Package 'fitDRC'

June 8, 2018

Type Package

Title Fitting Density Ratio Classes

Version 1.1.1

Date 2018-06-08

Author Simon L. Rinderknecht and Peter Reichert <peter.reichert@eawag.ch>

Maintainer Peter Reichert <peter.reichert@eawag.ch>

Description Fits Density Ratio Classes to elicited probability-quantile points or intervals.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2018-06-08 18:45:42 UTC

R topics documented:

`fitDRC-package` *Fitting Density Ratio Classes*

Description

Constructs the smallest *Density Ratio Class* for elicited probability-quantile points (or intervals) given a lower and upper distributional shape. Used optimisation algorithms are the methods `Nelder-Mead` and `L-BFGS-B` implemented in the standard R function `optim`. The package is easily customizable by using templates from the example section for distributions and transformations that are not implemented yet.

Details

Package:	fitDRC
Type:	Package
Version:	1.1.1
Date:	2018-06-08
License:	GPL-2
LazyLoad:	yes

The most important functions producing objects or results are:

(1) `dist.distributionfamily.create(...)`	see for the class description `distribution`
(2) `trans.transformationkind.create(...)`	see for the class description `transformation`
(3) `dist.trans.create(...)`	see for the constructor description `dist.trans.create`
(4) `process.elidat(...)`	see for the processing description `process.elidat`

(1) creates an object of the class `distribution` that is used to design lower and upper distribution of the *Density Ratio Class* whereby the implemented distributions are:

Normal	`dist.normal.create`
Student	`dist.student.create`
Weibull	`dist.weibull.create`
Lognormal	`dist.lognormal.create`
Beta	`dist.beta.create`
Gamma	`dist.gamma.create`
F	`dist.f.create`
Uniform	`dist.uniform.create`
Logistic	`dist.logistic.create`

(2) creates an object of the class `transformation` that is used to transform the distribution(s) with (3) (see `dist.trans.create`) returning again an object of the class `distribution`. However, implemented transformations are:

Arctan	`trans.arctan.create`
Tan	`trans.tan.create`
Log	`trans.log.create`
Exponential	`trans.exp.create`
Dilation	`trans.dil.create`

(4) processes the probability-quantile intervals/points in combination with the lower and upper distribution and returns an object of the class `drclass` that is the *Density Ratio Class* one wants to obtain (see `process.elidat`). Several methods are implemented for each class described above. Use the templates from the example sections for the implementation of distributions and transformations that are not implemented yet.

Author(s)

Simon L. Rinderknecht and Peter Reichert.

