Package 'QuantileGH'

October 12, 2022

Type Package

Title Quantile Least Mahalanobis Distance Estimator for Tukey g-&-h Mixture

Version 0.1.2

Date 2022-05-05

Author Tingting Zhan [aut, cre, cph], Inna Chervoneva [ctb]

Maintainer Tingting Zhan <tingtingzhan@gmail.com>

Description Functions for simulation, estimation, and model selection of finite mixtures of Tukey's g-and-h distributions.

License GPL-2

Imports methods, goftest, graphics, LaplacesDemon, latex2exp, mixtools, rstpm2, scales, tclust

Encoding UTF-8

Language en-US

Depends ggplot2

Suggests fitdistrplus,lmtest,gk,OpVaR,dgof

RoxygenNote 7.1.2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-05-05 23:30:07 UTC

R topics documented:

`QuantileGH-package` *Quantile Least Mahalanobis Distance Estimator for Tukey g-&-h Mixture*

Description

Tools for simulating and fitting finite mixtures of the 4-parameter Tukey g-&-h distributions. Tukey g-&-h mixture is highly flexible to model multimodal distributions with variable degree of skewness and kurtosis in the components. The Quantile Least Mahalanobis Distance estimator (`QLMDe`) is used for estimating parameters of the finite Tukey g-&-h mixtures. `QLMDe` is an indirect estimator that minimizes the Mahalanobis distance between the sample and model-based quantiles. A backward-forward stepwise model selection algorithm is provided to find

- a parsimonious Tukey g-&-h mixture model, conditional on a given number-of-components; and
- the optimal number of components within the user-specified range.

Examples

```
# see ?QLMDe
```

autoplot.fitdist *Plot fitdist Object using* R*hrefhttps://CRAN.R-project.org/package=ggplot2***ggplot2**

Description

Plot fitdist object using **ggplot2**.

Usage

```
## S3 method for class 'fitdist'
autoplot(
  object,
  data = object[["data"]],
  type = c("density", "distribution"),
  xlim = c(min(data), max(data)),
  obs.col = "black",
  est.col = "red",
  xlab = NULL,
  ylab = object$distname,
  title = NULL,
  caption = NULL,
  ...
)
```

Arguments

`object`	a fitdist object
`data`	numeric or integer vector, actual observations
`type`	character scalar, whether the `'density'` (default) or the probability `'distribution'` curve should be plotted
`xlim`	numeric vector of length two, the horizontal limit of the figure. Default value is the range of the actual observations.
`obs.col`	character scalar, color to represent the observed values, default black
`est.col`	character scalar, color to represent the estimated values, default red
`xlab, ylab, title, caption`	character scalars, see labs
`...`	potential parameters of stat_function

Value

autoplot.fitdist plots fitdist object using **ggplot2**. No value is returned.

Examples

```
library(fitdistrplus)
x1 = rpois(n = 100, lambda = 4)
xfit1 = fitdist(x1, distr = 'pois')
autoplot(xfit1, xlim = c(-3L, 15L))
autoplot(xfit1, type = 'distribution')

x2 = rnorm(n = 1e3L, mean = 2.3, sd = .7)
xfit2 = fitdist(x2, distr = 'norm')
autoplot(xfit2, type = 'density')
autoplot(xfit2, type = 'distribution')
```

---

autoplot_fmx *Plot fmx and fmx_QLMDe objects using* R*hrefhttps://CRAN.R-project.org/package=ggplot2***ggplot2**

Description

Plot fmx and fmx_QLMDe objects using **ggplot2**.

Usage

```
## S3 method for class 'fmx'
autoplot(
  object,
  type = c("density", "distribution"),
  data = attr(object, which = "data", exact = TRUE),
  epdf = attr(object, which = "epdf", exact = TRUE),
  probs = attr(object, which = "p", exact = TRUE),
  init = attr(object, which = "init", exact = TRUE),
  origK = attr(object, which = "orig_K", exact = TRUE),
  xlim = if (!length(data)) qfmx(p = c(0.01, 0.99), dist = object),
  hist.fill = "grey95",
  hist.col = "white",
  curve.col = 1,
  xlab = attr(object, which = "data.name", exact = TRUE),
  ylab = paste(object@distname, "mixture"),
  title = TeX(fmx_constraint_brief(object)),
  caption = NULL,
  ...
)
```

Arguments

object	an fmx or fmx_QLMDe object

`type`	character scalar. Option `'density'` (default) plots the probability density for fmx input or the histogram and the estimated probability density for fmx_QLMDe input. Option `'distribution'` plots the cumulative probability distribution for fmx input or the empirical cumulative distribution together with the estimated cumulative distribution function for fmx_QLMDe input.
`data`	(optional) numeric vector of the observations. For fmx_QLMDe input, the default is `object@data`.
`epdf`	..
`probs`	numeric vector, the percentages (to be) used in `QLMDe`, can be plotted as vertical lines. Use `probs = NULL` to suppress the printing of these lines.
`init`	logical scalar, whether to plot the initial estimates used in `QLMDe`, default `FALSE`.
`origK`	logical scalar, whether to plot the fmx_QLMDe at the user-specified number of components K if backward-forward selection on number of component is performed, default `FALSE`.
`xlim`	horizontal range, see stat_function.
`hist.fill`	color of the body of histogram, default `'grey95'`. Passed as parameter `fill` in geom_histogram.
`hist.col`	color of the border around the histogram bars, default `'white'`. See parameter `colour` of geom_histogram.
`curve.col`	color of the density curve of the fitted finite mixture distribution. Passed as parameter `colour` in stat_function.
`xlab, ylab, title, caption`	character scalars, the horizontal and vertical label, title and caption. See `xlab`, `ylab`, labs.
`...`	potential parameters of stat_function

Value

autoplot.fmx returns a ggplot object.

See Also

autoplot

Examples

```
(d2 = fmx('GH', A = c(1,6), B = 2, g = c(0,.3), h = c(.2,0), w = c(1,2)))
curve(dfmx(x, dist = d2), xlim = c(-3, 11))
curve(pfmx(x, dist = d2), xlim = c(-3, 11))
autoplot(d2)
autoplot(d2, type = 'distribution')
```

crossprod_inv *Inverse of $X'X$ by QR Decomposition*

Description

Compute $(X'X)^{-1}$ from the R part of the QR decomposition of X.

