, T, T, T, T, T
```

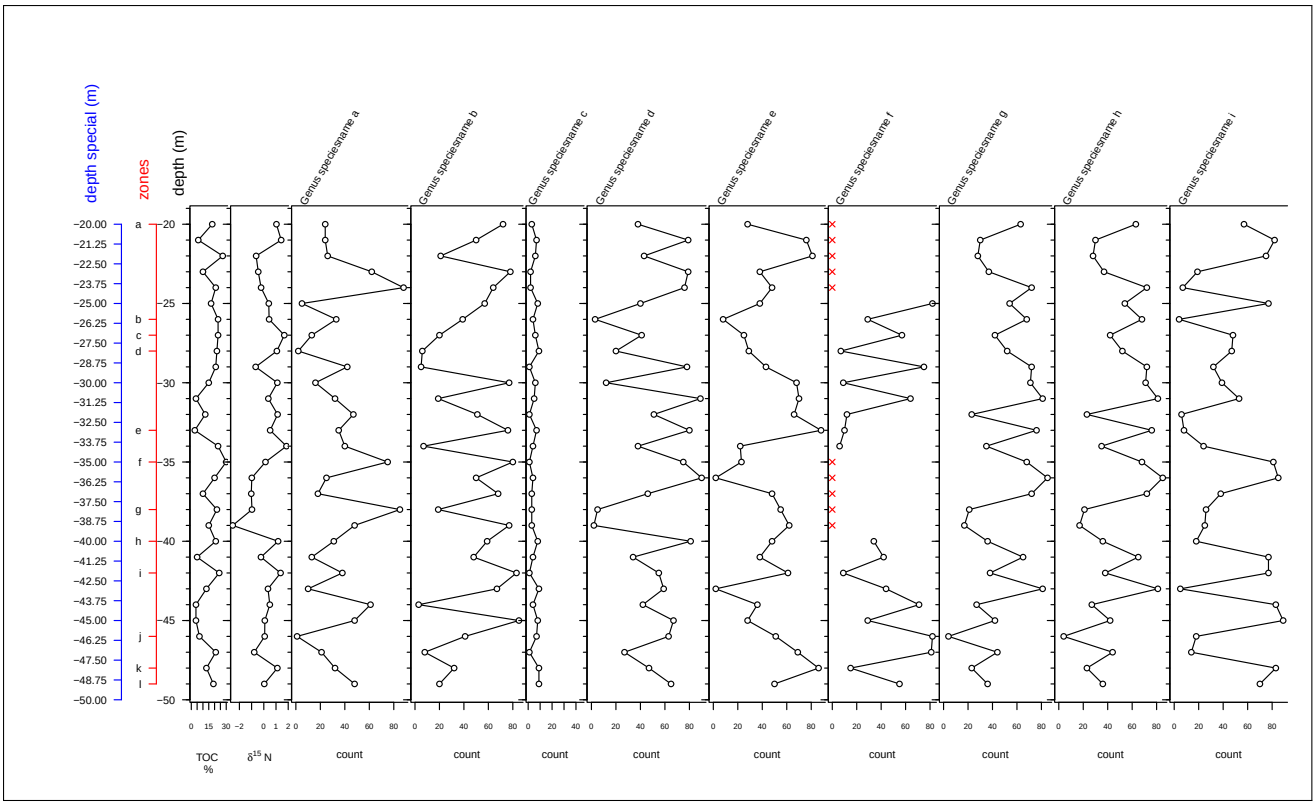
```
  )
) # End plot.depth(..)
```




Example 7: additional axes

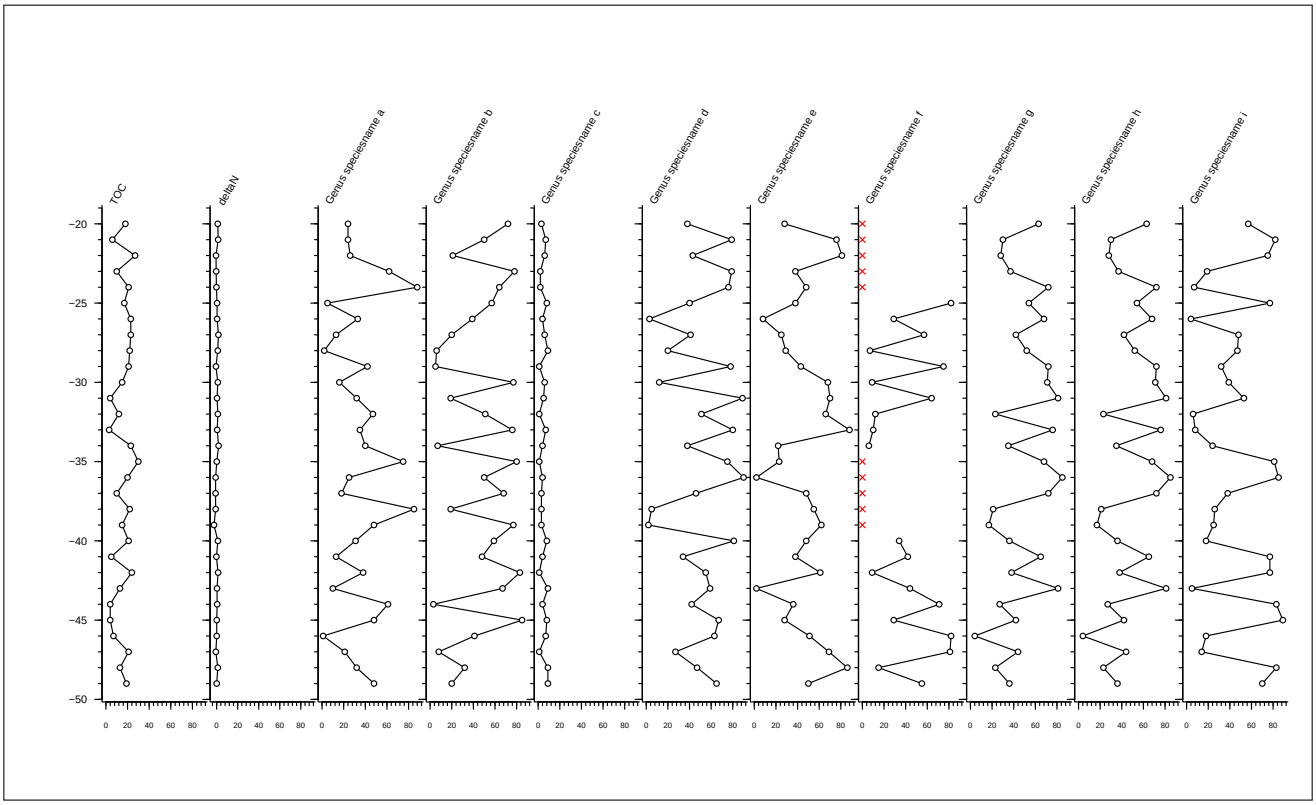
```
testabiotic <- cbind(
  test[,1], # depth data from 1st column
  "TOC"= toc <- sample(30, replace=TRUE),
  "deltaN"=deltaN <- rnorm(30),
  test[,-1] # rest of data
)
# text for axis
text <- letters[1:12]
# axis positions
where.y <- c(-20, -26, -27, -28, -33, -35, -38, -40, -42,-46, -48, -49)
par(las=1)
plot.depth(testabiotic,
  mar.outer = c(1, 12, 4, 1), # c(bottom , left , top, right)
  nx.minor.ticks=0,
  bty ="c", # box like a 'C'
  plot.before =list(
    expression( # at 1. plot
      axis(side=2, labels=text, at=where.y, pos=-30, col="red"),
      # y-axis with text + at
      axis(side=2, pos=-60, col="blue", yaxp=c(-20, -50, 24))
      # special axis intervals
    ),
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
    # 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. plot
  ),
```