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht,

S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

Examples

```
################## Example 01 ################

# Demonstration of the construction of a very narrow Density Ratio Class
# for a case where the input quantiles correspond to the quantiles of the
# parametric distribution used for the lower and upper densities (in this
# case both are Normal distributions).

# Definition of quantiles:

p <- c(0.05, 0.25, 0.5, 0.75, 0.95)
q <- qnorm(p)

# Definition of parametric shapes of lower and upper densities (Normal):

dist.lower <- dist.normal.create(par=c(1,2))
dist.upper <- dist.normal.create(par=c(3,4))

# Parameter estimation (attention: runtime several minutes):

#res <- process.elidat(p = p,
#                      q = q,
#                      dist.lower,
#                      dist.upper,
#                      start.dist.lower.par = c("Mean"=2,"StDev"=3),
#                      start.dist.upper.par = c("Mean"=4,"StDev"=5))

# Extract density ratio class, print and plot results:

#drc01 <- res$drc
#print(drc01)
#plot(drc01)

################## Example 02 ################

# Demonstration of the construction of a Density Ratio Class using tans-
# formed beta distributions for lower and upper densities.

# Definition of quantiles:

p <- c(0.05, 0.25, 0.5, 0.75, 0.95)
q <- c(80, 145, 200, 240, 280)

# Definition of parametric shapes of lower and upper densities (transf. beta):

dist.beta  <- dist.beta.create(par=c(2.5,2.5))
trans.dil <- trans.dil.create(c(60, 305, 0, 1))

dist.lower <- dist.trans.create(dist.beta,trans.dil)
```

```
dist.upper <- dist.lower

# Parameter estimation (attention: runtime several minutes):

#res <- process.elidat(p = p,
#                      q = q,
#                      dist.lower,
#                      dist.upper,
#                      start.dist.lower.par =c("Shape1"=2.5,"Shape2"=2.5),
#                      start.dist.upper.par = c("Shape1"=2.5,"Shape2"=2.5))

# Extract density ratio class, print and plot results:

#drc02 <- res$drc
#print(drc02)
#plot(drc02)

# Note, due to the transformation, mean, standard deviation, median and mode
# cannot be calculated analytically (print(drc02 returns NA)). However, these
# characteristics can be calculated  numerically, using a sample from the
# distribution (demonstrated for the lower density):

# Mean:
#mean(CDFinv(drc02$dist.lower,runif(100000,0,1),drc02$dist.lower$par))

# Standard Deviation:
#sd(CDFinv(drc02$dist.lower,runif(100000,0,1),drc02$dist.lower$par))

# Median:
#CDFinv(drc02$dist.lower,0.5)

# Mode:
#samp <- runif(100000,0,1)
#ind.max <- which.max(PDF(drc02$dist.lower,
#                         CDFinv(drc02$dist.lower,samp,drc02$dist.lower$par),
#                         drc02$dist.lower$par))
#CDFinv(drc02$dist.lower,samp[ind.max],drc02$dist.lower$par)

################## Example 03 ################

# Demonstration of the construction of a Density Ratio Class using different
# parametric shapes for lower (Normal) and upper (Student t) distributions.

# Definition of quantiles:

p <- c(0.05, 0.25, 0.5, 0.75, 0.95, 0.05, 0.25, 0.5, 0.75, 0.95)
q <- log(c(1, 2, 4, 6, 10, 2, 3, 5, 9, 14))

# Definition of parametric shapes of lower and upper densities (Normal and
# Student t):

dist.lower <- dist.normal.create(par=c(1,1))
dist.upper <- dist.student.create(par=c(1,1,3))
```

```
# Parameter estimation (attention: runtime several minutes):

#res <- process.elidat(p = p,
#                      q = q,
#                      dist.lower,
#                      dist.upper,
#                      start.dist.lower.par = c("Mean"=1,"StDev"=1),
#                      start.dist.upper.par = c("Mean"=1,"StDev"=1))

# Extract density ratio class, print and plot results:

#drc03 <- res$drc
#print(drc03)
#plot(drc03,range=c(0.001,15))
```

`aberr.l.bfgs.b` *aberr.l.bfgs.b*

Description

No description, see information about the package `fitDRC` or in the code of it.

`aberr.nelder.mead` *aberr.nelder.mead*

Description

No description, see information about the package `fitDRC` or in the code of it.

`calc.k` *Calculating the ratio of normalising constants of the not necessarily normalised lower and upper probability density functions of a Density Ratio Class.*

Description

`calc.k` calculates `Kappa` which is the ratio of the normalising constants of the lower and upper probability density functions of a *Density Ratio Class*. It can be interpreted as a measure of the size of a *Density Ratio Class*. Hence `Kappa` is the target value that is minimized in the algorithm.

Usage

```
calc.k(p, q, dist.lower, dist.upper, par.lower, par.upper)
```

Arguments

p	Probabilities (in the corresponding order to the quantiles)
q	Quantiles (in the corresponding order to the probabilities)
dist.lower	Lower distribution of the *Density Ratio Class* (see distribution).
dist.upper	Upper distribution of the *Density Ratio Class* (see distribution).
par.lower	Distributional parameters of the lower distribution.
par.upper	Distributional parameters of the upper distribution.

Details

To calculate Kappa the density with heavier tails *must* be the upper one. Otherwise the choice is *not* compatible with the definition of the Density Ratio Class (see for an example below).

Value

Kappa	Kappa >=1 the minimally possible ratio of the not normalised probability density functions of a Density Ratio Class that is compatible with the chosen lower and upper distribution families and elicited probability-quantile points or intervals.
Lambda	Lambda >=1 the minimally possible ratio of the not normalised probability density functions that is compatible with the chosen lower and upper distribution families only.

Note

For a better understanding what Kappa and Lambda are see also Eq. (4), (10), (11) and (12) in the referenced paper.

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

See also fitDRC.