Usage

```
crossprod_inv(X)
```

Arguments

X	$m * n$ matrix

Value

crossprod_inv returns the inverse matrix of cross product $X'X$.

References

https://en.wikipedia.org/wiki/QR_decomposition, section **Rectangular matrix**

See Also

chol2inv chol.default

Examples

```
set.seed(123); (X = array(rnorm(40L), dim = c(8L, 5L)))
stopifnot(all.equal.numeric(solve(crossprod(X)), crossprod_inv(X)))
```

CvM_test *Cramer-Von Mises Test of Goodness-of-Fit for Distribution Estimates*

Description

Perform the Cramer-Von Mises test of goodness-of-fit for distribution estimates.

Usage

```
CvM_test(x, data, nullname, ...)
```

Arguments

x	an R object of distribution estimates
data	double vector, the actual observations
nullname, ...	additional parameters of cvm.test

Details

Note that we are currently not using the discrete version cvm.test.

Value

CvM_test returns an htest object, in which the element $statistic is the Cramer-Von Mises quadratic distance.

See Also

cvm.test

Examples

```
(d1 <- fmx('norm', mean = c(0, 1.5), sd = .5, w = c(.4, .6)))
x = rfmx(1e2L, dist = d1)
CvM_test(d1, data = x)
```

---

drop1_fmx *Add or Drop One Possible Parameters of fmx_QLMDe Object*

Description

Compute all the single terms in the scope argument that can be added to or dropped from the model, fit those models and compute a table of the changes in fit.

Usage

```
## S3 method for class 'fmx_QLMDe'
drop1(object, scope, test = c("logLik", "AIC", "BIC"), ...)

## S3 method for class 'fmx_QLMDe'
add1(object, scope, ...)
```

Arguments

object	fmx_QLMDe object
scope	a list of character vectors to denote one or more constraints
test	character, either 'logLik' (default), 'AIC' or 'BIC'
...	additional parameters, currently not in use.

Details

Do **not** write as S3 method of dropterm function, as there's no **term** for fmx_QLMDe object.

Value

drop1.fmx_QLMDe returns an ANOVA table with additional attributes

- `models` a list of fmx_QLMDe objects
- `objF` a list of objective functions (depends on `test`)
- `o1` the location of the optimal models by `test`. If the original model is optimal, this value is `integer()`

add1.fmx_QLMDe will be added in the next release.

See Also

add1 drop1

`fmx` *The Finite Mixture Distribution*

Description

Density function, distribution function, quantile function and random generation for a finite mixture distribution with normal or Tukey's g-&-h components.

Usage

```
fmx(distname, w = 1, ...)

dfmx(
  x,
  dist,
  distname = dist@distname,
  K = dim(parM)[1L],
  parM = dist@parM,
  w = dist@w,
  ...,
  log = FALSE
)

pfmx(
  q,
  dist,
  distname = dist@distname,
  K = dim(parM)[1L],
  parM = dist@parM,
```

```
  w = dist@w,
  ...,
  lower.tail = TRUE,
  log.p = FALSE
)

qfmx(
  p,
  dist,
  distname = dist@distname,
  K = dim(parM)[1L],
  parM = dist@parM,
  w = dist@w,
  interval = qfmx_interval(dist = dist),
  ...,
  lower.tail = TRUE,
  log.p = FALSE
)

rfmx(
  n,
  dist,
  distname = dist@distname,
  K = dim(parM)[1L],
  parM = dist@parM,
  w = dist@w
)
```

Arguments

`distname, K, parM, w`	auxiliary parameters, whose default values are determined by the fmx object provided in argument `dist`. The user-specified vector of `w` does not need to sum up to 1; `w/sum(w)` will be used internally.
`...`	mixture distribution parameters in fmx function. See dGH for the names and default values of Tukey's g-&-h distribution parameters, or dnorm for the names and default values of normal distribution parameters.
`x, q`	vector of quantiles, `NA_real_` value(s) allowed.
`dist`	fmx object, representing a finite mixture distribution
`log, log.p`	logical scalar. If `TRUE`, probabilities p are given as $\log(p)$.
`lower.tail`	logical scalar. If `TRUE` (default), probabilities are $Pr(X \leq x)$, otherwise, $Pr(X > x)$.
`p`	vector of probabilities.
`interval`	interval for root finding, see vuniroot
`n`	number of observations.

Details

A computational challenge in dfmx is when mixture density is very close to 0, which happens when the per-component log densities are negative with big absolute values. In such case, we cannot compute the log mixture densities (i.e., `-Inf`), for the log-likelihood using logLik.fmx function. Our solution is to replace these `-Inf` log mixture densities by the weighted average (using the mixing proportions of `dist`) of the per-component log densities.

qfmx gives the quantile function, by numerically solving the pfmx function. One major challenge when dealing with the finite mixture of Tukey's g-&-h family distribution is that Brent–Dekker's method needs to be performed in both pGH and qfmx functions, i.e. *two layers* of root-finding algorithm.

Value

fmx returns an fmx object which specifies the parameters of a finite mixture distribution.

dfmx returns a vector of probability density values of an fmx object at specified quantiles `x`.

pfmx returns a vector of cumulative probability values of an fmx object at specified quantiles `q`.

qfmx returns an unnamed vector of quantiles of an fmx object, based on specified cumulative probabilities `p`. Note that qnorm returns an unnamed vector of quantiles, although quantile returns a named vector of quantiles.

rfmx generates random deviates of an fmx object.

Examples

```
# paramter is recycled
fmx('norm', mean = c(4, 1, 14, 11), w = c(1, 2))

x = (-3):7

(e1 = fmx('norm', mean = c(0,3), sd = c(1,1.3), w = c(1, 1)))
isS4(e1) # TRUE
slotNames(e1)
autoplot(e1)
hist(rfmx(n = 1e3L, dist = e1), main = '1000 obs from e1')
# generate a sample of size 1e3L from mixture distribution `e1`
round(dfmx(x, dist = e1), digits = 3L)
round(p1 <- pfmx(x, dist = e1), digits = 3L)
stopifnot(all.equal.numeric(qfmx(p1, dist = e1), x, tol = 1e-4))

(e2 = fmx('GH', A = c(0,3), g = c(.2, .3), h = c(.2, .1), w = c(2, 3)))
hist(rfmx(n = 1e3L, dist = e2), main = '1000 obs from e2')
round(dfmx(x, dist = e2), digits = 3L)
round(p2 <- pfmx(x, dist = e2), digits = 3L)
stopifnot(all.equal.numeric(qfmx(p2, dist = e2), x, tol = 1e-4))

(e3 = fmx('GH', A = 0, g = .2, h = .2)) # one-component Tukey
hist(rfmx(1e3L, dist = e3))
hist(rGH(n = 1e3L, A = 0, g = .2, h = .2))
# identical (up to random seed); but ?rfmx has much cleaner code
```

```
round(dfmx(x, dist = e3), digits = 3L)
round(p3 <- pfmx(x, dist = e3), digits = 3L)
stopifnot(all.equal.numeric(qfmx(p3, dist = e3), x, tol = 1e-4))

if (FALSE) {
  # log-mixture-density smoothing, for developers
  (e4 = fmx('norm', mean = c(0,3), w = c(2, 3)))
  curve(dfmx(x, dist = e4, log = TRUE), xlim = c(-50, 50))
}
```