```
  xaxes.equal = c(F,F,rep(T, 9)), # not equal axes on all axes
  colnames = c(F,F,rep(T, 9)),
  xlabel = list(
    "", # 1. plot
    expression(delta^15 ~ N), # 2. plot
    "count", # 3. plot
    "count", # 4. plot
    "count", # 5. plot
    "count", # 6. plot
    "count", # 7. plot
    "count", # 8. plot
    "count", # 9. plot
    "count", # 10. plot
    "count" # 10. plot
  ),
  subtitle = c(
    "TOC \n%", # 1. plot
    rep("", 10) # 2. ... 11. plot
  )
) # Ende plot.depth(..)
# labelling y-axes with mouse
par(xpd=TRUE) -> original # save graphical parameter
  # from left to right
  locator(1) -> where # with mouse
  text(where$x, where$y, "depth special (m)", adj=0, srt=90, col= "blue")
  # adj=0 linksbündig; srt=90 Grad drehen
  locator(1) -> where # with mouse
  text(where$x, where$y, "zones", adj=0, srt=90, col= "red")
  locator(1) -> where # with mouse
  text(where$x, where$y, "depth (m)", adj=0, srt=90)
par(original) # graphical parameters back
```



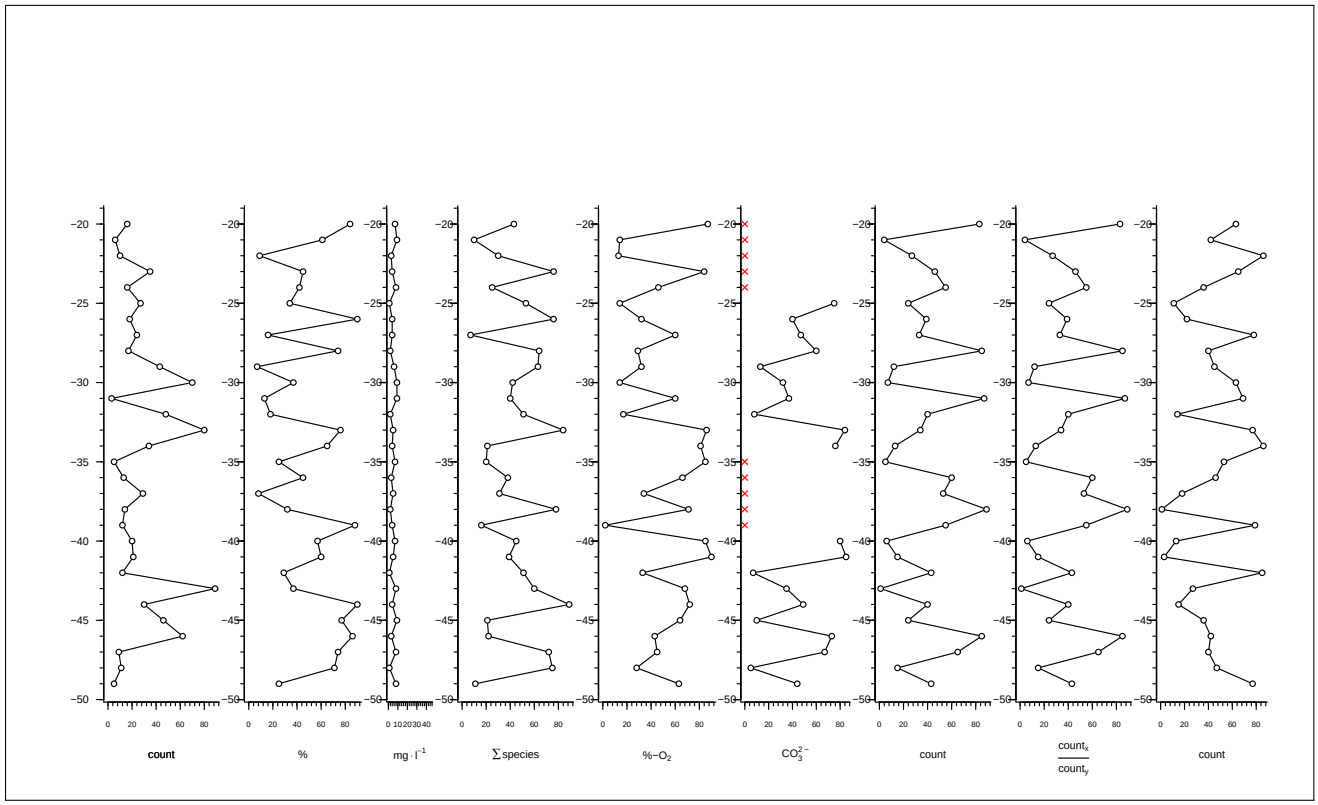
Example 8: equal scaling of x-axes