Examples

```
p <- c(0.05, 0.25, 0.5, 0.75, 0.95)
q <- qnorm(p)

dist.lower <- dist.normal.create(par=c(1,2))
dist.upper <- dist.normal.create(par=c(3,4))

par.lower <- dist.lower$par
par.upper <- dist.upper$par

#calc.k(p, q, dist.lower, dist.upper, c(0,1), c(0,1))       # perfect matching
                                                            # of elicited data
                                                            # with chosen shapes
#calc.k(p, q, dist.lower, dist.upper, par.lower, par.upper) # general case
#calc.k(p, q, dist.lower, dist.upper,  c(0,50), c(0,1))     # not compatible
                                                            # with the def. of
                                                            # the DRC.


## The function is currently defined as
function (p, q, dist.lower, dist.upper, par.lower, par.upper)
{
    x.min.lower <- CDFinv(dist.lower, 0.001, par.lower)
    x.max.lower <- CDFinv(dist.lower, 0.999, par.lower)
    x.min.upper <- CDFinv(dist.upper, 0.001, par.upper)
    x.max.upper <- CDFinv(dist.upper, 0.999, par.upper)
    x.min <- min(x.min.lower, x.min.upper)
    x.max <- max(x.max.lower, x.max.upper)
    x <- seq(x.min, x.max, length = 1e+06)
    Lambda <- max(PDF(dist.lower, x, par.lower)/PDF(dist.upper,
                  x, par.upper))
    k1 <- (p * (1 - CDF(dist.lower, q, par.lower)))/(CDF(dist.upper,
           q, par.upper) * (1 - p))
    k2 <- (CDF(dist.lower, q, par.lower) * (1 - p))/(p * (1 -
           CDF(dist.upper, q, par.upper)))
    Kappa <- max(Lambda, k1, k2)
    return(list(Kappa = Kappa, Lambda = Lambda))
  }
```

CDF *CDF*

Description

No description, see information about the package fitDRC or in the code of it.

CDFinv *CDFinv*

Description

No description, see information about the package fitDRC or in the code of it.

dist.trans.create *Transformed distributions.*

Description

Function that transforms a distribution.

Usage

```
dist.trans.create(dist, trans)
```

Arguments

dist	Object of the class distribution.
trans	Object of the class transformation.

Details

no details

Value

distribution	Transformed distribution as an object of the class distribution.

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

See also fitDRC.

Examples

```
## Example 01 ##
trans.tan <- trans.tan.create(c(33,99))
dist.student <- dist.student.create(c(0,1,1000))
dist  <- dist.trans.create(dist.student,trans.tan)
dist
#plot(dist)

## Example 02 ##
trans.arctan <- trans.arctan.create(c(0,10))
dist.uniform <- dist.uniform.create(c(0,10))
dist         <- dist.trans.create(dist.uniform,trans.arctan)
dist
#plot(dist)
```

`Distributions` *The class 'distribution' in the fitDRC-package: its constructors and methods.*

Description

Lower and upper distributions of the *Density Ratio Class* must be in the form of an object of the class `distribution` such as described in this sheet. Objects of the class `distribution` can be used in a second step in function `process.elidat` that finally yields the smallest *Density Ratio Class* given the the probability-quantile intervals/poins. The described functions below create `distribution` objects for wich some methods are implemented too. The distributional parameter(s) (at least one) that finally shall be optimized for the smallest *Density Ratio Class* must be specified with name. For transformed distributions see `transformation` and `dist.trans.create`.

Usage

```
dist.normal.create(par = NA)
dist.student.create(par = NA)
dist.weibull.create(par = NA)
dist.lognormal.create(par = NA)
dist.beta.create(par = NA)
dist.gamma.create(par = NA)
dist.f.create(par = NA)
dist.uniform.create(par= NA)
dist.logistic.create(par = NA)

## S3 method for class 'distribution'
print(x = dist,...)
## S3 method for class 'distribution'
summary(object = dist,...)
## S3 method for class 'distribution'
plot(x = dist, par = dist$par, range = NA, what = "PDF", plot = TRUE, length = 101, ...)
```

```
## S3 method for class 'distribution'
PDF(dist, x, par = dist$par, log = FALSE,...)
## S3 method for class 'distribution'
CDF(dist, x, par = dist$par,...)
## S3 method for class 'distribution'
CDFinv(dist, p, par = dist$par,...)
## S3 method for class 'distribution'
RANGE(dist, par = dist$par,...)
## S3 method for class 'distribution'
MEAN(dist, par = dist$par, ...)
## S3 method for class 'distribution'
SD(dist, par = dist$par, ...)
## S3 method for class 'distribution'
MEDIAN(dist, par = dist$par, ...)
## S3 method for class 'distribution'
MODE(dist, par = dist$par, ...)
```

Arguments

par	vector of the parameters of the distribution, if not named in the implemented order as shown in the list below for each implemented distribution. At least one parameter value has to be specified. Unspecified parameter values will take the default values (see list below).

