Package 'bda'

March 12, 2024

Version 18.2.2

Date 2024-03-12

Title Binned Data Analysis

Author Bin Wang <bwang831@gmail.com>

Maintainer Bin Wang <bwang831@gmail.com>

Depends R (>= 3.5.0), boot

Description Algorithms developed for binned data analysis,
gene expression data analysis and
measurement error models for ordinal data analysis.

License Unlimited

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-03-12 17:30:03 UTC

R topics documented:

---

bda *Binned (Grouped) Data Analysis*

Description

This package collects algorithms for binned (grouped) data analysis.

References

Wang B. (2020). *A Zipf-plot based normalization method for high-throughput RNA-seq data*. PLOS ONE 15(4): e0230594.

Wang, B. and Wang, X-F. (2015). *Fitting The Generalized Lambda Distribution To Pre-binned Data*. Journal of Statistical Computation and Simulation. Vol. 86 , Iss. 9, 1785-1797.

Ge C, Zhang S-G, Wang B (2020) Modeling the joint distribution of firm size and firm age based on grouped data. PLoS ONE 15(7): e0235282. doi.org/10.1371/journal.pone.0235282

Wang, B. and Wertelecki, W. (2013) Density Estimation for Data With Rounding Errors, Computational Statistics and Data Analysis, 65: 4-12. doi: 10.1016/j.csda.2012.02.016.

Wang, X-F. and Wang, B. (2011) Deconvolution Estimation in Measurement Error Models: The R Package decon, Journal of Statistical Software, 39(10), 1-24.

---

binning *Data Binning*

Description

To bin a univariate data set in to a consecutive bins.

Usage

```
binning(x, counts, breaks,lower.limit, upper.limit)
```

Arguments

x	A vector of raw data. 'NA' values will be automatically removed.
counts	Frequencies or counts of observations in different classes (bins)
breaks	The break points for data binning.
lower.limit, upper.limit	The lower and upper limits of the bins.

Details

To create a 'bdata' object. If 'x' is given, a histogram will be created. Otherwise, create a histogram-type data using 'counts' and 'breaks' (or class limits with 'lower.limit' and/or 'upper.limit').

Value

ll	lower limits
ul	upper limits
freq	frequencies
xhist	histogram
xZipf	Zipf plot

Examples

```
y <- c(10, 21, 56,79,114,122,110,85,85,61,47,49,47,44,31,20,11,4,4)
x <- 14.5 + c(0:length(y))
out1 <- binning(counts=y, breaks=x)
plot(out1)

z = rnorm(100, 34.5, 1.6)
out1 <- binning(z)
plot(out1)

data(FSD)

x <- as.numeric(FirmAge[38,]);
age <- c(0,1:6,11,16,21,26,38);
y <- binning(counts=x, lower.limit=age)
plot(y)
plot(y, type="Zipf")

x <- as.numeric(FirmSize[38,]);
names(FirmSize)
ll <- c(1,5,10,20,50,100,250,500,1000,2500,5000,10000);
ul <- c(4,9,19,49,99,249,499,999,2499,4999,9999,Inf)
y <- binning(counts=x, lower.limit=ll,upper.limit=ul)
plot(y)
plot(y, type="Zipf")
```

bootkde *Density estimation for data with rounding errors*

Description

To estimate density function based on data with rounding errors.

Usage

```
bootkde(x,freq,a,b,from, to, gridsize=512L,method='boot')
```

Arguments

x,freq	raw data if 'freq' missing, otherwise as distinct values with frequencies.
a,b	the lower and upper bounds of the rounding error.
from,to,gridsize	start point, end point and size of a fine grid where the distribution will be evaluated.
method	type estimate: "mle" or "Berkson", or "boot" (default).

Value

y	Estimated values of the smooth function over a fine grid.
x	grid points where the smoothed function are evaluated.
pars	return 'bw' for Berkson method, or return the mean and SD for MLE method

References

Wang, B and Wertelecki, W, (2013) Computational Statistics and Data Analysis, 65: 4-12.

Examples

```
#data(ofc)
#x0 = round(ofc$Head)
x0 = round(rnorm(100,34.5,1.6))
fx1 = bootkde(x0,a=-0.5,b=0.5,method="Berkson")
```

bootPRO *Effectiveness Evaluation based on PROs with bootstrapping method*

Description

Effectiveness Evaluation based on PROs with bootstrapping method.

Usage

```
bootPRO(x,type="relative",MCID,iter=999,conf.level=0.95)
```

Arguments

x	Data frame of repeated PRO measures. First column gives the treatment groups: 'group' or 'treat'; Repeated measures listed after column 1.
type	use absolute changes ("absolute") or relative (percent) changes ("relative").
MCID	A positive value to define responders.
iter	number of iterations used by the bootstrap algorithm.
conf.level	Confidence level of the bootstrapping confidence interval.

Value

NONE.

References

To be updated.

Examples

```
data(Pain)
x <- pain[,c(2,3:9,11:17)]
grp <- rep("treat",nrow(x))
grp[x[,1]==0] <- "control"
x[,1] <- grp
#bootPRO(x,type='mean')
#bootPRO(x,type='mean',MCID=1)
#bootPRO(x,type='mean',MCID=1.5)
#bootPRO(x,type='mean',MCID=2)

#bootPRO(x,type='rel')
#bootPRO(x,type='relative',MCID=.2)
#bootPRO(x,type='relative',MCID=.3)
##bootPRO(x,type='relative',MCID=.5)
```

BreastCancer *Breast Cancer Data*

Description

Cleaned breast cancer data from TCGA. Three datasets: primary, normal and meta.

Usage

```
data(BreastCancer)
```

Value

None

References

TCGA(2012) Comprehensive molecular portraits of human breast tumours. Nature,490(7418),61-70.

Examples

```
data(BreastCancer)
head(normal)
head(primary)
head(meta)
```

EUFirmSize *Firm size data of 10 EU countries*

Description

Mining and quarrying firm employment sizes. Classes: "0-9","10-19","20-49","50-249","250+".

Usage

```
data(EUFirmSize)
```

Value

Class	Firm size classes
Year20xx	Firm size data of 10 EU countries for year 20xx.

References

Eurostat, Enterprises in Europe, Data 1994-95, fifth report Edition, European Commission, Brussels, 1998.