```
plot.depth(testabiotic,
  min.scale.level=1,
  min.scale.rel=1
)
```



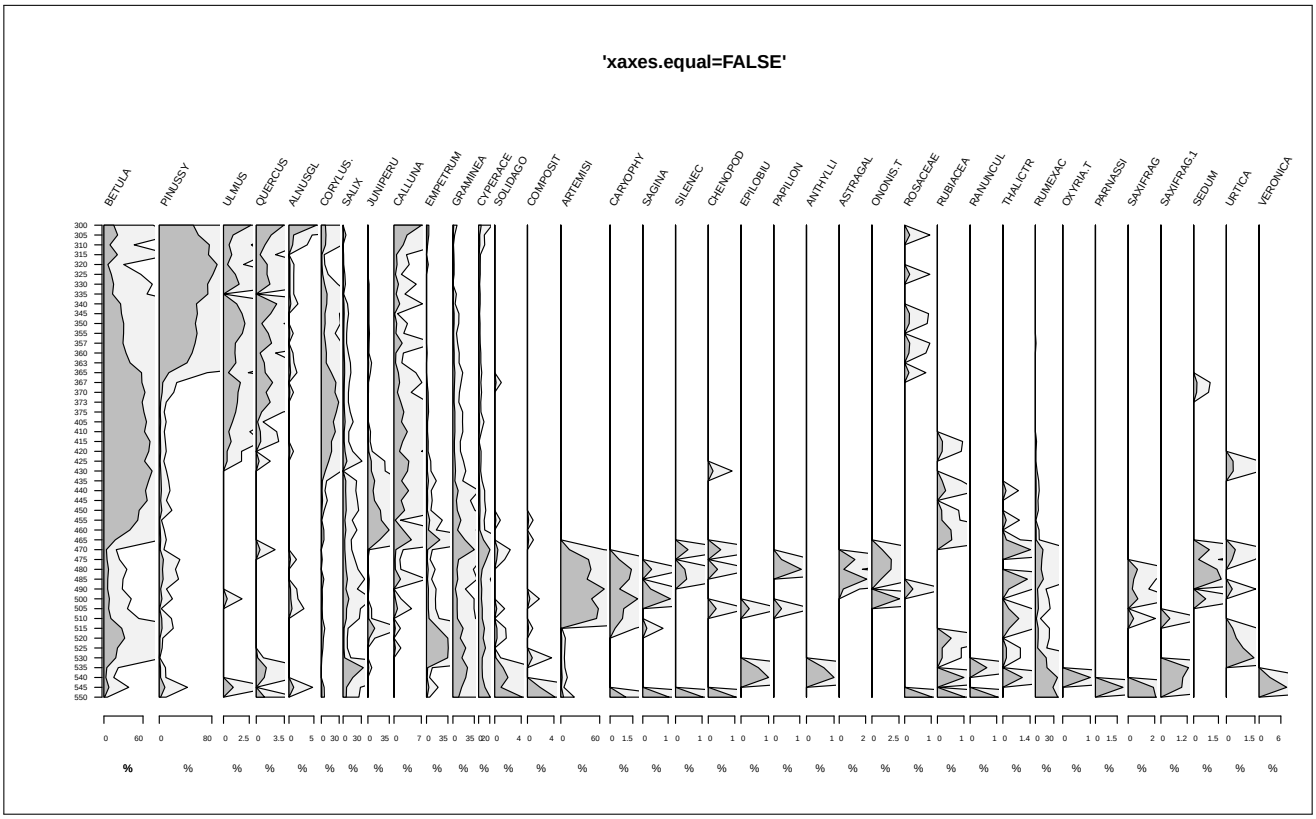

Example 9: labelling

```
par(las = 1) # label assignment
plot.depth(test,
  yaxis.num="s",
  colnames = c(T,T,F,T,F,F,T,T,T), # T - TRUE, F - FALSE
  xlabel = list("count", "%",
    expression(mg%.%l^-1), # mg/l-1
    expression(sum(species)), #
    expression(paste("%-",O[2])), # % - O2
    expression(CO[3]^paste(2," -")), # CO2-  3
    "count","",
    "count"
  ),
  subtitle = list("","","","","","","","",
    expression(over(count[x],count[y])),""  # fract: count_x / count_y
  )
)
```



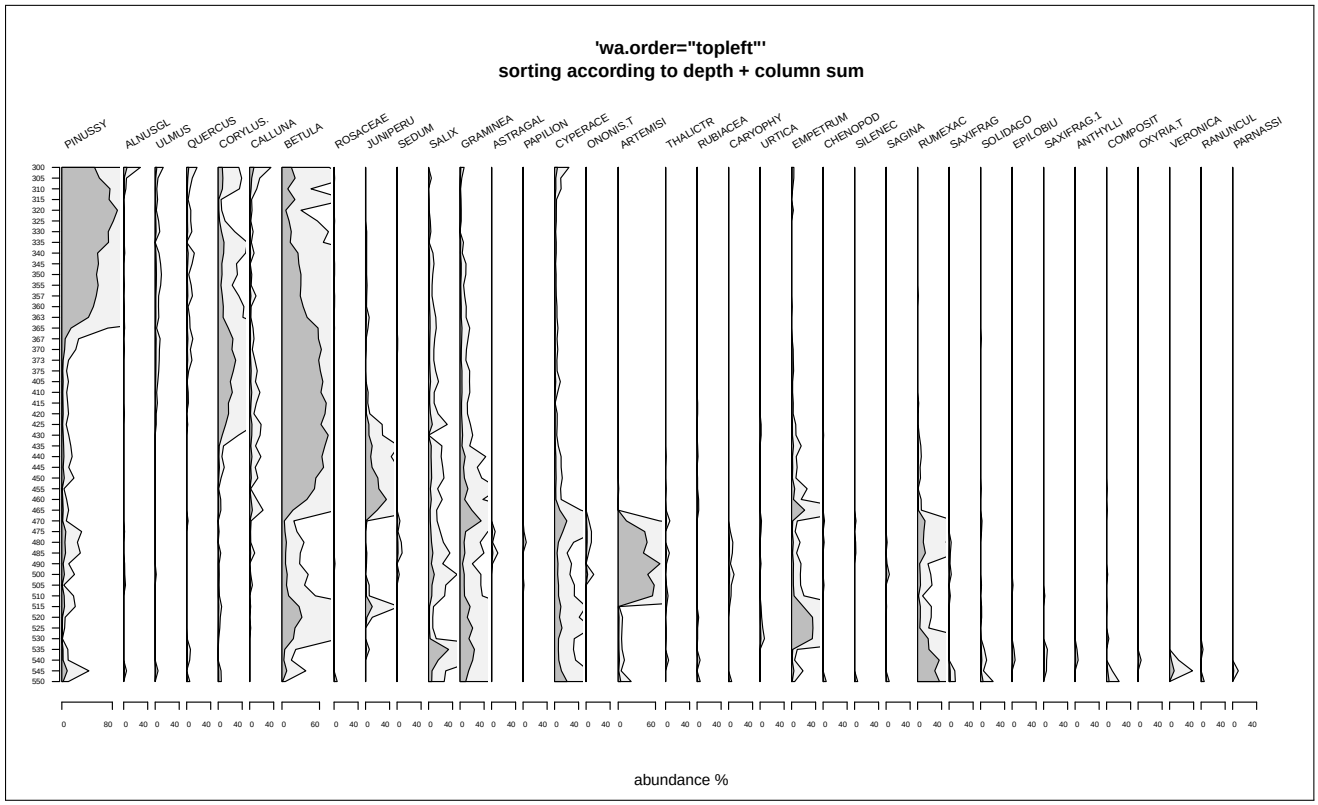

Example 10: no equal scaling of x-axes