normal:	`par = c("Mean" = 0,"StDev" = 1)`
student:	`par = c("Mean" = 0, "StDev" = 1, "DF" = 3)`
weibull:	`par = c("Shape" = 2, "Scale" = 2 )`
lognormal:	`par = c("Mean" = 1, "StDev" = 1)`
beta:	`par = c("Shape1" = 1,"Shape2" = 1)`
gamma:	`par = c("shape" = 1,"rate" = 1)`
f:	`par = c("df1" = 3, "df2" = 5, "ncp" = 0)`
uniform:	`par = c("Min" = 0,"Max" = 1)`
logistic:	`par = c("Location" = 0,"Scale" = 1)`

dist	object of class `distribution`.
x	in dependence of the function either an object of the class `distribution` or the location of where to evaluate the distribution.
object	object of the class `distribution`.
p	probability of where the inverse distribution has to be evaluated.
range	used in the method `plot`: specifies the plot range
what	determines what to plot or calculate, can be set to: `PDF`, `CDF` or `CDFinv`.
plot	argument used in the method `plot`: creates a plot if set to TRUE, returns a matrix with x an y column if set to FALSE.
length	plot resolution.
log	if TRUE the logarithm of the PDF is returned, default is FALSE.
...	further arguments that can be passed to a function.

Details

Implement your own distribution using the template from the example section below if the distribution you are looking for is not implemented.

Value

`name`	the name of the distribution
`range`	the range of the distribution
`par.names`	the names of the parameters of the distribution
`par.ranges`	the ranges of the parameters of the distribution
`par`	the values of the parameters of the distribution
`mean`	a function to calculate the mean of the distribution
`sd`	a function to calculate the standard deviation of the distribution
`median`	a function to calculate the median of the normal distribution
`mode`	a function to calculate the mode of the distribution (does not exist for e.g. the Uniform distribution)
`pdf`	a function to calculate the pdf (probability density function) of the distribution
`cdf`	a function to calculate cdf (cumulative distribution function) of the distribution
`cdf.inv`	a function to calculate the inverse cdf of the distribution

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

See also `fitDRC` for general information and `transformation`, `dist.trans.create` for transformed distributions.

Examples

```
print(dist.normal.create(c(Mean = 0, StDev = 1)))
print(dist.student.create(c(DF=99)))
dist.weibull.create(c(Shape=2,Scale=99))
summary(dist.lognormal.create(c(StDev=2)))

plot(dist.beta.create(c(2,1)),plot=FALSE)
plot(dist.gamma.create(c(2,1)),main="myGamma",xlab="x",ylab="pdf")
plot(dist.f.create(c(ncp=99)),main="F",what="CDF",xlab="x",ylab="cdf")
```

```
plot(dist.uniform.create(c(-1,1)),main="Uniform",what="CDFinv",xlab="p",ylab="inv-cdf")
plot(dist.logistic.create(c(2,1)),par=c(Scale=5))

dist.normal <- dist.normal.create(c(StDev=2))
is(dist.normal)                       # element of class distribution
plot(dist.normal,par=c(StDev=3))
dist.normal$par <- c(2,2)             # "permanent" parameter change
plot(dist.normal)
plot(dist.normal, par = c(Mean = 0))  # "temporary" parameter change

# Default setting of the parameters:

dist.normal.create(par = c(Mean = 0, StDev = 1))
dist.student.create(par = c("Mean" = 0, "StDev" = 1, "DF" = 3))
dist.weibull.create(par = c( "Shape" = 2, "Scale" = 2 ))
dist.lognormal.create(par = c("Mean" = 1, "StDev" = 1))
dist.beta.create(par = c("Shape1" = 1, "Shape2" = 1))
dist.gamma.create(par = c("shape" = 1, "rate" = 1))
dist.f.create(par = c("df1" = 3,"df2" = 5, "ncp" = 0))
dist.uniform.create(par= c("Min" = 0, "Max" = 1))
dist.logistic.create(par = c("Location" = 0,"Scale" = 1))



##############################################################################
       ### if you want to create your own distribution read this ###
##############################################################################

# use the template below and replace the code in between *<* ... *>*
# accordingly. Do not forget to delete the *<* and *>* that are only used to
# indicate the custom fields.
# type 'dist.normal.create' to see an already implemented distribution.

##############################################################################
### if you want to create your own distribution use the following template ###
##############################################################################

# dist.*<*YourDistributionFamilyName*>*.create <- function(par=NA)
# {
#   # set default parameter values:
#   par.default <- c(*<*YourFirstParamterDefaulValue, ...*>*)
#   names(par.default) <- c( *<*"YourParameterNameOfYourFirstParamter", "..."*>* )
#   p <- mergePar(par,par.default)
#   # construct class:
#   dist            <- list()
#   dist$name       <- "*<*YourDistributionName*>*"
#   dist$range      <- function(par) # range of the distribution
#                      {
#                         return(c(*<*YourLowerRange,YourUperRange*>*))
#                      }
#   dist$par.names  <- names(p)
#   dist$par.ranges <- matrix(
#                             c(*<*-NA, +NA,*>*      # ranges of par01
```

```
#                               *<*-NA, +NA*>*),     # ...
#                               byrow=TRUE,ncol=2)
#   dist$par        <- p
#   dist$mean       <- function(par)
#                      {
#                         mean <- *<*MeanFormula(par)*>*
#                      names(mean) <- "Mean"
#                      return(mean)
#                      }
#   dist$sd         <- function(par)
#                      {
#                        sd <- *<*StandardDevFormula(par)*>*
#                        names(sd) <- "StDev"
#                        return(sd)
#                      }
#   dist$median     <- function(par)
#                      {
#                        median <- *<*MedianFormula(par)*>*
#                        names(median) <- "Median"
#                        return(median)
#                      }
#   dist$mode       <- function(par)
#                     {
#                        mode <- *<*ModeFormula(par)*>*
#                        names(mode) <- "Mode"
#                        return(mode)
#                      }
#   dist$pdf        <- function(x,par) { return( *<*dyourdist(x, par)*>* ) }
#   dist$cdf        <- function(x,par) { return( *<*pyourdist(x, par)*>* ) }
#   dist$cdf.inv    <- function(p,par) { return( *<*qyourdist(p, par)*>* ) }
#   class(dist)     <- "distribution"
#   return(dist)
#}
```