---

fmx-class *Specification of fmx Class*

Description

Parameter specification for a one-dimensional finite mixture distribution.

Slots

`distname` character scalar, name of parametric distribution of the mixture components. Currently, normal (`'norm'`) and Tukey's g-&-h (`'GH'`) distributions are supported.

`parM` double matrix, all distribution parameters in the mixture. Each row corresponds to one component. Each column includes the same parameters of all components. The order of rows corresponds to the (non-strictly) increasing order of the component location parameters. The column names match the formal arguments of the corresponding distribution, e.g., `mean` and `sd` for norm distribution (see `dnorm`), or `A`, `B`, `g` and `h` for Tukey's g-&-h distribution (see `dGH`).

`w` numeric vector of mixing proportions that must sum to 1

---

fmx_constraint *Parameter Constraint(s) of Mixture Distribution*

Description

Determine the parameter constraint(s) of a finite mixture distribution, either by the value of parameters of such mixture distribution, or by a user-specified string.

Usage

```
fmx_constraint(
  dist,
  distname = dist@distname,
  K = dim(dist@parM)[1L],
  parM = dist@parM
)
```

```
fmx_constraint_user(distname, K, user)

fmx_constraint_brief(dist)
```

Arguments

dist	an fmx object, can be missing
distname	character scalar, name of distribution
K	integer scalar, number of components
parM	double matrix, distribution parameters of a finite mixture distribution as slot @parM of fmx object.
user	character vector, constraint(s) to be imposed for an fmx object. For example, for a two-component Tukey's g-&-h mixture, user = c('g2', 'h1') indicates the g-parameter for the first component (with smaller mean value) and the h-parameter for the second component (with larger mean value) are to be constrained, i.e., $g_2 = h_1 = 0$.

Value

fmx_constraint returns the indexes of internal parameters (only applicable to Tukey's g-&-h mixture distribution, yet) to be constrained, based on the input fmx object dist.

fmx_constraint_user returns the indexes of internal parameters (only applicable to Tukey's g-&-h mixture distribution, yet) to be constrained, based on the type of distribution (distname), number of components (K) and a user-specified string (e.g., c('g2', 'h1')).

fmx_constraint_brief returns a character scalar (of LaTeX expression) of the constraint, primarily intended for end-users in plots.

Examples

```
(d0 = fmx('GH', A = c(1,4), g = c(.2,.1), h = c(.05,.1), w = c(1,1)))
(c0 = fmx_constraint(d0))
stopifnot(identical(c0, fmx_constraint_user(distname = 'GH', K = 2L, user = character())))
fmx_constraint_brief(d0)

(d1 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(0,.1), w = c(1,1)))
(c1 = fmx_constraint(d1))
stopifnot(identical(c1, fmx_constraint_user(distname = 'GH', K = 2L, user = c('g2', 'h1'))))
fmx_constraint_brief(d1)

(d2 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(.15,.1), w = c(1,1)))
(c2 = fmx_constraint(d2))
stopifnot(identical(c2, fmx_constraint_user(distname = 'GH', K = 2L, user = 'g2')))
fmx_constraint_brief(d2)

fmx_constraint_brief(fmx('norm', mean = c(0, 1)))
```

fmx_QLMDe-class *Specification of fmx_QLMDe Class*

Description

Quantile least Mahalanobis distance estimates of one-dimensional finite mixture distribution. The fmx_QLMDe object contains (i.e., inherits from) the fmx object.

Slots

`data` numeric vector, the one-dimensional observations

`data.name` character scalar, a human-friendly name of observations

`epdf` empirical probability density function fitted by approxfun

`quantile_vv` variance-covariance matrix of selected quantiles (based on the selected probabilities stored in slot `@p`)

`vcov` variance-covariance matrix of the internal (i.e., unconstrained) estimates

`init` fmx object, the initial values to be sent to optim

`p` numeric vectors of probabilities, where the distance between the empirical and true quantiles are measured

`optim` a list returned from optim

K.fmx *Number of Components in fmx and fmx_QLMDe Object*

Description

Obtain the number of components in fmx and fmx_QLMDe object.

Usage

```
K.fmx(x)
```

Arguments

`x`	fmx and fmx_QLMDe object.

Details

For user convenience

Value

An integer value indicating the number of components in an fmx and.or fmx_QLMDe object.

Examples

```
(d2 = fmx('GH', A = c(1,6), B = 2, g = c(0,.3), h = c(.2,0), w = c(1,2)))
K.fmx(d2)
```

KL_dist *Kullback-Leibler Divergence for Distribution Estimates*

Description

Calculate the Kullback-Leibler divergence for distribution estimates.

Usage

```
KL_dist(x, base, ...)
```

Arguments

x	an R object of distribution estimates
base	see KLD
...	additional parameters, currently not in use

Value

KL_dist returns a list, which is returned from KLD function.

See Also

KLD

---

ks_test *Kolmogorov-Smirnov Tests for Distribution Estimates*

Description

Perform the Kolmogorov-Smirnov tests for various distribution estimates.

Usage

```
ks_test(x, data, ...)
```

Arguments

x	an R object of distribution estimates
data	double vector, the actual observations
...	additional parameters of ks.test

Value

ks_test returns an htest object, in which the element $statistic is the Kolmogorov–Smirnov distance.

See Also

ks.test

---

letterValue *Letter-Value Estimation of Tukey g-&-h Distribution*

Description

Letter-value based estimation (Hoaglin, 2006) of Tukey g-&-h distribution and its constrained versions (g-distribution, h-distribution).