```
par(las = 1) # label assignment
# palaeo → http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/analysis.htm
data(aber, package="palaeo")
plot.depth(aber,
  yaxis.first=FALSE,
  polygon=TRUE,# Polygon
  min.scaling=TRUE,# switch on minimum-scaling
  plot.type="n",# no points
  xaxes.equal=FALSE, # no equal x axes
  xaxes.adjustlabels=TRUE, # shrink x-labels
  xlabel="%",
  ylabel="sample number",
  xaxis.ticks.minmax=TRUE,# show only min to max
  yaxis.ticks=FALSE,# no  main y ticks
  bty="n"# no box
)
title("'xaxes.equal=FALSE'")
```




Example 11: weigthed averaging according to depth and `colSum`

```
par(las = 1) # label assignment
# http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/analysis.htm
data(aber, package="palaeo")
plot.depth(aber,
  yaxis.first=FALSE,
  polygon=TRUE,# Polygon
  min.scaling=TRUE,# switch on minimum-scaling
  min.scale.level=0.75,
  mar.outer=c(1,3,0,3),
  xaxes.adjustlabels=TRUE, # shrink x-labels
  plot.type="n",# no points
  #xlabel="%",
  rotation=30,
  ylabel="sample number",
  xaxis.ticks.minmax=TRUE,# show only min to max
  yaxis.ticks=FALSE,# no  main y ticks
  bty="n",# no box
  wa.order = "topleft" # plot abundant species topleft
)
title("'wa.order=\"topleft\"'\nsorting according to depth + column sum",sub="abundance in %")
```


'wa.order="topleft"'
sorting according to depth + column sum
PINUSSY
ALNUSGL
ULMUS
QUERCUS
CORYLUS
CALLUNA
BETULA
ROSACEAE
JUNIPERU
SEDUM
SALIX
GRAMINEA
ASTRAGAL
PAPILION
CYPERACE
ONONIS.T
ARTEMISI
THALICTR
RUBIACEA
CARYOPHY
URTICA
EMPETRUM
CHENOPOD
SILENEC
SAGINA
RUMEXAC
SAXIFRAG
SOLIDAGO
EPILOBIU
SAXIFRAG.1
ANTHYLLI
COMPOSIT
OXYRIA.T
VERONICA
RANUNCUL
PARNASSI
abundance %

## 2 Source codes

**User function 1:** `plot.depth()` draws depth profiles of drilling cores, pollen data or other proxies. The whole function can be used in R parsing the content of a text file once named for instance `myuserfunction.R` with `source("path/to/myuserfunction.R")`. Take care of line breaks indicated by ↩.
Log-history:
*2010-03-27* v1.2: added 50 columns threshold: plotting with a warning
*2010-03-25* added xaxes.adjustlabels.

```
########################################
# see also below at function's arguments #
########################################
# data                dataset as a data.frame or matrix: with first column as depth
# yaxis.first=TRUE    TRUE/FALSE does first column contain depth datas?
# yaxis.num="n"       switch on/off numbers at remaining y-axes on="s" off="n"
# xaxes.equal=TRUE,   equal scaling of xaxes; can be set individually by  c(...)
# xaxis.ticks.minmax=FALSE, only minmax is drawn; can be set individually by  c(...)
# xaxis.num="s",      switch on/off numbers+ticks at x-axis on="s" off="n"
# bty="L"             boxtype as in plot: L, c, o ...; can be set individually by  c(...)
# l.type="solid"      line type default; can be set individually by  c(...)
# l.width=1,          line width; can be set individually by  list(...) or nested with c()
# lp.color="black"    line color; can be set individually by  c(...)
# plot.type="o"       type of plot - as in plot(); can be set individually by  c(...)
#    possible: o, b, c, n, h, p, l, s, S
#    "p" for points,
#    "l" for lines,
#    "b" for both,
#    "c" for the lines part alone of "b",
#    "o" for both "overplotted",
#    "h" for "histogram" like horizontal lines,
#    "s" or "S" for stair steps,
#    "n" for no plotting.
# plot.before=NULL   evaluate/draw before plotting
#                    eg.: grid() as expression(); nested: 'expression(grid())'
#                    can be set individually by  list(...) or nested with expression()
# plot.after=NULL    evaluate/draw after plotting
#                    additional graphics eg.: points(), lines() as expression()
#                    expression(lines(...)) - can be set individually by  list(...)
#                    or nested with expression()
# yaxis.lab=FALSE     no additional labels on remaining y-axes
# yaxis.ticks=TRUE    add y-ticks to graph ?
# axis.top=list(c(FALSE, FALSE)) -- x-axis also on top?
#                    call for axis and labels as c(axis=TRUE, labels=TRUE)
#                    can be nested with list( c(T,F), c(T,T), ...)
# nx.minor.ticks=5   number of intervals at x-axis if package Hmisc loadable
#                    can be set individually by  c(...)
# ny.minor.ticks=5   number of intervals at y-axis if package Hmisc loadable
#                    can be set individually by  c(...)
# mar.outer=c(1,6,4,1) --  margin at outer side: c(bottom, left , top, right)
# mar.top=9          margin at the top
# mar.bottom=5       margin at the bottom
# txt.xadj=0.1       align text at plot-top in x-axis direction: 0...1 left...right
# txt.yadj=0.1       align text at plot-top in y-axis direction: in scalenumbers
#                    + -> to the top - -> to the bottom
# colnames=TRUE      can be set individually by  c(...)
# rotation=60        text rotation: can be set individually by  c(...)
# p.type=21          type of points like pch in points()
```