`drclass` *The class 'drclass' in the fitDRC-package: its constructors and methods.*

Description

An object of the class `drclass` defines a *Density Ratio Class* and has a structure of a list containing `name`, `range`, `p`, `q`, `dist.lower`, `dist.upper`. Methods for density ratio class such as `metrics` calculates the relative (to a credible interval `1-alpha`) ambiguity of important attributes such as width, shape and mode. See the referenced literature for further information.

Usage

```
drclass.create(p = c(0.05,0.25,0.5,0.75,0.95),
               q = qnorm(c(0.05,0.25,0.5,0.75,0.95)),
```

```
                dist.lower = dist.normal.create(c(0,1)),
                dist.upper = dist.normal.create(c(0,1)))

## S3 method for class 'drclass'
print(x  = drc, ...)
## S3 method for class 'drclass'
summary(object = drc, alpha = 0.05, ...)
## S3 method for class 'drclass'
plot(x = drc, range = NA, plot.stat.values = FALSE, makePDF = FALSE, ...)

## S3 method for class 'drclass'
Kappa(drc, ...)
## S3 method for class 'drclass'
Lambda(drc, ...)

## S3 method for class 'drclass'
metrics(drc, alpha = 0.05, ...)
## S3 method for class 'drclass'
metric.ci(drc, alpha = 0.05, ...)
## S3 method for class 'drclass'
metric.width(drc, alpha = 0.05, ...)
## S3 method for class 'drclass'
metric.shape(drc, alpha = 0.05, ...)
## S3 method for class 'drclass'
metric.mode(drc, alpha = 0.05, ...)
```

Arguments

p	vector of probabilities according to q.
q	vector of quantiles according to p.
dist.lower	object of the class distribution.
dist.upper	object of the class distribution.
x	object of the class drclass
object	object of the class drclass
drc	object of the class drclass
alpha	Defines the credible level 1-alpha.
range	Plotrange.
plot.stat.values	Statistical values are added to the plot if TRUE.
makePDF	Creates a pdf if TRUE else not.
...	Further arguments that can be passed to the function.

Details

No details.

Value

name	the name of the Density Ratio Class
range	the range of the Density Ratio Class
p	probabilities
q	quantiles
dist.lower	object of the class distribution
dist.upper	object of the class distribution

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

See also fitDRC for general information and distribution, transformation, dist.trans.create for details.