All equation numbers mentioned below refer to Hoaglin (2006).

Usage

```
letterValue(
  x,
  p_g = seq.int(from = 0.15, to = 0.25, by = 0.005),
  p_h = seq.int(from = 0.15, to = 0.35, by = 0.005),
  halfSpread = c("both", "lower", "upper"),
  ...
)

letterV_B_g_h(A, g, p_h, x, halfSpread, ...)

letterV_B_h(A, p_h, x, halfSpread)

letterV_B(A, g, p_g, x, halfSpread)

letterV_g(A, p_g, x)
```

Arguments

x	double vector, one-dimensional observations
p_g	double vector, the probabilities used for estimating parameter g. Or, use p_g = FALSE to implement the constraint $g = 0$.
p_h	double vector, the probabilities used for estimating parameter h. Or, use p_h = FALSE to implement the constraint $h = 0$.
halfSpread	character scalar, either to use 'both' half-spreads (default), 'lower' half-spread, or 'upper' half-spread.
...	additional parameters, currently not in use
A, g	estimated mean $\hat{A}$ and skewness $\hat{g}$ of Tukey's g-&-h distribution

Details

letterV_g estimates parameter g using equation (10).

letterV_B estimates parameter B for Tukey's g-distribution i.e., when $h = 0$ and $g \neq 0$, using equation (8a) and (8b).

letterV_B_g_h estimates parameters B and h when $g \neq 0$, using equation (33).

letterV_B_h estimates parameters B and h for Tukey's h-distribution, i.e., when $g = 0$ and $h \neq 0$, using equation (26a), (26b) and (27).

`letterValue` plays a similar role as `fitdistrplus:::start.arg.default`, thus extends fitdist for estimating Tukey's g-&-h distributions.

Value

letterValue returns a double vector of estimates $(\hat{A}, \hat{B}, \hat{g}, \hat{h})$ for a Tukey's g-&-h distribution.

References

Hoaglin, D.C. (2006). Summarizing Shape Numerically: The g-and-h Distributions. In *Exploring Data Tables, Trends, and Shapes* (eds D.C. Hoaglin, F. Mosteller and J.W. Tukey), Wiley Series in Probability and Statistics. doi:10.1002/9781118150702.ch11

See Also

fitdist

Examples

```
set.seed(77652); x = rGH(n = 1e3L, g = -.3, h = .1)
letterValue(x, p_g = FALSE, p_h = FALSE)
letterValue(x, p_g = FALSE)
letterValue(x, p_h = FALSE)

(y0 = letterValue(x))
library(fitdistrplus)
fit <- fitdist(x, distr = 'GH', start = as.list.default(y0))
autoplot(fit)
```

`LikRatio` *Likelihood Ratio Test for General Models*

Description

Likelihood ratio test for models fitted by R.

Usage

```
LikRatio(dots, type = c("plain", "vuong"), compare = c("seq", "first"), ...)
```

Arguments

`dots`	a list of regression models, or a list of logLik objects.
`type`	character scalar, ordinary likelihood ratio test (`'plain'`, default) or Vuong's closeness test for non-nested models (`'vuong'`).
`compare`	type of comparison between the models, sequentially (`'seq'`, default) or all models versus the first model (`'first'`)
`...`	additional arguments of logLik function(s)

Value

LikRatio returns an ANOVA table for likelihood ratios test, or a `'vuong'` object for Vuong's test.

References

Vuong's closeness test, doi:10.2307/1912557.

See Also

lrtest.default

Examples

```
# no examples for now
```

mahalanobis_int *A Simpler and Faster Mahalanobis Distance*

Description

A simpler and faster Mahalanobis distance.

Usage

```
mahalanobis_int(x, center, invcov)
```

Arguments

x	numeric vector
center	numeric vector, mean μ
invcov	numeric matrix, *inverted* variance-covariance $\mathbf{\Sigma}$

Value

mahalanobis_int returns a numeric scalar.

See Also

mahalanobis

mlogis *Transformation between Multinomial Probabilities & Logits*

Description

Performs transformation between vectors of multinomial probabilities and multinomial logits.

This transformation is a generalization of plogis which converts scalar logit into probability and qlogis which converts probability into scalar logit.

Usage

```
qmlogis_first(p)

qmlogis_last(p)

pmlogis_first(q)

pmlogis_last(q)
```

Arguments

p	numeric vector of multinomial probabilities, adding up to 1
q	numeric vector of multinomial logits

Details

pmlogis_first and pmlogis_last take a length $k-1$ numeric vector of multinomial logits q and convert them into length k multinomial probabilities p, regarding the first or last category as reference, respectively.

qmlogis_first and qmlogis_last take a length k numeric vector of multinomial probabilities p and convert them into length $k-1$ multinomial logits q, regarding the first or last category as reference, respectively.

Value

pmlogis_first and pmlogis_last return a vector of multinomial probabilities p.

qmlogis_first and qmlogis_last returns a vector of multinomial logits q.

See Also

plogis qlogis

Examples

```
(a = qmlogis_last(c(2,5,3)))
(b = qmlogis_first(c(2,5,3)))
pmlogis_last(a)
pmlogis_first(b)

q0 = .8300964
(p1 = pmlogis_last(q0))
(q1 = qmlogis_last(p1))

# various exceptions
pmlogis_first(qmlogis_first(c(1, 0)))
pmlogis_first(qmlogis_first(c(0, 1)))
pmlogis_first(qmlogis_first(c(0, 0, 1)))
pmlogis_first(qmlogis_first(c(0, 1, 0, 0)))
pmlogis_first(qmlogis_first(c(1, 0, 0, 0)))
pmlogis_last(qmlogis_last(c(1, 0)))
pmlogis_last(qmlogis_last(c(0, 1)))
pmlogis_last(qmlogis_last(c(0, 0, 1)))
pmlogis_last(qmlogis_last(c(0, 1, 0, 0)))
pmlogis_last(qmlogis_last(c(1, 0, 0, 0)))
```

`outer_allequal` *Test if Two double Vectors are Element-Wise (Nearly) Equal*

Description

Test if two double vectors are element-wise (nearly) equal.