Examples

```
data(EUFirmSize)
head(EUFirmSize)
```

`fit.FSD` *Fitting firm size-age distributions*

Description

To fit firm size and/or age distributions based on binned data.

Usage

```
fit.FSD(x,breaks,dist)
```

Arguments

`x`	'x' can be a vector or a matrix, or a 'histogram'.
`breaks`	a matrix with two columns if 'x' is a matrix. Otherwise, it is a vector. Can be missing if 'x' is a vector and 'x' will be grouped uisng the default parameters with `hist`.
`dist`	distribution type for 'x' and/or 'y'. Options include `Weibull`, `gpd` – generalized Pareto distribution, `pd` or `Pareto`, `EWD`– exponentiated Weibull distribtion.

Value

`x.fit, y.fit`	Fitted marginal distribution for row- and column-data when 'x' is a matrix.
`Psi`	Plackett estimate of the Psi. ONLY for 'fit.FSD2'

If each of `x.fit` and `y.fit`, the following values are available:

`xhist`	histogram.
`dist`	distribution type. Options include `Weibull`, `gpd` – generalized Pareto distribution, `pd` or `Pareto`, `EWD`– exponentiated Weibull distribtion.
`size`	Total number of observations (sample size)
`pars`	Estimates of parameters.
`y,y2,x`	the pdf (`y`) and cdf (`y2`) values evaluated on a grid `x`