```
#                    can be set individually by list(...) also nested
# p.bgcolor="white"  point background color: can be set individually by c(...)
# p.size = 1         point size: can be set individually by list(...) also nested
# subtitle=""        subtitle: can be set individually by list(...)
# xlabel=""          x-labeling: can be set individually by list(...)
# main=""            titel of individual plots: can be set individually by list(...)
# polygon=FALSE      plot polygon on/off: can be set individually by  c(...)
# polygon.color="gray" -- color of polygon plot; can be set individually by  c(...)
# show.na=TRUE       show NA values as red cross
# min.scale.level=0.2
#                   0...1 if data are less than 0.2(=20%) from maximum of the data
#                   than draw raltive 'min.scale.rel'-width for the plot
# min.scale.rel=0.5,
#                   0...1 relative space for minimal data
#                   1 means maximal width
# min.scaling=FALSE --  add upscaling plots to rare data; can be set individually by  c(...)
# color.minscale="gray95" -- color for rare scaled data; can be set individually by list(...)
# wa.order="none", sort variables according to the weighted average with y
#                  "bottomleft", "topleft" from strat.plot(palaeo - pkg)
# ...  passed to function 'lines()'
#
##############################
plot.depth <- function(
    data,# data.frame assumed with first column as y-axis
  # axis settings
    yaxis.first=TRUE, # is 1st data-column 1st y-axis?
    yaxis.num="n",    # supress labelling at remaining y-axis
    xaxis.num="s",    # show labelling at x-axis
    xaxes.equal=TRUE, # equal scaling of all x-axes
    cex.x.axis=par("cex.axis")*0.8,# size x-axis labels
    cex.y.axis=par("cex.axis")*0.8,# size y-axis labels
  # ticks/labels
    xaxis.ticks.minmax=FALSE, # only min-max?
    xaxes.adjustlabels=if(xaxis.ticks.minmax) TRUE else FALSE, # adjust labels of x-axes
    yaxis.lab=FALSE,  # axis labels on remaining y-axis?
    yaxis.ticks=TRUE, # add y-ticks to graph?
    axis.top=list(c(FALSE, FALSE)),# axis on top? c(axis=TRUE, labels=TRUE)
    nx.minor.ticks=if(xaxis.ticks.minmax) 0 else 5,# number intervals for minor ticks; 0 hides them
    ny.minor.ticks=5, # number intervals for minor ticks; 0 hides them
    bty="L",          # boxtype
  # plotting types: "o", "b", "c", "n", "h", "p", "l", "s", "S" see example(points)
    plot.type="o",    # point-plot type
    plot.before=NULL, # something to plot BEFORE the graph is drawn?
    plot.after=NULL,  # something to plot AFTER the graph is drawn?
  # lines
    l.type="solid",     # line type
    l.width=1,          # line width
    lp.color="black", # line/point color
  # points
    p.type=21,          # point type
    p.bgcolor="white",# point background color
    p.size = 1,         # point size
  # polygon
    polygon=FALSE, # plot Polygon?
    polygon.color="gray", # color polygon
  # NA - data (not available)
    show.na=TRUE,        # show missing values?
```

```
# margins
  mar.outer=c(1,6,4,1),# outer margin of whole plot
  mar.top=   9,# margin on the top
  mar.bottom=5,# margin on the bottom
  mar.right= 0,# margin on the right side
# minimum sclaing: for minimum data
  min.scaling=FALSE,
  color.minscale="gray95",# color for minimum scaled data
  min.scale.level=0.2,     # 0...1 plot's width: if data take less than 0.2(=20%)
  min.scale.rel  =0.5,     # 0...1 relative space for all minimal data
# texts, labels, subtitles
  txt.xadj=0.1,  # at plot-top: x-adjusting text in x-axis direction
  txt.yadj=0.1,  # at plot-top: y-adjusting text in y-axis direction
  colnames=TRUE, # add columnames
  rotation=60,   # columnames rotation
  subtitle="",   # below every plot
    xlabel="",   # x-labels
    ylabel="",   # first y-label
    main  ="",   # title for each plot
# weighted averageing of data
  wa.order="none", # "bottomleft", "topleft"
  ... # other arguments passed to other functions
){
# ------8<---- function minor.tick start
# from Hmisc package added: axis=c(1,2) + '...' for axis( , ...)
# axis=c(3,4) draws also ticks on top or right
minor.tick <- function (nx = 2, ny = 2, tick.ratio = 0.5, axis=c(1,2), ...)
{
    ax <- function(w, n, tick.ratio) {
        range <- par("usr")[if (w == "x")
            1:2
        else 3:4]
        tick.pos <- if (w == "x")
            par("xaxp")
        else par("yaxp")
        distance.between.minor <- (tick.pos[2] - tick.pos[1])/tick.pos[3]/n
        possible.minors <- tick.pos[1] - (0:100) * distance.between.minor
        low.minor <- min(possible.minors[possible.minors >= range[1]])
        if (is.na(low.minor))
            low.minor <- tick.pos[1]
        possible.minors <- tick.pos[2] + (0:100) * distance.between.minor
        hi.minor <- max(possible.minors[possible.minors <= range[2]])
        if (is.na(hi.minor))
            hi.minor <- tick.pos[2]
        if (.R.)
            axis(if (w == "x")
                axis[1]
            else axis[2], seq(low.minor, hi.minor, by = distance.between.minor),
                labels = FALSE, tcl = par("tcl") * tick.ratio, ...)
        else axis(if (w == "x")
            axis[1]
        else axis[2], seq(low.minor, hi.minor, by = distance.between.minor),
            labels = FALSE, tck = par("tck") * tick.ratio, ...)
    }
    if (nx > 1)
        ax("x", nx, tick.ratio = tick.ratio)
    if (ny > 1)
```