Usage

```
outer_allequal(target, current, tolerance = sqrt(.Machine$double.eps), ...)
```

Arguments

`target`	length-n double vector, the target value, missing value not allowed
`current`	length-n double vector, the value to be compared with `target`, missing value not allowed
`tolerance`	double scalar, see all.equal.numeric.
`...`	potential parameters, currently not in use

Details

outer_allequal is different from all.equal.numeric, such that (1). only comparisons between real double values are performed; (2). element-wise comparison is performed, with the rows of returned matrix correspond to `current` and columns correspond to `target`; (3). a logical scalar is returned for each element-wise comparison.

Value

outer_allequal returns an $m * n$ logical matrix indicating whether the length-n vector `current` is element-wise near-equal to the length-m vector `target` within the prespecified `tolerance`.

See Also

all.equal.numeric outer

Examples

```
x = c(.3, 1-.7, 0, .Machine$double.eps)
outer_allequal(current = x, target = c(.3, 0))
```

QLMDe *Quantile Least Mahalanobis Distance estimates*

Description

The quantile least Mahalanobis distance algorithm estimates the parameters of single-component or finite mixture distributions by minimizing the Mahalanobis distance between the vectors of sample and theoretical quantiles. See QLMDp for the default selection of probabilities at which the sample and theoretical quantiles are compared.

The default initial values are estimated based on trimmed k-means clustering with re-assignment.

Usage

```
QLMDe(
  x,
  distname = c("norm", "GH"),
  K,
  data.name = deparse1(substitute(x)),
  constraint = character(),
  p = QLMDp(x = x),
  init = c("logLik", "letterValue", "normix"),
  tol = .Machine$double.eps^0.25,
  maxiter = 1000,
  ...
)
```

Arguments

x	numeric vector, the one-dimensional observations.
distname	character value, name of mixture distribution to be fitted. Currently supports 'norm' and 'GH'.
K	integer scalar, number of components (e.g., must use 2L instead of 2).
data.name	character value, name for the observations for user-friendly print out.
constraint	character vector, parameters (g and/or h for Tukey's g-&-h mixture) to be set at 0. See fmx_constraint for details.
p	numeric vector, percentiles at where the sample and theoretical quantiles are to be matched. See QLMDp for details.
init	character scalar for the method of initial values selection, or an fmx object of the initial values. See QLMDinit for more details.
tol, maxiter	see vuniroot2
...	additional parameters of optim.

Details

Quantile Least Mahalanobis Distance estimator fits a single-component or finite mixture distribution by minimizing the Mahalanobis distance between the theoretical and observed quantiles, using the empirical quantile variance-covariance matrix quantile_vcov.

Value

An fmx_QLMDe object

See Also

optim QLMDinit

Examples

```
# Generated from 1-component normal; start with 2-component normal fit
set.seed(1623); (y0n <- QLMDe(rnorm(1e3L), distname = 'norm', K = 2L))
(y1n <- StepK_fmx(y0n, test = 'logLik', Kmax = 2L)) # one-component
vcov(y1n)

# Generated from 2-component normal; start with 1-component normal fit
(d1 <- fmx('norm', mean = c(0, 1.5), sd = .5, w = c(.4, .6)))
set.seed(100); hist(x1 <- rfmx(n = 1e3L, dist = d1))
StepK_fmx(QLMDe(x1, distname = 'norm', K = 1L), test = 'logLik', Kmax = 2L)


(d2 = fmx('GH', A = c(1,6), B = 1.2, g = c(0,.3), h = c(.2,0), w = c(1,2)))
set.seed(3123); hist(x2 <- rfmx(n = 1e3L, dist = d2))
# using user-specified constraint
system.time(QLMDe(x2, distname = 'GH', K = 2L, constraint = c('g1', 'h2')))

# using Step_fmx
system.time(y2gh <- QLMDe(x2, distname = 'GH', K = 2L)) # ~2 secs
y2gh
ks_test(y2gh)
CvM_test(y2gh)
KL_dist(y2gh)
Step_fmx(y2gh, test = 'logLik') # identified true constraint :)



system.time(y1gh <- QLMDe(x2, distname = 'GH', K = 1L))
y1gh
StepK_fmx(y1gh, test = 'logLik', Kmax = 2L) # correct

#set.seed(1323); x3 <- rGH(n = 1e3L, g = .2, h = .1)
#system.time(tmp <- QLMDe(x3, distname = 'GH', K = 2L)) # ~2 secs
#StepK_fmx(tmp, Kmax = 2L) # very difficult to drop K ..
```

QLMDinit *Initial Values for Quantile Least Mahalanobis Distance (QLMD) Estimates*

Description

Various methods for obtaining the initial values for Quantile Least Mahalanobis Distance (QLMD) estimates of finite mixture distribution fmx.

Usage

```
QLMDinit_letterValue(x, K, constraint = character(), alpha = 0.05, ...)

QLMDinit_normix(x, K, alpha = 0.05, R = 10L, ...)

QLMDinit(x, distname = c("GH", "norm"), test = c("logLik", "CvM", "KS"), ...)
```

Arguments

x	numeric vector, the actual observations
K	integer scalar, number of mixture components
constraint	character vector, parameters (g and/or h for Tukey's g-&-h mixture) to be set at 0. See fmx_constraint for details.
alpha	numeric scalar, proportion of observations to be trimmed by trimmed k-means algorithm tkmeans
...	additional arguments
R	integer scalar, number of normalmixEM replicates
distname	character scalar, name of parametric distribution
test	character scalar, criteria for selecting the optimal estimates. See **Details**.

Details

First of all, if the specified number of components $K \geq 2$, trimmed k-means clustering with reassignment will be performed; otherwise, all observations will be considered as one single cluster. The standard k-means clustering is not used since the heavy tails of Tukey's g-&-h distribution could be mistakenly classified as individual cluster(s). In each of the one or more clusters,

- The letter-value based estimates of Tukey's g-&-h distribution (Hoaglin, 2006) are calculated, for any $K \geq 1$, serving as the starting values for QLMD algorithm. These estimates are provided by QLMDinit_letterValue.
- the median and MAD will serve as the starting values for μ and σ (or A and B for Tukey's g-&-h distribution, with $g = h = 0$), for QLMD algorithm when $K = 1$. Otherwise, the cluster centers are provided as the starting values of μ's for the univariate normal mixture by EM algorithm. R replicates of normal mixture estimates are obtained, and the one with maximum likelihood will serve as the starting values for QLMD algorithm. These estimates are provided by QLMDinit_normix.