Examples

```
x <- rweibull(1000,2,1)
(out <- fit.FSD(x))

data(FSD)
b <- c(0,1:5,6,11,16,21,26,38,Inf);
x <- as.numeric(FirmAge[38,]);
## not run
##(out <- fit.FSD(x))#treated as raw data
xh <- binning(counts=x, breaks=b)
(out <- fit.FSD(xh))
#(out <- fit.FSD(xh,dist="ewd"))
(out <- fit.FSD(xh,dist="pd"))
(out <- fit.FSD(xh,dist="gpd"))


x <- as.numeric(FirmSize[nrow(FirmSize),])
brks.size <- c(0,4.5,9.5,19.5,49.5,99.5,249.5,499.5,
               999.5,2499.5,4999.5, 9999.5,Inf)
xh <- binning(counts=x,breaks=brks.size)
Fn <- cumsum(x)/sum(x);Fn
k <- length(Fn)
i <- c((k-5):(k-1))
out1 <- fit.GLD(xh, qtl=brks.size[i+1],
               qtl.levels=Fn[i],lbound=0)

i <- c(2,3,4,10,11)
out2 <- fit.GLD(xh, qtl=brks.size[i+1],
               qtl.levels=Fn[i],lbound=0)

i <- c(1,2,8,9,10)
out3 <- fit.GLD(xh, qtl=brks.size[i+1],
               qtl.levels=Fn[i],lbound=0)

plot(xh,xlim=c(0,120))
lines(out1, col=2)
lines(out2, col=3)
lines(out3, col=4)

ZipfPlot(xh,plot=TRUE)
lines(log(1-out1$y2)~log(out1$x), col=2)
lines(log(1-out2$y2)~log(out2$x), col=3)
lines(log(1-out3$y2)~log(out3$x), col=4)

## sample codes for the figures and tables in the PLoS ONE manuscript

## Table 1 ************************************************
#xtable(Firm2)

## Figure 1 ***********************************************

#rm(list=ls())
```

```
#require(bda)
#data(FSD)

##postscript(file='fig1.eps',paper='letter')
#par(mfrow=c(2,2))

#tmp <- ImportFSD(FirmAge, year=2014,type="age");
#xh <- tmp$age
#plot(xh, xlab="X", main="(a) Histogram of 2014 firm age data")
#fit0 <- fit.FSD(xh, dist='exp')
#lines(fit0$x.fit$ly~fit0$x.fit$lx, lty=2)

#ZipfPlot(fit0, lty=2,
#         xlab="log(b)",ylab="log(r)",
#         main="(b) Zipf plot of firm age (2014)")

#tmp <- ImportFSD(FirmAge, year=1988,type="age");
#xh2 <- tmp$age
#fit1 <- fit.FSD(xh2, dist='exp')
#ZipfPlot(fit1, lty=2,
#         xlab="log(b)",ylab="log(r)",
#         main="(c) Zipf plot of firm age (1988)")

#ZipfPlot(fit0, lty=2, type='l',
#         xlab="log(b)",ylab="log(r)",
#         main="(d) Zipf plots of firm age (1979-2014)")
#for(year in 1979:2014){
#    tmp <- ImportFSD(FirmAge, year=year,type="age")
#    tmp2 <- ZipfPlot(tmp$age, plot=FALSE)
#    lines(tmp2)
#}

#dev.off()

## Figure 2 ***************************************************

#rm(list=ls())
#require(bda)
#data(FSD)

#postscript(file='fig2.eps',paper='letter')
#par(mfrow=c(2,2))
## plot (a)
#tmp <- ImportFSD(FirmSize, year=2014,type="size");
#yh <- tmp$size
#plot(yh, xlab="Y", main="(a) Histogram of firm size (2014)")

## plot (b)
#lbrks <- log(yh$breaks); lbrks[1] <- log(1)
#cnts <- yh$freq
#xhist2 <- binning(counts=cnts,breaks=lbrks)
#plot(xhist2,xlab="log(Y)",
#     main="(b) Histogram of firm size (2014, log-scale)")
```

```
## plot (c)
#ZipfPlot(yh, plot=TRUE,
#         main="(c) Zipf plot firm size (2014)",
#         xlab="log(r)",ylab="log(b)")

## plot (d)

#ZipfPlot(yh, plot=TRUE,type='l',
#         main="(d) Zipf plots of firm size (1977-2014)",
#         xlab="log(r)",ylab="log(b)")

#res <- NULL
#for(year in 1977:2014){
#    tmp <- ImportFSD(FirmSize,year=year,type="size");
#    yh0 <- tmp$size
#    zipf1 <- ZipfPlot(yh0,plot.new=FALSE)
#    lines(zipf1)
#    res <- c(res, zipf1$slope)
#}

#dev.off()

#mean(res); sd(res)
#quantile(res, prob=c(0.025,0.097))


## Figure 3a & 3b ****************************************************
#rm(list=ls())
#require(bda)
#data(FSD)

#postscript(file='fig3a.eps',paper='letter')

tmp <- ImportFSD(FirmAge, year=2014,type="age");
xh <- tmp$age
(fit1 <- fit.FSD(xh, dist='exp'));
(fit2 <- fit.FSD(xh, dist='weibull'));
#(fit3 <- fit.FSD(xh, dist='ewd')); #slow
#(fit0 <- fit.FSD(xh, dist='gpd')); # test, not used
(fit4 <- fit.FSD(xh, dist='gld'));

#plot(xh, xlab="X", main="(a) Fitted Distribtions")
#lines(fit1, lty=1, col=1,lwd=3)
#lines(fit2, lty=2, col=1,lwd=3)
#lines(fit3, lty=3, col=1,lwd=3)
#lines(fit4, lty=4, col=1,lwd=3)
#lines(fit0, lty=4, col=2,lwd=3)

legend("topright",cex=2,
       legend=c("EXP","Weibull","EWD","GLD"),
       lty=c(1:4),lwd=rep(3,4),col=rep(1,4))
dev.off()
```

```
## Table 3
#r2 <- c(fit1$x.fit$Dn.Zipf, fit2$x.fit$Dn.Zipf,
#        fit3$x.fit$Dn.Zipf, fit4$x.fit$Dn.Zipf)
#dn <- c(fit1$x.fit$Dn, fit2$x.fit$Dn,
#        fit3$x.fit$Dn, fit4$x.fit$Dn)
#aic <- c(fit1$x.fit$AIC, fit2$x.fit$AIC,
#         fit3$x.fit$AIC, fit4$x.fit$AIC)
#bic <- c(fit1$x.fit$BIC, fit2$x.fit$BIC,
#         fit3$x.fit$BIC, fit4$x.fit$BIC)
#aicc <- c(fit1$x.fit$AICc, fit2$x.fit$AICc,
#          fit3$x.fit$AICc, fit4$x.fit$AICc)
#tbl3 <- data.frame(
#    Dn.Zipf = r2, Dn=dn, AIC=aic, BIC=bic,AICc=aicc)
#rownames(tbl3) <- c("EXP","WD","EWD","GLD")
##save(tbl3, file='tbl3.Rdata')
#tbl3
#require(xtable)
#xtable(tbl3,digits=c(0,4,4,0,0,0))


#postscript(file='fig3b.eps',paper='letter')
#par(mfrow=c(1,1))
#ZipfPlot(fit1, plot.new=TRUE,col=1,lwd=3,lty=1,
#         xlab="log(x)",ylab="log(S(x))",
#         main="(b) Zipf Plots of Fitted Distribtions")
#ZipfPlot(fit2, plot.new=FALSE,col=1,lty=2,lwd=3)
#ZipfPlot(fit3, plot.new=FALSE,col=1,lty=3,lwd=3)
#ZipfPlot(fit4, plot.new=FALSE,col=1,lty=4,lwd=3)

#legend("bottomleft",cex=2,
#       legend=c("EXP","Weibull","EWD","GLD"),
#       lty=c(1:4),lwd=rep(3,4),col=rep(1,4))
#dev.off()

## TABLE 4 ##########################################
## More Results ************************************

#mysummary <- function(x,dist){
#    .winner <- function(x,DIST=dist){
#        isele <- which(x==min(x))
#        if(length(isele)>1)
#            warning("multiple winners, only the first is used")
#        DIST[isele[1]]
#    }
#    .mysum <- function(x,y) sum(y==x)
#    mu1 <- tapply(x$Dn.Zipf,x$Dist, mean, na.rm=TRUE)
#    mu2 <- tapply(x$Dn,x$Dist, mean, na.rm=TRUE)
#    sd1 <- tapply(x$Dn.Zipf,x$Dist, sd, na.rm=TRUE)
#    sd2 <- tapply(x$Dn,x$Dist, sd, na.rm=TRUE)
#    win1 <- tapply(x$Dn.Zipf,x$Year, .winner,DIST=dist)
#    win2 <- tapply(x$Dn,x$Year, .winner,DIST=dist)
#    win3 <- tapply(x$BIC,x$Year, .winner,DIST=dist)
```

```
#    dist0 <- levels(as.factor(x$Dist))

#    out1 <- sapply(dist0, .mysum,y=win1)
#    out2 <- sapply(dist0, .mysum,y=win2)
#    out3 <- sapply(dist0, .mysum,y=win3)
#    #sele1 <- match(names(out1), dist0)
#    #sele2 <- match(names(out2), dist0)
#    #sele3 <- match(names(out3), dist0)

#    out <- data.frame(Mean.Dn.Zipf=mu1, SD.Dn.Zipf=sd1,
#                      Mean.Dn=mu2, SD.Dn=sd2,
#                      Win.Zipf=out1,
#                      Win.Dn=out2,
#                      Win.BIC=out3)
#    out
#}

#require(bda) #version 14.3.11+