Examples

```
drc <- drclass.create (p = c(0.05,0.25,0.666,0.75,0.95),
                       q = qnorm(c(0.05,0.25,0.5,0.75,0.95)),
                       dist.lower = dist.normal.create(c(0,1)),
                       dist.upper = dist.normal.create(c(0,1)))
drc
print(drc)   # prints the Density Ratio Class.
#summary(drc) # adds the metrics.
#plot(drc)    # plots the Density Ratio Class.

Kappa(drc)
Lambda(drc)

#metrics(drc) # all metrics.
#metric.ci(drc, 0.1) # outer credible interval for 0.9 content

##############################################################################
### if you want to create your own Density Ratio Class use the following template ###
##############################################################################

# drclass.create <- function(p = c(yourProbabilities),    # according to q
#                            q = qnorm(c(yourQuantiles)),  # according to p
#                            dist.lower = dist.yourDistribution.create(par),
#                            dist.upper = dist. yourDistribution.create(par)  )
```

```
# {
#
# drc                     <- list()
# drc$name                <- "yourDRCname"
# drc$range               <- dist.upper$range(dist.upper$par)
# drc$p                   <- p
# drc$q                   <- q
# drc$dist.lower          <- dist.lower
# drc$dist.upper          <- dist.upper
# class(drc)              <- "drclass"
# return(drc)
# }
```

`Fl` *Fl*

Description

No description, see information about the package `fitDRC` or in the code of it.

`Fl.drclass` *Fl.drclass*

Description

No description, see information about the package `fitDRC` or in the code of it.

`FlInv` *FlInv*

Description

No description, see information about the package `fitDRC` or in the code of it.

`FlInv.drclass` *FlInv.drclass*

Description

No description, see information about the package `fitDRC` or in the code of it.

`Fu` *Fu*

Description

No description, see information about the package `fitDRC` or in the code of it.

`Fu.drclass` *Fu.drclass*

Description

No description, see information about the package `fitDRC` or in the code of it.

`FuInv` *FuInv*

Description

No description, see information about the package `fitDRC` or in the code of it.

`FuInv.drclass` *FuInv.drclass*

Description

No description, see information about the package `fitDRC` or in the code of it.

`Kappa` *Kappa*

Description

No description, see information about the package `fitDRC` or in the code of it.

Lambda *Lambda*

Description

No description, see information about the package fitDRC or in the code of it.

MEAN *MEAN*

Description

No description, see information about the package fitDRC or in the code of it.

MEDIAN *MEDIAN*

Description

No description, see information about the package fitDRC or in the code of it.

mergePar *mergePar*

Description

No description, see information about the package fitDRC or in the code of it.

metric.ci *metric.ci*

Description

No description, see information about the package fitDRC or in the code of it.

`metric.mode` *metric.mode*

Description

No description, see information about the package `fitDRC` or in the code of it.

`metric.shape` *metric.shape*

Description

No description, see information about the package `fitDRC` or in the code of it.

`metric.width` *metric.width*

Description

No description, see information about the package `fitDRC` or in the code of it.

`metrics` *metrics*

Description

No description, see information about the package `fitDRC` or in the code of it.

`MODE` *MODE*

Description

No description, see information about the package `fitDRC` or in the code of it.

PDF *PDF*

Description

No description, see information about the package `fitDRC` or in the code of it.

process.elidat *Process the elicited data to an optimised* Density Ratio Class

Description

Constructs the smallest *Density Ratio Class* for elicited probability-quantile points (or intervals) given a lower and upper distributional shape. Used optimisation algorithms are the methods `Nelder-Mead` and `L-BFGS-B` implemented in the standard R function `optim`.

Usage

```
process.elidat(p = p, q = q, dist.lower, dist.upper,
               start.dist.lower.par = NA, start.dist.upper.par = NA, ...)
```

Arguments

p	Vector of probabilities in the according order to q.
q	Vector of quantiles in the according order to p.
dist.lower	Lower distribution as an element of the class `distribution`.
dist.upper	Upper distribution as an element of the class `distribution`.
start.dist.lower.par	Start values of the parameters of the lower distribution that shall be optimized.
start.dist.upper.par	Start values of the parameters of the upper distribution that shall be optimized.
...	

Details

Only the specified start values of the lower and upper distribution are optimised. If no optimisation shall be executed (fixed parameters of the distributions) then use `calc.k`.

Value

drclass	an object of the class `drclass`.

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

See also `fitDRC`, `distribution`, `transformation`, `dist.trans.create`.

`RANGE` *RANGE*

Description

No description, see information about the package `fitDRC` or in the code of it.