QLMDinit compares the Tukey's g-&-h mixture estimate provided by QLMDinit_letterValue and the normal mixture estimate by QLMDinit_normix, and select the one either with maximum likelihood (test = 'logLik', default), with minimum Cramer-von Mises distance (test = 'CvM') or with minimum Kolmogorov-Smirnov distance (test = 'KS').

Value

QLMDinit_letterValue, QLMDinit_normix and QLMDinit all return fmx objects.

See Also

kmeans tkmeans reAssign letterValue normalmixEM

Examples

```
d1 = fmx('norm', mean = c(1, 2), sd = .5, w = c(.4, .6))
set.seed(100); hist(x1 <- rfmx(n = 1e3L, dist = d1))
QLMDinit_normix(x1, distname = 'norm', K = 2L)

(d2 = fmx('GH', A = c(1,6), B = 2, g = c(0,.3), h = c(.2,0), w = c(1,2)))
set.seed(100); hist(x2 <- rfmx(n = 1e3L, dist = d2))
QLMDinit_letterValue(x2, K = 2L)
QLMDinit_letterValue(x2, K = 2L, constraint = c('g1', 'h2'))
QLMDinit_normix(x2, K = 2L)
QLMDinit(x2, distname = 'GH', K = 2L)
```

QLMDp *Percentages for Quantile Least Mahalanobis Distance estimation*

Description

A vector of probabilities to be used in Quantile Least Mahalanobis Distance estimation (QLMDe).

Usage

```
QLMDp(
  from = 0.05,
  to = 0.95,
  length.out = 15L,
  equidistant = c("prob", "quantile"),
  extra = c(0.005, 0.01, 0.02, 0.03, 0.97, 0.98, 0.99, 0.995),
  x
)
```

Arguments

`from`	numeric scalar, minimum of the equidistant (in probability or quantile) probabilities. Default `.05`.
`to`	numeric scalar, maximum of the equidistant (in probability or quantile) probabilities. Default `.95`.
`length.out`	non-negative integer scalar, the number of the equidistant (in probability or quantile) probabilities.
`equidistant`	character scalar. If `'prob'` (default), then the probabilities are equidistant. If `'quantile'`, then the quantiles (of the observations `x`) corresponding to the probabilities are equidistant.
`extra`	numeric vector of *additional* probabilities, default `c(.005, .01, .02, .03, .97, .98, .99, .995)`.
`x`	numeric vector of observations, only used when `equidistant = 'quantile'`.

Details

The default arguments of QLMDp returns the probabilities of `c(.005, .01, .02, .03, seq.int(.05, .95, length.out = 15L), .97, .98, .99, .995)`.

Value

A numeric vector of probabilities to be supplied to parameter `p` of Quantile Least Mahalanobis Distance QLMDe estimation). In practice, the length of this probability vector `p` must be equal or larger than the number of parameters in the distribution model to be estimated.

Examples

```
(d2 = fmx('GH', A = c(1,6), B = 2, g = c(0,.3), h = c(.2,0), w = c(1,2)))
set.seed(100); hist(x2 <- rfmx(n = 1e3L, dist = d2))
p_hist = geom_histogram(
  mapping = aes(x = x2, y = ..density..), bins = 30L, colour = 'white', alpha = .1
)

(p1 = QLMDp()) # equidistant in probabilities
autoplot(d2, v = setNames(qfmx(p1, dist = d2), nm = sprintf('%.1f%%', 1e2*p1)))
autoplot(d2, v = quantile(x2, probs = p1, digits = 3L)) + p_hist

(p2 = QLMDp(equidistant = 'quantile', x = x2)) # equidistnat in quantiles
autoplot(d2, v = quantile(x2, probs = p2, digits = 3L)) + p_hist
```

`quantile_vcov` *Variance-Covariance of Quantiles*

Description

Computes the variance-covariance matrix of quantiles based on Theorem 1 and 2 of Mosteller (1946).

Usage

```
quantile_vcov(p, d)
```

Arguments

`p`	numeric vector of cumulative probabilities at the given quantiles
`d`	numeric vector of probability densities at the given quantiles

Details

The end user should make sure no densities too close to 0 is included in argument `d`.

quantile_vcov must not be used in a compute-intensive way.

Value

The variance-covariance matrix of quantiles based on Mosteller (1946).

References

Frederick Mosteller. On Some Useful "Inefficient" Statistics. *The Annals of Mathematical Statistics*, 17 (4) 377-408, December, 1946. doi:10.1214/aoms/1177730881

`reAssign` *Re-Assign Observations Trimmed Prior to Trimmed k-Means Clustering*

Description

Re-assign the observations, which are trimmed in the trimmed k-means algorithm, back to the closest cluster as determined by the smallest Mahalanobis distance.

Usage

```
reAssign(x, ...)

## S3 method for class 'tkmeans'
reAssign(x, ...)
```

Arguments

x	a tkmeans object
...	potential parameters, currently not in use.

Details

Given the tkmeans input, the Mahalanobis distance is computed between each trimmed observation and each cluster. Each trimmed observation is assigned to the closest cluster (i.e., with the smallest Mahalanobis distance).

Value

An 'reAssign_tkmeans' object, which inherits from tkmeans class.

See Also

tkmeans

Examples

```
library(tclust)
data(geyser2)
clus = tkmeans(geyser2, k = 3L, alpha = .03)
plot(clus, main = 'Before Re-Assigning')
plot(reAssign(clus), main = 'After Re-Assigning')
```

S3_fmx_QLMDe *Inference for Quantile Least Mahalanobis Distance estimates*

Description

Additional methods of class fmx and/or fmx_QLMDe, for generic functions defined in **stats** package.

Usage

```
## S3 method for class 'fmx_QLMDe'
vcov(object, internal = FALSE, ...)

## S3 method for class 'fmx'
coef(object, internal = FALSE, ...)

## S3 method for class 'fmx_QLMDe'
confint(object, ..., level = 0.95)

## S3 method for class 'fmx'
```

```
logLik(object, data = object@data, ...)

## S3 method for class 'fmx_QLMDe'
nobs(object, ...)
```

Arguments

`object`	an fmx or fmx_QLMDe object
`internal`	logical scalar, either for the user-friendly parameters (`FALSE`, default) (e.g., `mean,sd` for normal mixture, and `A,B,g,h` for Tukey's g-and-h mixture), or for the internal/unconstrained parameters (`TRUE`).
`...`	place holder for S3 naming convention
`level`	confidence level, default 95%.
`data`	double vector, actual observations

Details

The inference for the user-friendly parameters is obtained via delta-method.