#data(FSD)

#Dn.Zipf <- NULL
#Dn <- NULL
#BIC <- NULL
#Year <- NULL
#DIST <- NULL; dist0 <- c("EXP","WD","EWD","GLD")
#for(year in 1983:2014){
#    tmp <- ImportFSD(FirmAge, year=year,type="age");
#    xh <- tmp$age
#    fit1 <- fit.FSD(xh, dist='exp');fit1
#    DIST <- c(DIST,"EXP")
#    Year <- c(Year, year)
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='weibull');fit1
#    DIST <- c(DIST,"WD")
#    Year <- c(Year, year)
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='ewd');fit1
#    DIST <- c(DIST,"EWD")
#    Year <- c(Year, year)
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='gld');fit1
#    DIST <- c(DIST,"GLD")
#    Year <- c(Year, year)
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#}
```

```
#RES <- data.frame(Year=Year,Dist=DIST,Dn.Zipf=Dn.Zipf,Dn=Dn,BIC=BIC)
#save(RES, file='tbl4.Rdata')
#load(file='tbl4.Rdata')

#DIST <- c("EXP","WD","EWD","GLD")
#sele <- RES$Year>=1983 & RES$Year<=1987;sum(sele)
#(out <- mysummary(RES[sele,],dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

#sele <- RES$Year>=1988 & RES$Year<=1992;sum(sele)
#(out <- mysummary(RES[sele,],dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

#sele <- RES$Year>=1993 & RES$Year<=1997;sum(sele)
#(out <- mysummary(RES[sele,],dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

#sele <- RES$Year>=1998 & RES$Year<=2002;sum(sele)
#(out <- mysummary(RES[sele,],dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

#sele <- RES$Year>=2003 & RES$Year<=2014;sum(sele)
#(out <- mysummary(RES[sele,],dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

#(out <- mysummary(RES,dist=DIST))
#xtable(out[c(2,4,1,3),],digits=c(0,4,4,4,4,0,0,0))

## Figure 4a & 4b ****************************************
#rm(list=ls())
#require(bda)
#data(FSD)


tmp <- ImportFSD(FirmSize, year=2014,type="size");
yh <- tmp$size

(fit1 <- fit.FSD(yh, dist='lognormal'));
(fit2 <- fit.FSD(yh, dist='pareto'));
(fit3 <- fit.FSD(yh, dist='gpd'));
(fit4 <- fit.FSD(yh, dist='gld'));

#postscript(file='fig4a.eps',paper='letter')
#par(mfrow=c(1,1))
#plot(yh, xlab="Y", main="(a) Fitted Distributions",
#     xlim=c(0,50))
#lines(fit1, lty=1, col=1,lwd=3)
#lines(fit2, lty=2, col=1,lwd=3)
#lines(fit3, lty=3, col=1,lwd=3)
#lines(fit4, lty=4, col=1,lwd=3)
```

```
#legend("topright",cex=2,
#       legend=c("LN","PD","GPD","GLD"),
#       lty=c(1:4),lwd=rep(3,4),col=rep(1,4))
#dev.off()

#postscript(file='fig4b.eps',paper='letter')
#par(mfrow=c(1,1))
#ZipfPlot(fit1, plot.new=TRUE,col=1,lwd=3,lty=1,
#         xlab="log(y)",ylab="log(S(y))",
#         main="(b) Zipf Plots of Fitted Distribtions")
#ZipfPlot(fit2, plot.new=FALSE,col=1,lty=2,lwd=3)
#ZipfPlot(fit3, plot.new=FALSE,col=1,lty=3,lwd=3)
#ZipfPlot(fit4, plot.new=FALSE,col=1,lty=4,lwd=3)

#legend("bottomleft",cex=2,
#       legend=c("LN","PD","GPD","GLD"),
#       lty=c(1:4),lwd=rep(3,4),col=rep(1,4))
#dev.off()

## TABLE 5 ##############################################
## More information about firm size


#Dn.Zipf <- NULL
#Dn <- NULL
#BIC <- NULL
#Year <- NULL
#DIST <- NULL;

#dist0 <- c("LN","PD","GPD","GLD")
#for(year in 1977:2014){
#    DIST <- c(DIST,dist0)
#    Year <- c(Year, rep(year,length(dist0)))
#    tmp <- ImportFSD(FirmSize, year=year,type="size");
#    xh <- tmp$size
#    fit1 <- fit.FSD(xh, dist='lognormal');
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='pd');
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='gpd');
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#    fit1 <- fit.FSD(xh, dist='gld');
#    Dn.Zipf <- c(Dn.Zipf, fit1$x.fit$Dn.Zipf)
#    Dn <- c(Dn, fit1$x.fit$Dn)
#    BIC <- c(BIC, fit1$x.fit$BIC)
#}
```

```
#RES <- data.frame(Year=Year,Dist=DIST,Dn.Zipf=Dn.Zipf,Dn=Dn,BIC=BIC)
#save(RES, file='tbl5.Rdata')
#load(file='tbl5.Rdata')

#dist0 <- c("LN","PD","GPD","GLD")
#(out <- mysummary(RES,dist=dist0))
#require(xtable)
#xtable(out[c(3,2,1),],digits=c(0,4,4,4,4,0,0,0))


## Figure 5  ***************************************************

#xy <- tmp <- ImportFSD(Firm2);
#out1 <- fit.FSD(xy$xy, breaks=xy$breaks,dist=c("EWD","GPD"));out1
#out2 <- fit.FSD(xy$xy, breaks=xy$breaks,dist=c("EWD","GLD"));out2
#out3 <- fit.FSD(xy$xy, breaks=xy$breaks,dist=c("GLD","GPD"));out3
#out4 <- fit.FSD(xy$xy, breaks=xy$breaks,dist=c("GLD","GLD"));out4

#postscript(file="fig5.eps",paper='letter')

#par(mfrow=c(2,2))
#out <- out1
#res2 <- plot(out,grid.size=40,nlevels=30,
#             ylim=c(0,15),xlim=c(0,30),
#             xlab="Firm Age",ylab="Firm Size",
#             main="(a) Contour Plot -- (EWD+GPD)")

#out <- out2
#res2 <- plot(out,grid.size=40,nlevels=30,
#             ylim=c(0,15),xlim=c(0,30),
#             xlab="Firm Age",ylab="Firm Size",
#             main="(b) Contour Plot -- (EWD+GLD)")
#out <- out3
#res2 <- plot(out,grid.size=40,nlevels=30,
#             ylim=c(0,15),xlim=c(0,30),
#             xlab="Firm Age",ylab="Firm Size",
#             main="(c) Contour Plot -- (GLD+GPD)")
#
#out <- out4
#res2 <- plot(out,grid.size=40,nlevels=30,
#             ylim=c(0,15),xlim=c(0,30),
#             xlab="Firm Age",ylab="Firm Size",
#             main="(d) Contour Plot -- (GLD+GLD)")
##dev.off()


## Figure 6  ***************************************************
#xy2 <- tmp <- ImportFSD(Firm2);

## this is an example showing how to use partial data to fit FSD.
## get marginal frequency distribution for firm age:
#(X <- apply(xy2$xy,1,sum));
#brks.age <- c(0,1,2,3,4,5,6,11,16,21,26,38,Inf)
```

```
## get marginal frequency distribution for firm size:
#(Y <- apply(xy2$xy,2,sum));
#(Y <- Y[-1])
#brks.size <- c(5,10,20,50,100,250,500,1000,2500,5000,10000,Inf)
#mxy2 <- xy2$xy[,-1]

#(fitx1 <- fit.FSD(X, breaks=brks.age, dist="ewd"))
#(fitx2 <- fit.FSD(X, breaks=brks.age, dist="gld"))
#(fity1 <- fit.FSD(Y, breaks=brks.size, dist="gld"))
#(fity2 <- fit.FSD(Y, breaks=brks.size, dist="pd"))
#(fity3 <- fit.FSD(Y, breaks=brks.size, dist="gpd"))

#(out11 <- fit.Copula(fitx1, fity1, mxy2))
#(out12 <- fit.Copula(fitx1, fity2, mxy2))
#(out21 <- fit.Copula(fitx2, fity1, mxy2))
#(out22 <- fit.Copula(fitx2, fity2, mxy2))
#(out13 <- fit.Copula(fitx1, fity3, mxy2))
#(out23 <- fit.Copula(fitx2, fity3, mxy2))

##postscript(file="fig6.eps",paper='letter')

#par(mfrow=c(1,1))
#plot(out11,grid.size=40,nlevels=20,lty=2,
#     ylim=c(4.8,16),xlim=c(0,18),
#     xlab="Firm Age",ylab="Firm Size",
#     main="(d) Contour Plot -- (GLD+GLD)")
#plot(out12,grid.size=40,nlevels=30, col=2, plot.new=FALSE)
## or use the command below
## plot(out2,grid.size=50,nlevels=50, col=4, add=TRUE)
##dev.off()
```

