Package ‘QuantileGH’

October 16, 2023

Type Package

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

Version 0.1.5

Date 2023-10-15

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

Maintainer Tingting Zhan <`Tingting.Zhan@jefferson.edu`>

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

License GPL-2

Imports methods, goftest, latex2exp, mixtools, rstpm2, scales, tclust, VGAM, sn

Encoding UTF-8

Language en-US

VignetteBuilder knitr

LazyData true

LazyDataCompression xz

Depends R (>= 4.3.0), ggplot2

Suggests fitdistrplus, mixsmsn, knitr, rmarkdown

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-10-16 19:30:21 UTC

R topics documented:

approxdens 3
as.fmx 4
as.fmx.fitdist 5
as.fmx.mixEM 5
as.fmx.Normal 6
as.fmx.Skew.normal 7
as.fmx.Skew.t 8
as.fmx.t 9
autolayer_fmx_continuous 10
autolayer_fmx_discrete 11
autoplot.fmx 12
autoplot.fmxS 13
CK5 13
clusterList 14
coef.fmx 14
confint.fmx 15
crossprod_inv 16
CycD1 16
dbl2fmx 17
dfmx 17
distArgs 20
distType 20
dist_logtrans 21
drop1_fmx 21
fmx 22
fmx-class 23
fmx2dbl 24
fmxS 25
fmx_cluster 26
fmx_constraint 27
fmx_diagnosis 28
fmx_hybrid 29
fmx_normix 30
geom_function_args 30
getTeX 31
Kolmogorov_dist 32
letterValue 33
logLik.fitdist 35
logLik.fmx 35
logLik.mixEM 36
mahalanobis_int 36
mixEM_pars 37
mlogis 37
moment_ 39
nobs.fitdist 41
nobs.fmx 42

npar.fmx 43
outer_allequal 43
print.fmx 44
QLMDe 45
QLMDe_stepK 46
QLMDp 47
quantile_vcov 48
reAssign 49
show,fmx-method 50
show,moment-method 51
sort.mixEM 51
sort_mixsmsn 52
step_fmx 53
TukeyGH 54
ud_allequal 55
user_constraint 56
vcov.fmx 57
vuniroot2 58
[,fmx,ANY,ANY,ANY-method 59

Index **60**

`approxdens` *Empirical Density Function*

Description

..

Usage

```
approxdens(x, ...)
```

Arguments

`x`	numeric vector, observations
`...`	additional parameters of density.default

Details

approx inside density.default

another 'layer' of approxfun

Value

approxdens returns a function.

Examples

```
x = rnorm(1e3L)
f = approxdens(x)
f(x[1:3])
```

as.fmx *Turn Various Objects to fmx*

Description

Turn various objects that are created in other packages to fmx class

Usage

```
as.fmx(x, data, ...)
```

Arguments

`x`	an R object
`data`	numeric vector
`...`	..

Details

In order to take advantage of all methods for fmx objects

Value

S3 generic function `as.fmx()` returns an fmx object.

See Also

`as.fmx.fitdist()` `as.fmx.mixEM()`

`as.fmx.fitdist` *Convert fitdist Objects to fmx Objects*

Description

..

Usage

```
## S3 method for class 'fitdist'
as.fmx(x, data = x[["data"]], ...)
```

Arguments

`x`	fitdist object
`data`	numeric vector
`...`	..

Value

Function `as.fmx.fitdist()` returns an fmx object.

`as.fmx.mixEM` *Convert `mixEM` Objects to fmx Objects*

Description

..

Usage

```
## S3 method for class 'mixEM'
as.fmx(x, data = x[["x"]], ...)
```

Arguments

`x`	`mixEM` object
`data`	numeric vector
`...`	..

Value

Function `as.fmx.mixEM()` returns an fmx object.

Note

plot.mixEM not plot gammamixEM returns, as of 2022-09-19.