`SD` *SD*

Description

No description, see information about the package `fitDRC` or in the code of it.

`TRANS.BACKW.transformation`
TRANS.BACKW.transformation

Description

No description, see information about the package `fitDRC` or in the code of it.

`TRANS.DERIV.transformation`
TRANS.DERIV.transformation

Description

No description, see information about the package `fitDRC` or in the code of it.

`TRANS.FORW.transformation` *TRANS.FORW.transformation*

Description

No description, see information about the package `fitDRC` or in the code of it.

`trans.from.interval.to.interval` *trans.from.interval.to.interval*

Description

No description, see information about the package `fitDRC` or in the code of it.

`trans.from.interval.to.R` *trans.from.interval.to.R*

Description

No description, see information about the package `fitDRC` or in the code of it.

`trans.from.R.to.interval` *trans.from.R.to.interval*

Description

No description, see information about the package `fitDRC` or in the code of it.

`trans.from.R.to.Rplus` *trans.from.R.to.Rplus*

Description

No description, see information about the package `fitDRC` or in the code of it.

`trans.from.Rplus.to.R` *trans.from.Rplus.to.R*

Description

No description, see information about the package `fitDRC` or in the code of it.

`TRANS.RANGE.X.transformation`
TRANS.RANGE.X.transformation

Description

No description, see information about the package `fitDRC` or in the code of it.

`TRANS.RANGE.Y.transformation`
TRANS.RANGE.Y.transformation

Description

No description, see information about the package `fitDRC` or in the code of it.

`Transformations` *The class 'transformation' in the fitDRC package: its constructors and methods.*

Description

To transform a lower or upper distribution in order to find even a better fit for a *Density Ratio Class*, one has firstly to specify an object of the class `transformation` due to the constructors `trans.transformationkind.create(par)` that are described in this help sheet. Secondly, once an object of the class `transformation` is created use the function `dist.trans.create` to obtain an object of the class `distribution` that is finally used for fitting the *Density Ratio Class* with the help of the function `process.elidat`.

Implemented tranformations are the `arctan`, `tan`, `dilation`, `log` and a particular `trans.exp.create` transformation. They are defined as follows:

```
arctan:   0.5*(Min+Max) + (Max-Min)/pi*atan(x)
 tan:     tan(0.5*pi*(2*x-Max-Min)/(Max-Min))
 dil:     (x-Min1) * (Max2-Min2)/(Max1-Min1) + Min2
 log:     log(x)
 exp:     -(a/b^2) * exp(-b*x) + c*x + (a/b^2)
```

It is also possible to implement an own object of the class `transformation`. Do this by using the template below from the example section.

Usage

```
trans.arctan.create(par = NA)
trans.tan.create(par = NA)
trans.dil.create(par = NA)
trans.log.create(par = NA)
trans.exp.create(par = NA)

## S3 method for class 'transformation'
print(x = trans,...)
## S3 method for class 'transformation'
summary(object,...)
## S3 method for class 'transformation'
plot(x = trans, par = trans$par,
        range.x = NA, range.y = NA, what = "TRANS.FORW", plot = TRUE,
        length = 101,...)
```

Arguments

`par`	vector of the parameters of the transformation, if not named in the implemented order. At least one parameter value has to be specified. Unspecified values will be default values as in the list below:

arctan:	`par = c("Min" = 0, "Max" = 1)`
tan:	`par = c("Min" = 0,"Max" = 1)`
dil:	`par = c("Min1" = 0, "Max1" = 1, "Min2" = 0, "Max2" = 1)`
log:	`par = c("-" = NA)`
exp:	`par = c("a" = 0, "b" = 1, "c" = 0)`

`x`	object of the class `transformation`.
`object`	object of the class `transformation`.
`plot`	used in the method `plot`; if TRUE creates a plot, else returns values.
`what`	used in the method `plot`; can be either `TRANS.FORW` or `TRANS.BACKW` or `TRANS.DERIV` and defines what is to be plotted.
`range.x`	specifies the x-range of the plot in the method `plot`.
`range.y`	specifies the y-range of the plot in the method `plot`.
`length`	specifies the number of evaluations within the range for the plot in the method `plot`.
`...`	further arguments that can be passed to the function.

Details

Implemented methods for objects of the class `transformation` are: `print` `summary` `plot`.