`fit.GBP` *Fitting Mixture Model of Generalized Beta and Pareto*

Description

To fit a mixture model of generalize beta and Pareto to grouped data.

Usage

```
fit.GBP(x,breaks)
```

Arguments

`x`	'x' can be a vector or a matrix, or a 'histogram'.
`breaks`	a matrix with two columns if 'x' is a matrix. Otherwise, it is a vector. Can be missing if 'x' is a vector and 'x' will be grouped uisng the default parameters with `hist`.

Value

pars	estimated parameters.

Examples

```
data(FSD)
x <- as.numeric(FirmSize[nrow(FirmSize),])
brks.size <- c(0,4.5,9.5,19.5,49.5,99.5,249.5,499.5,
               999.5,2499.5,4999.5, 9999.5,Inf)
xhist1 <- binning(counts=x,breaks=brks.size)

(out <- fit.GBP(x,brks.size))
(out <- fit.GBP(xhist1))

plot(xhist1,xlim=c(0,110))
x0 <- seq(0,110,length=1000)
f0 <- dGBP(x0,out)
lines(f0~x0, col=2,lwd=2)

ZipfPlot(xhist1,plot=TRUE)
F0 <- pGBP(brks.size,out)
lines(log(1-F0)~log(brks.size), col=2)
```

---

fit.lognormal *Fitting log-normal distributions*

Description

To fit log-normal distributions to raw data.

Usage

```
fit.lognormal(x, k=1,normal=FALSE)
```

Arguments

x	Raw data or grouped data
k	number of components, Default: 1
normal	Fit normal mixture models if 'normal=TRUE'; otherwise fit log-normal mixture models.

Value

p0	The estimated proportion of zeros.
p,mean,sigma	The fitted parameters of mixing coefficients, means and standard deviations of the k normal components.
n	The sample size of data.
npar	Number of parameters to be estimated.
llk	Estimated log-likelihood.

Examples

```
mu = -.5
s = 2
```

`fit.Pareto` *Fit a Pareto Distribution to Binned Data*

Description

Fit a Pareto distribution to binned data.

Usage

```
fit.Pareto(x, xm, method='mle')
```

Arguments

`x`	grouped data
`xm`	The location parameter: lower bound of the support of the distribution
`method`	fitting method: 'mle'=maximum likelihood estimate, 'percentile'=percentile matching.

Value

`xm`	fitted location parameter
`alpha`	fitted scale parameter

Examples

```
xm <- 0.5
alpha <- 1.0

x <- rPareto(1000, xm, alpha)
(out <- fit.Pareto(x,method='mle'))
(out <- fit.Pareto(x,method='ls'))

xbrks <- c(0,4.5,9.5,19.5,49.5,99.5,249.5,499.5,999.5,
           2499.5,4999.5,9999.5,Inf)
xhist <- binning(x, breaks=xbrks)

(out <- fit.Pareto(xhist))
(out <- fit.Pareto(xhist,method='mle'))
(out <- fit.Pareto(xhist,method='ls'))
(out <- fit.Pareto(xhist,xm=.5,method='mle'))
```

`fit.PRO` *Fitting distributions to Patient-reported Outcome Data*

Description

To fit PRO data to distributions including GLD, PD, GPD, Weibull, EWD and other families.

Usage

```
fit.PRO(x,dist,x.range,nclass)
```

Arguments

`x`	'x' can be a vector or a matrix, a 'histogram', or binned data.
`dist`	distribution type for 'x' and/or 'y'. Options include `Weibull`, `gpd` – generalized Pareto distribution, `pd` or `Pareto`, `EWD`– exponentiated Weibull distribtion.
`x.range`	Specifies the range of data. Used only for interval PRO data.
`nclass`	Number of classes/bins. Used only for interval PRO data.

Value

`x.fit, y.fit`	Fitted marginal distribution for row- and column-data when 'x' is a matrix.
`Psi`	Plackett estimate of the Psi. ONLY for 'fit.FSD2'

If each of `x.fit` and `y.fit`, the following values are available:

`xhist`	histogram.
`dist`	distribution type. Options include `Weibull`, `gpd` – generalized Pareto distribution, `pd` or `Pareto`, `EWD`– exponentiated Weibull distribtion.
`size`	Total number of observations (sample size)
`pars`	Estimates of parameters.
`y,y2,x`	the pdf (`y`) and cdf (`y2`) values evaluated on a grid `x`

Examples

```
x <- rweibull(1000,2,1)
(out <- fit.PRO(x))
```

fnm *Distribution of Two Finite Gaussian Mixtures*

Description

To compute the values of the density and distribution functions of two finite Gaussian mixture models.

Usage

```
fnm(p1,p2,mu1,mu2,sig1,sig2,from,to)
```

Arguments

p1,p2	mixing coefficients.
mu1,mu2	vectors of the mean values of the Gaussian components.
sig1,sig2	vectors of the SD values of the Gaussian components.
from,to	to specify the range of data.