Value

nobs.fmx_QLMDe returns an integer scalar indicating the sample size of the observations used in QLMDe estimation.

logLik.fmx returns a logLik object indicating the log-likelihood. An additional attribute `attr(, 'logl')` indicates the point-wise log-likelihood, to be use in Vuong's closeness test.

coef.fmx returns the estimates of the user-friendly parameters (`parm = 'user'`), or the internal/unconstrained parameters (`parm = 'internal'`).

vcov.fmx_QLMDe returns the approximate asymptotic variance-covariance matrix of the user-friendly parameters via delta-method (`parm = 'user'`), or the asymptotic variance-covariance matrix of the internal/unconstrained parameters (`parm = 'internal'`).

confint.fmx_QLMDe returns the Wald-type confidence intervals based on the user-friendly parameters (`parm = 'user'`), or the internal/unconstrained parameters (`parm = 'internal'`).

When the distribution has constraints on one or more parameters, none of `coef.fmx`, vcov.fmx_QLMDe and confint.fmx_QLMDe return the corresponding values only for the constrained parameters.

See Also

nobs logLik coef vcov confint

show,fmx-method *Show fmx and/or fmx_QLMDe Object*

Description

Print the parameters of an fmx object and plot its density curves.

Usage

```
## S4 method for signature 'fmx'
show(object)
```

Arguments

object	an fmx or fmx_QLMDe object

Value

The show method for fmx and/or fmx_QLMDe object does not have a returned value.

StepK_fmx *Forward-Backward Selection of the Number of Components K*

Description

To compare gh-parsimonious models with different number of components K and select the optimal model using the Vuong's closeness test.

Usage

```
StepK_fmx(
  object,
  test = c("logLik", "AIC", "BIC"),
  Kmax = stop("must specify maximum `Kmax`"),
  ...
)
```

Arguments

object	fmx_QLMDe object
test	see parameter test of Step_fmx function
Kmax	integer scalar K_M, the maximum number of components to be considered
...	additional parameters

Details

StepK_fmx compares the gh-parsimonious models with different number of components, and selects the optimal model using the Vuong's closeness test.

The forward-backward selection starts with finding the gh-parsimonious model at a user-specified initial number of components $K = K_0$ (as reflected in the input `object`).

The forward selection compares the gh-parsimonious models at $K_0 + 1$ and at K_0 component, respectively, using the Vuong's closeness test. If K_0 component is preferred, then the forward-backward selection is stopped if $K_0 = 1$, otherwise (if $K_0 > 1$) switches to the backward selection. If $K_0 + 1$ component is preferred, then the algorithm is stopped if $K_0 + 1 = K_M$ (prespecified maximum number of components), otherwise (if $K_0 + 1 < K_M$) $K_0 + 2$ versus $K_0 + 1$ component is compared.

The backward selection is performed only if K_0 component is preferred over $K_0 + 1$ component. The gh-parsimonious model at $K_0 - 1$ and at K_0 component, respectively, is compared. If K_0 component is preferred, then the forward-backward selection is stopped. If $K_0 - 1$ component is preferred, then the forward-backward selection is stopped if $K_0 - 1 = 1$, otherwise (if $K_0 - 1 > 1$) $K_0 - 2$ versus K_0 (**not** $K_0 - 1$) component is compared.

Value

StepK_fmx returns an fmx_QLMDe object, with attributes

- `anova` ANOVA table
- `objF` value of the objective function (either the log-likelihood, AIC or BIC)

Examples

```
(d = fmx('norm', mean = c(1, 4, 8), w = c(3, 3, 4)))
x = rfmx(n = 1e3L, dist = d)
y1 = QLMDe(x, distname = 'norm', K = 1L)
StepK_fmx(y1, Kmax = 3L)

if (FALSE) {
# slow, but works
(d = fmx('GH', A = c(0, 3), g = c(.2, -.3), h = c(.2, .2), w = c(6, 4)))
x = rfmx(n = 1e3L, dist = d)
(y1 = QLMDe(x, distname = 'GH', K = 1L))
StepK_fmx(y1, Kmax = 3L)
}
```

`Step_fmx` *Backward Selection gh-parsimonious Model with Fixed Number of Components K*

Description

Step_fmx selects a gh-parsimonious model with g and/or h parameters equal to zero for all or some of the mixture components conditionally on fixed number of components K.

Usage

```
Step_fmx(object, test = c("logLik", "AIC", "BIC"), ...)
```

Arguments

`object`	fmx_QLMDe object
`test`	character scalar indicating the criterion to be used, either via likelihood ratio test 'logLik' (default and recommended) via Akaike's information criterion 'AIC' and via Bayesian information criterion 'BIC'.
`...`	additional parameters

Details

The algorithm starts with quantile least Mahalanobis distance estimates of either the full mixture of Tukey g-&-h distributions model, or a constrained model (i.e., some g and/or h parameters equal to zero according to the user input). Next, each of the non-zero g and/or h parameters is tested using the likelihood ratio test. If all tested g and/or h parameters are significantly different from zero at the level 0.05 the algorithm is stopped and the initial model is considered gh-parsimonious. Otherwise, the g or h parameter with the largest p-value is constrained to zero for the next iteration of the algorithm.

The algorithm iterates until only significantly-different-from-zero g and h parameters are retained, which corresponds to gh-parsimonious Tukey's g-&-h mixture model.

Value

Step_fmx returns an fmx_QLMDe object, with attributes

- `anova` ANOVA table
- `objF` value of the objective function (either the log-likelihood, AIC or BIC)

See Also

LikRatio

TukeyGH *Tukey's g-&-h Distribution*

Description

Density, distribution function, quantile function and random generation for the Tukey's g-&-h distribution with location parameter A, scale parameter B, skewness g and kurtosis h.