```
        ax("y", ny, tick.ratio = tick.ratio)
    invisible()
}
# ------8<---- function minor.tick end
# check data
if(!is.data.frame(data) & !is.matrix(data))
  stop(paste("\nFunction \'plot.depth(data, ...)\' expect a data.frame or matrix!\n Your data ↩
  are: \'",mode(data),"\'.\n  Use \"as.data.frame(depthdata) -> depthdata\" or ↩
  \"as.matrix(depthdata) -> depthdata\".", sep=""))
if(ncol(data) < 2)
  stop("\nStop: at least 2 columns in the data!")
if(ncol(data) > 50 && yaxis.first==FALSE){
  warning("\nOnly the first 50 columns will be drawn.")
  data <- data[,1:50]
}
if(ncol(data) > 51 && yaxis.first==TRUE){
  warning("\nOnly the first 50 columns will be drawn.")
  data <- data[,1:51]
}
nc <- ncol(data) # number of columns
nr <- nrow(data) # number of rows
if(yaxis.first==TRUE){# if 1st column is first y-axis
  nc.data <- nc-1 # number of columns for drawing
  draw <- 2:nc # what should be drawn
  y.depth <- data[,1] # depth scale
  y.axfirst.type ="s"
  warning("plot.depth() assumes yaxis.first=TRUE\n")
}
else{# no first y-axis
  nc.data <- nc# number of columns for drawing
  draw <- 1:nc # what should be drawn
  y.depth <- (1:nr)*(-1) # depth scale
  warning("Your data will be drawn as category numbers (=number of rowname)\n")
  y.axfirst.type ="n"
}
# weighted averageing order
# (from package paleo http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/analysis.htm)
if (wa.order == "topleft" || wa.order == "bottomleft") {
    colsum <- colSums(data[,draw])
    opt <- (t(data[,draw]) %*% y.depth)/colsum
    if (wa.order == "topleft")
        opt.order <- rev(order(opt))
    else opt.order <- order(opt)
    draw <- opt.order
    cat("Column Index (wa.order):",draw,"\n")
    # data <- data[, opt.order]
}

x.maximum <- max(apply(data[,draw],2,max, na.rm=TRUE))
x.maxima <- apply(data[,draw],2,max, na.rm=TRUE)
# cat(x.maximum) control
x.max <- apply(data[,draw],2,max, na.rm=TRUE)
stopifnot(0 <= min.scale.level && min.scale.level <=1)
stopifnot(0 <= min.scale.rel && min.scale.rel <=1)
par(no.readonly=TRUE) -> original # save graphical settings
# ---8<--- get settings for layout
# maxima from each column
```

```
  apply(data[,draw],2,max, na.rm=TRUE) -> x.widths
  xwidths <- NULL # temporary vector
  for(i in 1:length(x.widths)){# for each maximum
    # allow individual settings for plots via index
    ifelse(length(xaxes.equal)==nc.data, equal.i <- i, equal.i <- 1)
    ifelse(x.widths[i]/max(x.widths) <= min.scale.level,
      {# x.widths/max <= 0.5
        xwidths[i] <- min.scale.rel # 0...min.scale.rel
        # maximum for x-axis
        ifelse(xaxes.equal[equal.i]==FALSE,
          {# draw xaxes-equal FALSE:
            x.max[i] <- max(data[,draw[i]], na.rm=TRUE) # maximum of column
          }, {#  draw xaxes-equal TRUE
            x.max[i] <- x.maximum * min.scale.rel # maximum of all data
          }
        ) # xaxes.equal
      },{# x.widths/max > 0.5
        xwidths[i] <- x.widths[i]/max(x.widths) # 0...1
        # maximum for x-axis
        ifelse(xaxes.equal[equal.i]==FALSE,
          {# FALSE:
            x.max[i] <- max(data[,draw[i]], na.rm=TRUE) # maximum of column
          },{
            x.max[i] <- x.maxima[i] # maximum of all data
          }
        ) # xaxes.equal end
      }
    ) # minscale.level end
  }
  # set layout
  x.widths <- xwidths
  layout(matrix(1:nc.data,1 , nc.data), widths=x.widths)
  # ---8<--- end get settings for layout
      par(mar=c(
        mar.bottom, # bottom
        0, # left
        mar.top, # top
        ifelse(yaxis.num=="s", 1.5 + mar.right, mar.right) # right
        )+0.1,
        xpd=NA # NA to get no overplotted text
      )
    for(i in 1:length(draw)){# draw each plot
    #cat(i,"\n")
      # check for lists in list() or c() in differrent options
      ifelse(length(plot.type)          == nc.data, n.i <- i,      n.i <- 1)
      ifelse(length(ny.minor.ticks)     == nc.data, ny.i <- i,     ny.i <- 1)
      ifelse(length(nx.minor.ticks)     == nc.data, nx.i <- i,     nx.i <- 1)
      ifelse(length(polygon)            == nc.data, p.i <- i,      p.i <- 1)
      ifelse(length(min.scaling)        == nc.data, min.i <- i,    min.i <- 1)
      ifelse(length(l.type)             == nc.data, lt.i <-i,      lt.i <- 1)
      ifelse(length(lp.color)           == nc.data, lc.i <-i,      lc.i <-1)
      ifelse(length(l.width)            == nc.data, lw.i <-i,      lw.i <- 1)
      ifelse(length(p.type)             == nc.data, pt.i <-i,      pt.i <- 1)
      ifelse(length(p.size)             == nc.data, pw.i <- i,     pw.i <- 1)
      ifelse(length(p.bgcolor)          == nc.data, pbg.i <- i,    pbg.i <- 1)
      ifelse(length(colnames)           == nc.data, col.i <- i,    col.i <- 1)
      ifelse(length(rotation)           == nc.data, r.i <- i,      r.i <- 1)
```