Value

Return the densities ('y') and probabilities ('Fx') over a grid of 'x'.

Examples

```
data(Pain)

group <- pain$treat
x <- pain$recall0
y <- pain$recall1
#out <- tkde(x,y,group)
out <- tkde(x,y,group,type='percent')
plot(out$risk,type='l')
abline(h=1,col='gray')

plot(out$responder,type='l',ylim=c(-.28,.1))
lines(out$resp$ll~out$resp$x,lty=2,col=1+(out$resp$p<0.05))
lines(out$resp$ul~out$resp$x,lty=2,col=1+(out$resp$p<0.05))
abline(h=0,col='gray')

plot(out$g2,type='l')
lines(out$g1,col=2)
```

FSD *Firm size data*

Description

2014 US Private-sector firm size data.

References

https://www.sba.gov/advocacy/firm-size-data, Accessed on 2018-11-16

ImportFSD *Import Firm Size and Firm Age Data*

Description

To read firm size and/or firm age data from built-in datasets.

Usage

```
ImportFSD(x,type,year)
```

Arguments

x	A built-in firm size and/or firm age dataset.
type	type of data: "size" or "age". If missing, read the age data from rows and the size data from columns.
year	The number year the firm size/age data to be read. If missing, assume the year is 2014.

Value

xy	a matrix of joint frequency distribution table.
breaks	the class boundaries for firm age as a component age, and for firm size as a component size.
size	a 'bdata' subject of the firm size data
age	a 'bdata' object of the firm age data

Examples

```
data(FSD)
## bivariate data
xy = ImportFSD(Firm2)
## firm age of 2013
x = ImportFSD(FirmAge, type="age", year=2013)
## firm size of 2013
y = ImportFSD(FirmSize, type="size", year=2013)
```

`lps.variance` *compute the variance of the local polynomial regression function*

Description

To compute the variance of the local polynomial regression function

Usage

```
lps.variance(y,x,bw, method="Rice")
```

Arguments

`y,x`	Two numerical vectors: `y` is the response and `x` is the predictor.
`bw`	Smoothing parameter. Is used only when `method='Wasserman'` or `method='heteroscedastic'`.
`method`	We use four method to compute the variance of r(x): Method 1) Larry Wasserman– nearly unbiased. This method based on an lps object; Method 2) Rice 1984 Method 3) Gasser et al (1986) – a variation of method 3. Method 4) For heteroscedastic errors. Need to estimate based on an lpr object. Yu and Jones (2004). Defaulty method: `Rice`.

Value

the variance of r(x).

Examples

```
n = 100
x=rnorm(n)
y=x^2+rnorm(n)
bw = lps.variance
par(mfrow=c(1,1))
out=lpsmooth(y,x)
#plot(out, scb=TRUE, type='l')
vrx = lps.variance(y,x)
out=lpsmooth(y,x,sd.y=sqrt(vrx), bw=0.5)
```

```
plot(y~x, pch='.')
lines(out, col=2)

x0 = seq(min(x),  max(x), length=100)
y0 = x0^2
lines(y0~x0, col=4)
```

---

lpsmooth *non-parametric regression*

Description

To fit nonparametric regression model.

Usage

```
lpsmooth(y,x, bw, sd.y,lscv=FALSE, adaptive=FALSE,
   from, to, gridsize,conf.level=0.95)
npr(y,x,sd.x,bw,kernel='decon',optimal=FALSE,adaptive=FALSE,
    x0,from, to, gridsize,conf.level=0.95)
wlpsmooth(y,x,w,s.x,bw,from,to,gridsize,conf.level=0.95)
bootsmooth(y,x,type="relative",iter=100,conf.level=0.95)
```

Arguments

y,x	Two numerical vectors.
w	weights
s.x	standard deviation of the measurement error – Laplacian errors are assumed.
x0,from,to,gridsize	'x0' is the grid points where the fitted values will be evaluated. If it is missing, define a fine grid using the start point ("from"), end point ("to") and size ("gridsize").
bw	Smoothing parameter. Numeric or character value is allowed. If missing, adaptive (LSCV) bandwidth selector will be used.
kernel	kernel type: "normal","gauss","nw","decon" (default), "lp","nadaraya-watson"
lscv,adaptive	If lscv = FALSE, use the given bandwidth to fit lpr directly. If lscv = TRUE and adaptive = FALSE, compute lscv bandwidth and fit lpr. Initial bandwidth should be given. If lscv = TRUE and adaptive = TURE, compute lscv bandwidth, then compute varying smoothing parameter, then fit lpr. This algorithm could be extremeely slow when the sample size is very large.
optimal	Search for optimal bandwidth if TRUE.
sd.y	Standard deviation of y.
sd.x	Standard deviation of the measurement error x.

conf.level	Confidence level.
iter	Bootstrapping iteration number.
type	"relative" changes or "absolute" changes for effectiveness evaluation.

Value

y	Estimated values of the smooth function over a fine grid.
x	grid points where the smoothed function are evaluated.
x0,y0	cleaned data of x and y.
conf.level	confidence level of the simultaneous confidence bands.
pars	estimate parameters including smoothing bandwidth, and parameters for the tube formula.
ucb,lcb	upper and lower confidence bands.
call	function called

Examples

```
x <- rnorm(100,34.5,1.5)
e <- rnorm(100,0,2)
y <- (x-32)^2 + e
out <- lpsmooth(y,x)
out
plot(out, type='l')
x0 <- seq(min(x),max(x),length=100)
y0 <- (x0-32)^2
lines(x0, y0, col=2)
points(x, y, pch="*", col=4)
```

---

mediation.test *The Sobel mediation test*

Description

To compute statistics and p-values for the Sobel test. Results for three versions of "Sobel test" are provided: Sobel test, Aroian test and Goodman test.

Usage

```
mediation.test(mv,iv,dv)
```

Arguments

mv	The mediator variable.
iv	The independent variable.
dv	The dependent variable.

Details

To test whether a mediator carries the influence on an IV to a DV. Missing values will be automatically excluded with a warning.