Examples

```
library(mixtools)
(x = as.fmx(normalmixEM(faithful$waiting, k = 2)))
```

`as.fmx.Normal` *Convert* `Normal` *fit from* R*hrefhttps://CRAN.R-project.org/package=mixsmsn***mixsmsn** *to fmx*

Description

..

Usage

```
## S3 method for class 'Normal'
as.fmx(x, data, ...)
```

Arguments

`x`	`'Normal'` object, returned from smsn.mix with parameter `family = 'Normal'`
`data`	numeric vector
`...`	additional parameters, currently not in use

Value

Function `as.fmx.Normal()` returns an fmx object.

Note

smsn.mix does not offer a parameter to keep the input data, as of 2021-10-06.

Examples

```
library(mixsmsn)
# ?smsn.mix
arg1 = c(mu = 5, sigma2 = 9, lambda = 5, nu = 5)
arg2 = c(mu = 20, sigma2 = 16, lambda = -3, nu = 5)
arg3 = c(mu = 35, sigma2 = 9, lambda = -6, nu = 5)
set.seed(120); x = rmix(n = 1e3L, p=c(.5, .2, .3), family = 'Skew.t',
  arg = list(unname(arg1), unname(arg2), unname(arg3)))

# Normal
```

```
class(m2 <- smsn.mix(x, nu = 3, g = 3, family = 'Normal', calc.im = FALSE))
mix.hist(y = x, model = m2)
m2a = as.fmx(m2, data = x)
autoplot(m2a)
```

`as.fmx.Skew.normal` *Convert* `Skew.normal` *fit from* R*hrefhttps://CRAN.R-project.org/package=mixsmsn***mixsmsn** *to fmx*

Description

..

Usage

```
## S3 method for class 'Skew.normal'
as.fmx(x, data, ...)
```

Arguments

`x`	`'Skew.normal'` object, returned from smsn.mix with parameter `family` = `'Skew.normal'`
`data`	numeric vector
`...`	additional parameters, currently not in use

Value

Function `as.fmx.Skew.normal()` returns an fmx object.

Note

smsn.mix does not offer a parameter to keep the input data, as of 2021-10-06.

Examples

```
library(mixsmsn)
# ?smsn.mix
arg1 = c(mu = 5, sigma2 = 9, lambda = 5, nu = 5)
arg2 = c(mu = 20, sigma2 = 16, lambda = -3, nu = 5)
arg3 = c(mu = 35, sigma2 = 9, lambda = -6, nu = 5)
set.seed(120); x = rmix(n = 1e3L, p=c(.5, .2, .3), family = 'Skew.t',
  arg = list(unname(arg1), unname(arg2), unname(arg3)))

# Skew Normal
class(m1 <- smsn.mix(x, nu = 3, g = 3, family = 'Skew.normal', calc.im = FALSE))
mix.hist(y = x, model = m1)
m1a = as.fmx(m1, data = x)
(l1a = logLik(m1a))
```

```
autoplot(m1a)
autoplot(m1a, type = 'distribution')
```

as.fmx.Skew.t *Convert* Skew.t *fit from* R*hrefhttps://CRAN.R-project.org/package=mixsmsn***mixsmsn** *to fmx*

Description

..

Usage

```
## S3 method for class 'Skew.t'
as.fmx(x, data, ...)
```

Arguments

x	'Skew.t' object, returned from smsn.mix with parameter family = 'Skew.t'
data	numeric vector
...	additional parameters, currently not in use

Value

Function `as.fmx.Skew.t()` returns an fmx object.

Note

smsn.mix does not offer a parameter to keep the input data, as of 2021-10-06.