```
ifelse(length(xlabel)             == nc.data, xlab.i <- i,   xlab.i <- 1)
ifelse(length(subtitle)           == nc.data, sub.i <- i,    sub.i <- 1)
ifelse(length(main)               == nc.data, main.i <- i,   main.i <- 1)
ifelse(length(plot.before)        == nc.data, before.i <- i, before.i <- 1)
ifelse(length(plot.after)         == nc.data, after.i <- i,  after.i <- 1)
ifelse(length(axis.top)           == nc.data, axtop.i <- i,  axtop.i <- 1)
ifelse(length(xaxis.num)          == nc.data, xnum.i <- i,   xnum.i <- 1)
ifelse(length(xaxis.ticks.minmax) == nc.data, xminmax.i <- i,xminmax.i <- 1)
# margins of x-axis
if(i==1) par(oma=mar.outer, xaxt=xaxis.num[xnum.i])
else par(xaxt=xaxis.num[xnum.i])
# axis ticks and labelling
par(
  mgp=c(3, ifelse(yaxis.num=="s" && i > 1, 0.3, 1), 0)
)
# minimum
ifelse(
  min(data[,draw[i]], na.rm=TRUE) > 0,
  x.min <- 0,# 0... max
  x.min <- min(data[,draw[i]], na.rm=TRUE) # min...max
)
# draw plot()
par(xpd=FALSE)# to draw also ylabel
plot(data[,draw[i]], y.depth,
  ann=ifelse(i==1,TRUE, FALSE),# nichts an Achse
  type="n",# Punkttyp
  yaxt=ifelse(i==1,y.axfirst.type, yaxis.num),# y-Achse an/aus
  xlim=c(x.min,x.max[i]),
  bty=ifelse(length(bty)==nc.data, bty[i], bty),
  xlab=ifelse(length(xlabel)==nc.data, xlabel[i], xlabel),
  ylab=ylabel,#ifelse(i==1, ylabel, ""),
  panel.first = eval(plot.before[[before.i]]),
  xaxt = "n" # no x-axis
)
par(xpd=FALSE)
if(i==1 && y.axfirst.type=="n"){ # draw extra first y-axis
  axis(side=2, labels=rownames(data),
    at=(1:nr)*(-1), cex.axis=cex.y.axis
  )
  box(bty=bty)
}
# draw x-axis
axTicks(1,
  axp=if(xaxis.ticks.minmax[xminmax.i]==TRUE) {c(par()$xaxp[1:2], 1)} else NULL
) -> x.axis
for(itemp in seq.int(length(x.axis) -> nx))
  text(
    x=seq(from=x.axis[1] , to=x.axis[nx], length.out=nx)[itemp],
    y=par("usr")[3] - par()$cxy[2], # coordinates - character hight
    adj = if(xaxes.adjustlabels) seq(from=0,to=1, length.out=nx)[itemp] else 0.5,
    labels = x.axis[itemp],
    cex=cex.x.axis,
    xpd=NA
  )
  axis(1, at=x.axis, labels=FALSE)

# minor ticks if package Hmisc is installed
```

```
  if(yaxis.ticks==FALSE && i > 1) ny.minor.ticks[ny.i] <- 0

  if(require(Hmisc)) {minor.tick(
      ny=ifelse(i==1 && y.axfirst.type=="n", 0, ny.minor.ticks[ny.i]),
      nx=nx.minor.ticks[ny.i]
    )
  }
  else warning("Install package 'Hmisc' to add minor ticks on axes")

  # y-axis for remainig axes
  if(i > 1) { axis(side=2,
    labels=yaxis.lab,
    tick=yaxis.ticks,
    cex.axis=cex.y.axis
    )
  }
  # x-axis top
  if(length(axis.top[[axtop.i]])==2){
    if(axis.top[[axtop.i]][1]==TRUE){
      axis(side=3, labels=axis.top[[axtop.i]][2], tick=TRUE, tcl=0.5, cex.axis=cex.x.axis)
      minor.tick(ny=0, nx=nx.minor.ticks[ny.i], axis=c(3,4), tcl=0.25)
    }
  }
  else warning("Option 'axis.top' wants 2 arguments as list(...):",
  "\n2nd argument is for numbers on axis, so eg.: axis.top=list(c(T, F))")
  # labelling of columns
  if(colnames[col.i]==TRUE){
    min(par()$usr[1:2]) -> x.text
    abs(max(par()$usr[1:2])-x.text)*txt.xadj -> x.adj # %-width of x-axis
    max(par()$usr[3:4]) -> y.text
    par(xpd=NA) # NA to get no overplotted text
      text(x.text+x.adj, y.text+txt.yadj, labels=colnames(data)[draw[i]], adj=0, ↩
srt=rotation[r.i] )
    par(xpd=FALSE)
  }
  # title subtitle, xlabels
  title(
    sub=subtitle[[sub.i]],
    xlab=xlabel[[xlab.i]],
    main=main[[main.i]]
  )

  # pseudo histograms; width can be set with option 'l.width'
  if( plot.type[n.i] =="h"){
    for(n in 1:nr){
      x <- c(0,data[n,draw[i]])
      y <- c(y.depth[n], y.depth[n])
      par(lend="butt") # line-End
      lines(x,y,
        lty=l.type[[lt.i]],
        lwd=l.width[[lw.i]],
        col=ifelse(length(lp.color[[lc.i]])==nr, lp.color[[lc.i]][n], lp.color[[lc.i]]),
      )
      par(lend="round")
    }
  }
  # Polygonplot
```

```
  if (polygon[p.i]==TRUE){
    # add zero values to margins where NA values occur
    # eg.: NA NA 23  7 34 84 NA NA
    #      -1 -2 -3 -4 -5 -6 -7 -8
    # to: NA NA| 0| 23  7 34 84 | 0| NA NA
    #   -1 -2 |-3| -3 -4 -5 -6 |-6| -7 -8
    data.null <- data.frame(
      rbind(
        if(!is.na(data[1, draw[i]])) cbind(y.depth[1], 0),
        cbind(y.depth[1], data[1,draw[i]])
      )
    )
    for(r in 2:nr){
      data.null <- rbind(
        as.matrix(data.null),
        # r-1==NA && r!=NA -> 0r
        if(is.na(data[r-1, draw[i]]) && !is.na(data[r, draw[i]])) cbind(y.depth[r], 0),
        # r-1!=NA && r==NA -> 0r-1
        if(!is.na(data[r-1, draw[i]]) && is.na(data[r, draw[i]])) cbind(y.depth[r-1], 0),
        as.matrix( cbind(y.depth[r], data[r, draw[i]]) ),
        # r==nr -> 0r
        if(r==nr && !is.na(data[r, draw[i]])) cbind(y.depth[r], 0)
      )
    }