Value

a table showing the values of the test statistics (z-values) and the corresponding p-values for three tests, namely the Sobel test, Aroian test and Goodman test, respectively.

Author(s)

B. Wang <bwang@southalabama.edu>

References

MacKinnon, D. P., & Dwyer, J. H. (1993). Estimating mediated effects in prevention studies. *Evaluation Review*, 17, 144-158.

MacKinnon, D. P., Warsi, G., & Dwyer, J. H. (1995). A simulation study of mediated effect measures. *Multivariate Behavioral Research*, 30, 41-62.

Preacher, K. J., & Hayes, A. F. (2004). SPSS and SAS procedures for estimating indirect effects in simple mediation models. *Behavior Research Methods,Instruments, & Computers*, 36, 717-731.

Preacher, K. J., & Hayes, A. F. (2008). asymptotic and resampling strategies for assessing and comparing indirect effects in multiple mediator models. *Behavior Research Methods, Instruments, & Computers*, 40, 879-891.

Examples

```
mv = rnorm(100)
iv = rnorm(100)
dv = rnorm(100)
mediation.test(mv,iv,dv)
```

`mlnorm` *The mixed lognormal distribution*

Description

Density, distribution function, quantile function and random generation for the lognormal mixture distribution with means equal to 'mu' and standard deviations equal to 's'.

Usage

```
dmlnorm(x,p,mean,sd)
pmlnorm(q,p,mean,sd)
qmlnorm(prob,p,mean,sd)
rmlnorm(n,p,mean,sd)
```

Arguments

`x,q`	vector of quantiles in dmixnorm and pmixnorm. In qmixnorm, 'x' is a vector of probabilities.
`p`	proportions of the mixture components.
`prob`	A vector of probabilities.
`n`	number of observations. If 'length(n) > 1', the length is taken to be the number required.
`mean`	vector of means
`sd`	vector of standard deviations

Value

return the density, probability, quantile and random value for the four functions, respectively.

Examples

```
p <- c(.4,.6)
mu <- c(1,4)
s <- c(2,3)
dmlnorm(c(0,1,2,20),p,mu,s)
pmlnorm(c(0,1,2,20),p,mu,s)
qmlnorm(c(0,1,.2,.20),p,mu,s)
rmlnorm(3,p,mu,s)
```

mnorm *The mixed normal distribution*

Description

Density, distribution function, quantile function and random generation for the normal mixture distribution with means equal to 'mu' and standard deviations equal to 's'.

Usage

```
dmnorm(x,p,mean,sd)
pmnorm(q,p,mean,sd)
qmnorm(prob,p,mean,sd)
rmnorm(n,p,mean,sd)
```

Arguments

x,q	vector of quantiles in dmixnorm and pmixnorm. In qmixnorm, 'x' is a vector of probabilities.
p	proportions of the mixture components.
prob	A vector of probabilities.
n	number of observations. If 'length(n) > 1', the length is taken to be the number required.
mean	vector of means
sd	vector of standard deviations

Value

Return the density, probability, quantile and random value, respectively.

Examples

```
p <- c(.4,.6)
mu <- c(1,4)
s <- c(2,3)
dmnorm(c(0,1,2,20),p,mu,s)
pmnorm(c(0,1,2,20),p,mu,s)
qmnorm(c(0,1,.2,.20),p,mu,s)
rmnorm(3,p,mu,s)
```

ofc *occipitofrontal head circumference data*

Description

OFC data for singleton live births with gestational age at least 38 weeks.

Format

A data frame with 2019 observations on 4 variables.

Year	numeric	2006 – 2009
Sex	character	'male' or 'female'
Gestation	numeric	Gestational age (in weeks).
Head	numeric	head size.

References

Wang, B and Wertelecki, W, (2013) Computational Statistics and Data Analysis, 65: 4-12.

Examples

```
data(ofc)
head(ofc)
```

Pain *Pain data*

Description

Pain data using VAS.

Format

A data frame with 203 records on 17 variables.

group	character	'control' or 'treatment' group
t01-t07	numeric	VAS measures at T0 from diary
t11-t17	numeric	VAS measures at T1 from diary
rvas0	numeric	recalled VAS average at T0
rvas1	numeric	recalled VAS average at T1

References

To be updated

Pareto *The Pareto distribution*

Description

Density, distribution function, quantile function and random generation for the Pareto distribution.

Usage

```
dPareto(x,xm,alpha)
pPareto(q,xm,alpha)
qPareto(p,xm,alpha)
rPareto(n,xm,alpha)
```

Arguments

x,q	vector of quantiles in dmixnorm and pmixnorm. In qmixnorm, 'x' is a vector of probabilities.
p	A vector of probabilities.
n	number of observations. If 'length(n) > 1', the length is taken to be the number required.
xm,alpha	parameters of the Pareto distribution.

Value

NONE

Examples

```
xm = 0.1
alpha = 1
dPareto(.5, xm,alpha)
```

pro.test *Test effectiveness based on PROs*

Description

Tests for effectiveness evaluations based on PROs.

Usage

```
pro.test(x,y,group,cutoff,x.range,type)
```

Arguments

x,y	vector of PROs at T0 and T1.
group	Group assignment: control or treatment.
cutoff	Class boundaries to define states. Works only when 'x' and 'y' are numeric.
x.range	Range of the scores for 'x' and 'y'.
type	Data (grouping/binning) type: 'vas', 'nrs', 'wbf'.

Details

To be added.

Value

To be added.

References

To be added.

Examples

```
states <- c("low", "moderate", "high")
x0 <- sample(states, size=100, replace=TRUE)
x1 <- sample(states, size=100, replace=TRUE)
grp <- c(rep("control",50),rep("treatment",50))
pro.test(x=x0,y=x1,group=grp)
```

VAS *Algorithms for Visual Analogue Scales*

Description

Algorithms for VAS. The algorithms are applicable to other numerical variables with measurement errors as well.

Usage

```
VAS.ecdf(x,w,alpha=0.05)
```

Arguments

x	Raw data
w	weights
alpha	Significance level for confidence bands.

Value

Estimate of the emprical distribution function.

x	grid points
y	ECDF value Fn(x)
lb,ub	lower and upper confidence bands of ECDF.
alpha	significance level
data	raw data
ecdf	Draw ECDF if TRUE.