Value

name	the name of the transformation
range.x	the x-range of the transformation
range.y	the y-range of the transformation
par.names	the names of the transformation parameters
par.ranges	the ranges of the transformation parameters
par	the values of the transformation parameters
trans.forw	a function to calculate the forward transformation
trans.backw	a function to calculate the backward transformation
trans.deriv	a function to calculate the derivation of the transformation

Author(s)

Simon L. Rinderknecht

References

Rinderknecht, S.L., Borsuk, M.E. and Reichert, P. Eliciting Density Ratio Classes. International Journal of Approximate Reasoning 52, 792-804, 2011. doi10.1016/j.ijar.2011.02.002. \ Rinderknecht, S. L., Borsuk, M. E. and Reichert, P. Bridging Uncertain and Ambiguous Knowledge with Imprecise Probabilities, Environmental Modelling & Software 36, 122-130, 2012.

See Also

fitDRC, distribution, dist.trans.create and process.elidat.

Examples

```
trans.arctan <- trans.arctan.create(c(0,10))
print(trans.arctan)
summary(trans.arctan)
#x11()
#plot(trans.arctan)
#plot(trans.arctan,what = "TRANS.BACKW")
#plot(trans.arctan,what = "TRANS.DERIV")

trans.tan <- trans.tan.create(c(0,10))
#x11()
#plot(trans.tan)
#plot(trans.tan,what = "TRANS.BACKW")
#plot(trans.tan,what = "TRANS.DERIV")

trans.log <- trans.log.create()
#x11()
#plot(trans.log,range.x=c(-1,1))
#plot(trans.log,what = "TRANS.BACKW",range.y=c(-1,1))
#plot(trans.log,what = "TRANS.DERIV",range.x=c(-1,1))
```

```
trans.dil <- trans.dil.create(c(0,1,4,5))
#x11()
#plot(trans.dil,range.x=c(-1,1))
#plot(trans.dil,what = "TRANS.BACKW", range.y = c(-1,1))
#plot(trans.dil,what = "TRANS.DERIV", range.x = c(-1,1))

trans.exp <- trans.exp.create(c(3, 2, 1))
#x11()
#plot(trans.exp,range.x=c(-1,1))
#plot(trans.exp,what = "TRANS.BACKW", range.y = c(-4,3))
#plot(trans.exp,what = "TRANS.DERIV", range.x = c(-1,1))

# implemented default values are:
trans.arctan.create(par = c(Min = 0, Max = 1))
trans.tan.create(par = c(Min = 0, Max = 1))
trans.dil.create(par = c("Min1" = 0, "Max1" = 1, "Min2" = 0, "Max2" = 1))
trans.log.create(par = c("-" = NA))
trans.exp.create(par = c("a" = 0, "b" = 1, "c" = 0))


##############################################################################
      ### if you want to create your own transformation read this ###
##############################################################################

# use the template below and replace the code in between *<* ... *>*
# accordingly. Do not forget to delete the *<* and *>* that are only used to
# indicate the custom fields.
# type 'trans.exp.create' to see an already implemented transformation.

##############################################################################
### if you want to create your own transformation use the following template ###
##############################################################################

# trans.*<*yournameofyourtransformation*>*.create <- function(par=c(NA)
# {
#   # set default parameter values:
#   par.default <- c(*<*NA,...*>*)
#   names(par.default) <- c(*<*"a",...*>*)
#   p <- mergePar(par,par.default)
#   # construct class:
#   trans             <- list()
#   trans$name        <- "*<*yourname"*>*
#   trans$range.x     <- function(par){*<*return(c(min.x,max.x))*>*}
#   trans$range.y     <- function(par){*<*return(c(min.y,max.y))*>*}
#   trans$par.names   <- names(p)
#   # ranges of the parameters of the transformation
#   trans$par.ranges  <- matrix(
#                               c(*<*-NA, +NA*>*,      # range of 1st parameter
#                                 *<*-NA, +NA*>*),     # range of 2nd par.....
#                               byrow=TRUE,ncol=2)
#   trans$par         <- p
#   trans$trans.forw  <- function(x,par)
#                        { y <- *<*yourForwardFormula(x,par)*>*
```

```
#                         return(as.numeric(y))
#                       }
#   trans$trans.backw <- function(y,par)
#                       { x <- *<*yourBackwardFormula(y,par)*>*
#                         return(as.numeric(x)) }
#   trans$trans.deriv <- function(x,par)
#                       { dydx <- *<*yourDerivationFormula(x,par)*>*
#                         return(as.numeric(dydx)) }
#   class(trans)      <- "transformation"
#   return(trans)
# }
```

Index