    # min.scaling
    if (min.scaling[min.i]==TRUE || min.scaling[min.i] > 0){
      # default 5-scaled
      if(min.scaling[min.i]==TRUE) min.scaling[min.i] <- 5
      polygon(
        data.null[, 2]*min.scaling[min.i] ,
        data.null[, 1],
        col=ifelse(length(color.minscale)==nc.data,color.minscale[[i]],color.minscale[1]),
        xpd=FALSE
      )
      # scaling as message message
      message(paste("Column \'", colnames(data)[draw[i]],"\' is scaled ",
        min.scaling[min.i], "-times to original data.", sep="")
      )
    }# end min.scaling
    # default polygon
    polygon(
      data.null[, 2],
      data.null[, 1],
      col=ifelse(length(polygon.color)==nc.data, polygon.color[i], polygon.color),
      xpd=FALSE
    )
    # warning/recommendation, if NA in data
    if(any(is.na(data[,draw[i]]))) {warning("Column \'",
      colnames(data)[draw[i]], "\' contain NA-values.",
      "\nOther possibility to draw: switch off drawing polygon with option \'polygon=c(T, T, F, ↩
...)\'",
      "\nand set the column to \'F\' (FALSE) than draw histogram-like lines with the following ↩
two options:",
      "\n  plot.type=c(...,\"h\",...),\n  l.width=c(..., 15, ...), ",call. = FALSE)
    }
  }# polygon end
```

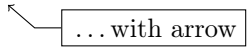
```
      if(show.na==TRUE){# draw red cross, at NA-value position
        which(is.na(data[,draw[i]])) -> na.index
        # add red 'x'
        points(y=y.depth[na.index], x=rep(0, length(na.index)), pch=4, col="red")
        if(length(na.index) > 0) {
          message("With option 'show.na=FALSE' you can switch off red crosses.")
        }
      }
      # points lines ....
      lines(data[,draw[i]], y.depth,
        ann=FALSE,# nichts an Achse
        type=ifelse(plot.type[n.i]=="h", "n",plot.type[n.i]),# type of points
        lty=l.type[[lt.i]],
        lwd=l.width[[lw.i]],
        pch=p.type[[pt.i]],
        col=lp.color[[lc.i]],
        bg=p.bgcolor[[pbg.i]],
        panel.last = eval(plot.after[[after.i]]),
        cex = p.size[[pw.i]],
        ...
      )
    }# end for i
  if(xaxis.ticks.minmax && xaxes.adjustlabels) cat("x-labels are adjusted...\n")
  par(original)
}# end plot.depth
cat("#####################################################################\n# read userfunction ↩
    plot.depth(mydata, yaxis.first=TRUE):       #\n#   * plots up to 50 species as abundances for ↩
    drilling cores   #\n#   * assumes (by default) first column in mydata being y-axis  ↩
    #\n#####################################################################\n")
```

**User function 2:** Function `asking(question, answerNo, answerYes)` asks the user a y/n question.

```
## ask user what to do
asking <- function(
    question="Are you a satisfied R user?",
    answerNo="This is impossible. YOU LIED!",
    answerYes="I knew it.",
    prompt="n",
    stop = FALSE # can force to stop the script
  ) {
  question <- paste(question,"(y/n)\n")
  cat("\a") # alarm for Linux
  ANSWER <- readline(question)
  #   cat(ANSWER)
  if (substr(ANSWER, 1, 1) == prompt){# no answer
    cat(answerNo,"\n")
    return(FALSE)# returns FALSE
    if(stop)
      stop("Breake here - user stopped.")
  }else{
    cat(answerYes,"\n") # green
    return(TRUE) # returns TRUE
  }
}
# asking() # after an idea of R-example ?readline
# asking("Continue? It takes long time: ","Stop here.","OK, continue ...")
# if(asking(...)) {then do a lot o R stuff} else {do another lot o R stuff}
```

```
cat("Read asking(question, no, yes): asks the user for input...\n")
```

**User function 3:** Function `arrowLegend()` places a legend by mouse click with an indicating arrow.




```
# draw a legend at given points additionally indicated by an arrow
arrowLegend <- function(
  text,       # legend text
  npoints=2, # pick 1-3 points with mouse
  lineHoirz=TRUE, # 2nd → 3rd point: draws a horizontal line
  adj=c(0, 0.5), # adjusting text
  ... # further args from legend()
  ){
  if(missing(text))
    stop("Stop here 'text' missing. Usage: arrowLegend(text='my text')")
    if(npoints >= 1 &  npoints <= 3){
      xjust = 0 # xalign
      cat(paste("Please select", npoints, "points for the arrow. Last point places the legend...\n"))
      xy <- locator(npoints)
      if(npoints!=1){# 2 or 3 points
        # adjust Box+Text to the given x-coordinates
        xjust <- if(xy$x[npoints-1] < xy$x[npoints]) 0 else 1
        if(npoints ==3) {# 3 points? add segments(...)
          if(lineHoirz){
            xy$y[2:3]  <- mean(xy$y[2:3])
          }
          segments( xy$x[2], xy$y[2], xy$x[3], xy$y[3] )
        }
        arrows( xy$x[1], xy$y[1], xy$x[2], xy$y[2] , code = 1, length = 0.12)
      }# 2 or 3 points
      legend(
        x = xy$x[npoints] , y = xy$y[npoints],
        legend = text,
        yjust=0.5,
        # do x-adjusting
        xjust = xjust, adj = adj,
        ...
      )
    }else {# all other points
      cat(paste(npoints, "points are not allowed...\n"))
    }
    cat('xjust:',xjust,"\n")
} # end arrowLegend()
cat("Read arrowLegend(text): draw a legend at given mouse click points additionally indicated by an ↩
    arrow...\n")
```

**User function 4:** `grDeviceUserSize()`'s mode is «landscape», i.e. a ratio of 12/7. It can be used to force a special size of graphical's device but tries to keep it if the size fits more or less, otherwise the window will be replaced by a new one.

```
## defines size of graphic device on Linux/Windows Apple?
grDeviceUserSize <- function(scale=12/7, dinMin = 6.9, dinMax = 7.1){
  # some information prompt
  cat("w:",mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
      "h:",mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1), 'dev.cur():',dev.cur(),"\n")
  if(dev.cur()==1){ # wenn kein device (also NULL) dann:
```

```
    # neues Grafikfenster
    switch(tolower(Sys.info()["sysname"]),
      linux     = {
        X11(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Linux
      windows = {
        windows(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Windows
      quartz   = {
        quartz(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      } # end MacOS
    )# end switch(Sys.info())
  }else if(
    any(
      par()$din[1] < dinMin * ifelse(scale>=1, scale, 1) ||
        par()$din[1] > dinMax * ifelse(scale>=1, scale, 1),
      par()$din[2] < dinMin * ifelse(scale<1, 1/scale, 1) ||
        par()$din[2] > dinMax * ifelse(scale<1, 1/scale, 1)
    )
  ){ # andernfalls
    #cat(par()$din,"\n")
    dev.off() # device off = close graphic device
    # neues Grafikfenster
    switch(Sys.info()["sysname"],
      linux     = {
        X11(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Linux
      windows = {
        windows(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Windows
      quartz   = {
        quartz(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      } # end MacOS
    )# end switch(Sys.info())
  }
} # end grDeviceUserSize()
cat("Read grDeviceUserSize(): defines size of graphic device. Default is landscape...\n")
```

# Index