Examples

```
x <- rnorm(100, -2.6, 3.1)
```

---

wkde *Compute a Binned Kernel Density Estimate for Weighted Data*

Description

Returns x and y coordinates of the binned kernel density estimate of the probability density of the weighted data.

Usage

```
wkde(x, w, bandwidth, freq=FALSE, gridsize = 401L, range.x,
  truncate = TRUE, na.rm = TRUE)
```

Arguments

x	vector of observations from the distribution whose density is to be estimated. Missing values are not allowed.
w	The weights of x. The weight w_i of any observation x_i should be non-negative. If x_i=0, x_i will be removed from the analysis.
bandwidth	the kernel bandwidth smoothing parameter. Larger values of bandwidth make smoother estimates, smaller values of bandwidth make less smooth estimates. Automatic bandwidth selectors are developed. Options include wnrd, wnrd0, wmise,blscv, and awmise.
freq	An indicator showing whether w is a vector of frequecies (counts) or weights.
gridsize	the number of equally spaced points at which to estimate the density.
range.x	vector containing the minimum and maximum values of x at which to compute the estimate. The default is the minimum and maximum data values, extended by the support of the kernel.
truncate	logical flag: if TRUE, data with x values outside the range specified by range.x are ignored.
na.rm	logical flag: if TRUE, NA values will be ignored; otherwise, the program will be halted with error information.

Details

The default bandwidth, "wnrd0", is computed using a rule-of-thumb for choosing the bandwidth of a Gaussian kernel density estimator based on weighted data. It defaults to 0.9 times the minimum of the standard deviation and the interquartile range divided by 1.34 times the sample size to the negative one-fifth power (= Silverman's 'rule of thumb', Silverman (1986, page 48, eqn (3.31)) _unless_ the quartiles coincide when a positive result will be guaranteed.

"wnrd" is the more common variation given by Scott (1992), using factor 1.06.

"wmise" is a completely automatic optimal bandwidth selector using the least-squares cross-validation (LSCV) method by minimizing the integrated squared errors (ISE).

Value

a list containing the following components:

x	vector of sorted x values at which the estimate was computed.
y	vector of density estimates at the corresponding x.
bw	optimal bandwidth.
sp	sensitivity parameter, none NA if adaptive bandwidth selector is used.

References

Wand, M. P. and Jones, M. C. (1995). *Kernel Smoothing.* Chapman and Hall, London.

Examples

```
mu = 34.5; s=1.5; n = 3000
x = round(rnorm(n, mu, s),1)
x0 = seq(min(x)-s,max(x)+s, length=100)
f0 = dnorm(x0,mu, s)

xt = table(x); n = length(x)
x1 = as.numeric(names(xt))
w1 = as.numeric(xt)
(h1 <- bw.wnrd0(x1, w1))
(h2 <- bw.wnrd0(x1,w1,n=n))

est1 <- wkde(x1,w1, bandwidth=h1)
est2 <- wkde(x1,w1, bandwidth=h2)
est3 <- wkde(x1,w1, bandwidth='awmise')
est4 <- wkde(x1,w1, bandwidth='wmise')
est5 <- wkde(x1,w1, bandwidth='blscv')

est0 = density(x1,bw="SJ",weights=w1/sum(w1));

plot(f0~x0, xlim=c(min(x),max(x)), ylim=c(0,.30), type="l")
lines(est0, col=2, lty=2, lwd=2)

lines(est1, col=2)
```

```
lines(est2, col=3)
lines(est3, col=4)
lines(est4, col=5)
lines(est5, col=6)
legend(max(x),.3,xjust=1,yjust=1,cex=.8,
 legend=c("N(34.5,1.5)", "SJ", "wnrd0",
 "wnrd0(n)","awmise","wmise","blscv"),
 col = c(1,2,2,3,4,5,6), lty=c(1,2,1,1,1,1,1),
 lwd=c(1,2,1,1,1,1,1))
```

`Zipf.Normalize` *Zipf Normalization*

Description

Zipf plot based normalization.

Usage

```
Zipf.Normalize(x, y, cutoff=6,optim=FALSE, method)
```

Arguments

`x,y`	data: two vectors.
`cutoff`	a large enought value such that the values larger than the `cutoff` (approximately) follows a power law distribution.
`optim`	Find the optimal normalization parameters if TRUE
`method`	use both power transformation and scalingby default. If 'scaling' is specified, skip power transformation.

Value

`x`	reference profile (not normalized)
`y`	normalized profile
`scaler`	Linear rescaling normalization parameter estimate
`power`	power transformation parameter estimate
`scaler.optim`	Optimized estimate of the linear rescaling parameter
`power.optim`	Optimzed estimate of the power transformation parameter.
`mat.optim`	A matrix of the objective function values generated to find the optimal estimates.
`coef`	Coefficient table to display the estimates.

References

Wang, B. (2020) A Zipf-plot based normalization method for high-throughput RNA-Seq data. PLoS ONE, (in press).

Examples

```
data(LCL)
names(LCL)
x <- LCL$p47
y <- LCL$p107
outx <- ZipfPlot(x)
plot(outx,type='l')
outy <- ZipfPlot(y)
lines(outy,col=2)

out2 <- Zipf.Normalize(x,y)
outy2 <- ZipfPlot(out2$y)
lines(outy2,col=4)
```

`ZipfPlot` *Draw Zipf Plot*

Description

Draw Zipf Plot.

Usage

```
ZipfPlot(x, x0, plot=FALSE,plot.new=TRUE, weights,...)
```

Arguments

`x`	data: two vectors.
`x0`	low bound to filter data.
`plot`	Draw Zipf plot if `TRUE`
`plot.new`	whether draw a new plot.
`weights`	Compute weighted least squares line if `weights` is given.
`...`	plotting parameters.

Value

None

References

Wang, B. (2020) A Zipf-plot based normalization method for high-throughput RNA-Seq data. PLoS ONE, (in press).

Examples

```
data(LCL)
names(LCL)
x <- LCL$p47
y <- LCL$p107
outx <- ZipfPlot(x)
plot(outx,type='l')
outy <- ZipfPlot(y)
lines(outy,col=2)

out2 <- Zipf.Normalize(x,y)
outy2 <- ZipfPlot(out2$y)
lines(outy2,col=4)
```

Index