Examples

```
# mixsmsn::smsn.mix with option `family = 'Skew.t'` is slow

library(mixsmsn)
# ?smsn.mix
arg1 = c(mu = 5, sigma2 = 9, lambda = 5, nu = 5)
arg2 = c(mu = 20, sigma2 = 16, lambda = -3, nu = 5)
arg3 = c(mu = 35, sigma2 = 9, lambda = -6, nu = 5)
set.seed(120); x = rmix(n = 1e3L, p=c(.5, .2, .3), family = 'Skew.t',
  arg = list(unname(arg1), unname(arg2), unname(arg3)))

# Skew t
class(m3 <- smsn.mix(x, nu = 3, g = 3, family = 'Skew.t', calc.im = FALSE))
mix.hist(y = x, model = m3)
m3a = as.fmx(m3, data = x)
```

```
autoplot(m3a)
(l3a = logLik(m3a))
stopifnot(all.equal.numeric(AIC(l3a), m3$aic), all.equal.numeric(BIC(l3a), m3$bic))
autoplot(m3a, type = 'distribution')
```

`as.fmx.t` *Convert* `Normal` *fit from* R*hrefhttps://CRAN.R-project.org/package=mixsmsn***mixsmsn** *to fmx*

Description

..

Usage

```
## S3 method for class 't'
as.fmx(x, data, ...)
```

Arguments

`x`	`'t'` object, returned from smsn.mix with parameter `family = 't'`
`data`	numeric vector
`...`	additional parameters, currently not in use

Value

Function `as.fmx.t()` has not been completed yet

Note

smsn.mix does not offer a parameter to keep the input data, as of 2021-10-06.

Examples

```
library(mixsmsn)
# ?smsn.mix
arg1 = c(mu = 5, sigma2 = 9, lambda = 5, nu = 5)
arg2 = c(mu = 20, sigma2 = 16, lambda = -3, nu = 5)
arg3 = c(mu = 35, sigma2 = 9, lambda = -6, nu = 5)
set.seed(120); x = rmix(n = 1e3L, p=c(.5, .2, .3), family = 'Skew.t',
  arg = list(unname(arg1), unname(arg2), unname(arg3)))

# t
class(m4 <- smsn.mix(x, nu = 3, g = 3, family = 't', calc.im = FALSE))
mix.hist(y = x, model = m4)
# autoplot(as.fmx(m4, data = x)) # not ready yet!!
```

autolayer_fmx_continuous

Create layer for Continuous fmx Objects

Description

..

Usage

```
autolayer_fmx_continuous(
  object,
  type = c("density", "distribution"),
  data = object@data,
  epdf = object@epdf,
  probs = object@probs,
  xlim = if (!length(data)) qfmx(p = c(0.01, 0.99), dist = object) else
    range.default(data),
  hist.fill = "grey95",
  curve.col = 1,
  n = 1001L,
  ...
)
```

Arguments

`object`	fmx object
`type`	character scalar. Option `'density'` (default) plots the probability density for fmx input (and the histogram if argument `data` is available). Option `'distribution'` plots the cumulative probability distribution for fmx input (and the empirical cumulative distribution if argument `data` is available).
`data`	(optional) numeric vector of the observations. Default is the slot `object@data`.
`epdf`	(optional) empirical probability density function returned by approxfun. Default is the slot `object@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.
`xlim`	numeric length-two vector, horizontal range
`hist.fill`	color of the body of histogram, default `'grey95'`
`curve.col`	color of the density curve of the fitted finite mixture distribution. Default `'black'`
`n`	integer, see stat_function
`...`	potential parameters of stat_function

Value

Function `autolayer_fmx_continuous()` returns a list of layers.

See Also

autolayer

`autolayer_fmx_discrete` *Create layer for Discrete fmx Objects*

Description

..

Usage

```
autolayer_fmx_discrete(
  object,
  type = c("density", "distribution"),
  data = object@data,
  xlim = if (length(data)) data else qfmx(p = c(0.01, 0.99), dist = object),
  bins = 60L,
  ...
)
```

Arguments

`object`	fmx object
`type`	character scalar. Option `'density'` (default) plots the probability density for fmx input (and the histogram if argument `data` is available). Option `'distribution'` plots the cumulative probability distribution for fmx input (and the cumulative histogram if argument `data` is available).
`data`	(optional) numeric (actually integer) vector of the observations. Default is the slot `object@data`.
`xlim`	numeric length-two vector, horizontal range
`bins`	integer scalar
`...`	additional parameters, currently not in use

Value

autolayer_fmx_discrete returns a list of layers.