Usage

```
dGH(x, A = 0, B = 1, g = 0, h = 0, log = FALSE, ...)

rGH(n, A = 0, B = 1, g = 0, h = 0)

qGH(p, A = 0, B = 1, g = 0, h = 0, lower.tail = TRUE, log.p = FALSE)

pGH(q, A = 0, B = 1, g = 0, h = 0, lower.tail = TRUE, log.p = FALSE, ...)

z2qGH(z, A = 0, B = 1, g = 0, h = 0)

qGH2z(q, q0 = (q - A)/B, A = 0, B = 1, ...)
```

Arguments

x, q	double vector, quantiles
A	double scalar, location parameter A, default $A = 0$ (as parameter mean of dnorm function)
B	double scalar, scale parameter $B > 0$, default $B = 1$ (as parameter sd of dnorm function)
g	double scalar, skewness parameter g, default $g = 0$ indicating no skewness
h	double scalar, kurtosis parameter $h \geq 0$, default $h = 0$ indicating no kurtosis
log, log.p	logical scalar, if TRUE, probabilities p are given as $\log(p)$.
...	other parameters of vuniroot2
n	integer scalar, number of observations
p	double vector, probabilities
lower.tail	logical scalar, if TRUE (default), probabilities are $Pr(X \leq x)$ otherwise, $Pr(X > x)$.
z	double vector, standard normal quantiles.
q0	double vector of $(q - A)/B$, for internal use to increase compute speed

Details

Argument A, B, g and h will be recycled to the maximum length of the four.

Value

dGH gives the density and accommodates vector arguments A, B, g and h. The quantiles x can be either vector or matrix. This function takes about 1/5 time of dgh function.

pGH gives the distribution function, only taking scalar arguments and vector quantiles q. This function takes about 1/10 time of pgh and pgh functions.

qGH gives the quantile function, only taking scalar arguments and vector probabilities p. This function takes about 1/2 time of qgh and 1/10 time of qgh functions.

rGH generates random deviates, only taking scalar arguments.

z2qGH is the Tukey's g-&-h transformation. Note that gk:::z2gh is only an **approximation** to Tukey's g-&-h transformation.

Unfortunately, qGH2z function, the inverse of Tukey's g-&-h transformation, does not have a closed form and needs to be solved numerically.

See Also

dgh dgh

Examples

```
(x = c(NA_real_, rGH(n = 5L, g = .3, h = .1)))
dGH(x, g = c(0,.1,.2), h = c(.1,.1,.1))

p0 = seq.int(0, 1, by = .2)
(q0 = qGH(p0, g = .2, h = .1))
range(pGH(q0, g = .2, h = .1) - p0)

q = (-2):3; q[2L] = NA_real_; q
(p1 = pGH(q, g = .3, h = .1))
range(qGH(p1, g = .3, h = .1) - q, na.rm = TRUE)
(p2 = pGH(q, g = .2, h = 0))
range(qGH(p2, g = .2, h = 0) - q, na.rm = TRUE)

curve(dGH(x, g = .3, h = .1), from = -2.5, to = 3.5)
```

---

ud_allequal *Determine Nearly-Equal Elements*

Description

Determine nearly-equal elements and extract non-nearly-equal elements in a double vector.

Usage

```
unique_allequal(x, ...)

duplicated_allequal(x, ...)
```

Arguments

x	double vector
...	additional parameters of outer_allequal

Value

duplicated_allequal returns a logical vector of the same length as the input vector, indicating whether each element is nearly-equal to any of the previous elements.

unique_allequal returns the non-nearly-equal elements in the input vector.

See Also

duplicated.default unique.default

Examples

```
x = c(.3, 1-.7, 0, .Machine$double.eps)
unique.default(x) # not desired
unique_allequal(x) # desired
```

---

vuniroot2 *Vectorised One Dimensional Root (Zero) Finding*

Description

To solve a monotone function $y = f(x)$ for a given vector of y values.

Usage

```
vuniroot2(
  y,
  f,
  interval = stop("must provide a length-2 `interval`"),
  tol = .Machine$double.eps^0.25,
  maxiter = 1000L
)
```

Arguments

y	numeric vector of y values
f	monotone function $f(x)$ whose roots are to be solved
interval	length two numeric vector
tol	double scalar, desired accuracy (convergence tolerance),
maxiter	integer scalar, maximum number of iterations

Details

vuniroot2 function, different from vuniroot function, does

- accept `NA_real_` as element(s) of y
- handle the case when the analytical root is at `lower` and/or `upper`
- return a root of `Inf` (if `abs(f(lower)) >= abs(f(upper))`) or `-Inf` (if `abs(f(lower)) < abs(f(upper))`), when the function value `f(lower)` and `f(upper)` are not of opposite sign.

Value

vuniroot2 returns a numeric vector x as the solution of $y = f(x)$ with given vector y.

See Also

vuniroot

Examples

```
library(rstpm2)
lwr = rep(1, times = 9L); upr = rep(3, times = 9L)

# ?rstpm2::vuniroot does not accept NA \eqn{y}
tryCatch(vuniroot(function(x) x^2 - c(NA, 1:8), lower = lwr, upper = upr), error = identity)

# ?rstpm2::vuniroot not good when the analytic root is at `lower` or `upper`
f <- function(x) x^2 - 1:9
tryCatch(vuniroot(f, lower = lwr, upper = upr, extendInt = 'no'), warning = identity)
tryCatch(vuniroot(f, lower = lwr, upper = upr, extendInt = 'yes'), warning = identity)
tryCatch(vuniroot(f, lower = lwr, upper = upr, extendInt = 'downX'), error = identity)
tryCatch(vuniroot(f, lower = lwr, upper = upr, extendInt = 'upX'), warning = identity)

vuniroot2(c(NA, 1:9), f = function(x) x^2, interval = c(1, 3)) # all good
```

`[,fmx,ANY,ANY,ANY-method`

Subset of Components in fmx and/or fmx_QLMDe Object

Description

Taking subset of components in fmx and/or fmx_QLMDe object

Usage

```
## S4 method for signature 'fmx,ANY,ANY,ANY'
x[i, j, drop]
```

Arguments

x	fmx and/or fmx_QLMDe object
i	integer or logical vector, the row index(es) of the subset of components to be chosen, see [
j	ignored (always TRUE, i.e., all parameters of such give distribution must be selected), see [
drop	ignored (always FALSE), see [

Details

Note that using definitions as S3 method dispatch `[.fmx` or `[.fmx_QLMDe` won't work for fmx and/or fmx_QLMDe objects.

Value

An fmx object consisting of a subset of components. Note that subsetting fmx_QLMDe object will return an fmx object, which contains only the mixture parameters, i.e., information about the observations (e.g. slots @data and @data.name), as well as other estimation related slots (e.g., @init) will be lost.

Examples

```
(d = fmx('norm', mean = c(1, 5, 9)))
d[1:2, ]
```

Index