See Also

autolayer

`autoplot.fmx` *Plot fmx Objects using* R*hrefhttps://CRAN.R-project.org/package=ggplot2***ggplot2**

Description

Plot fmx objects using **ggplot2**.

Usage

```
## S3 method for class 'fmx'
autoplot(
  object,
  xlab = attr(object, which = "data.name", exact = TRUE),
  ylab = NULL,
  title = TeX(getTeX(object)),
  caption = NULL,
  ...
)
```

Arguments

`object`	fmx object
`xlab, ylab, title, caption`	character scalars, the horizontal and vertical label, title and caption
`...`	potential parameters of autolayer_fmx_continuous and autolayer_fmx_discrete

Value

autoplot.fmx returns a ggplot object.

See Also

autolayer_fmx_continuous autolayer_fmx_discrete 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')
```

autoplot.fmxS *Plot fmxS Object*

Description

..

Usage

```
## S3 method for class 'fmxS'
autoplot(
  object,
  type = c("density", "distribution"),
 xlim = if (length(data)) range.default(data) else range.default(lapply(dots, FUN =
    qfmx, p = c(0.01, 0.99))),
  ...
)
```

Arguments

`object`	fmxS object
`type`	character scalar, `'density'` (default) or `'distribution'`
`xlim`	..
`...`	..

Value

Function `autoplot.fmxS()` returns a ggplot figure.

CK5 *CK5*

Description

..

Usage

```
CK5
```

Format

a list of double vectors

clusterList *Clustering Observations for Creation of fmx Object*

Description

..

Usage

```
clusterList(x, K, method = c("reassign_tkmeans"), alpha = 0.05, ...)
```

Arguments

`x`	numeric vector, one-dimensional observations
`K`	integer scalar, number of mixture components
`method`	character scalar, only `'reassign_tkmeans'` supported yet
`alpha`	numeric scalar, proportion of observations to be trimmed in trimmed k-means algorithm tkmeans
`...`	additional parameters, currently not in use

Value

Function `clusterList()` returns a list of numeric vectors.

See Also

`reAssign.tkmeans()` kmeans

coef.fmx *Parameter Estimates of fmx object*

Description

..

Usage

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

Arguments

`object`	fmx 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

Details

Function `coef.fmx()` returns the estimates of the user-friendly parameters (`parm = 'user'`), or the internal/unconstrained parameters (`parm = 'internal'`). When the distribution has constraints on one or more parameters, `coef.fmx()` does not return the estimates (which is constant `0`) of the constrained parameters.

Value

Function `coef.fmx()` returns a numeric vector.

`confint.fmx` *Confidence Interval of fmx Object*

Description

...

Usage

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

Arguments

`object`	fmx object
`...`	place holder for S3 naming convention
`level`	confidence level, default 95%.

Details

confint.fmx 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, confint.fmx does not return the confident intervals of for the constrained parameters.

Value

confint.fmx returns a matrix

`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 X

Value

Function `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 qr.default qr.R chol2inv

Examples

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

`CycD1` *CycD1*

Description

..

Usage

```
CycD1
```

Format

a list of double vectors

dbl2fmx *Inverse of fmx2dbl(), for internal use*

Description

..

Usage

```
dbl2fmx(x, K, distname, ...)
```

Arguments

`x`	numeric vector, unrestricted parameters
`K`	integer scalar
`distname`	character scalar
`...`	additional parameters, not currently used

Details

Only used in function `QLMDe()` and unexported function `qfmx_gr()`, not compute intensive

Value

Function `dbl2fmx()` returns a list with two elements `$pars` and `$w`

dfmx *Density, Distribution and Quantile of 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

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

pfmx(
  q,
  dist,
  distname = dist@distname,
  K = dim(pars)[1L],
  pars = dist@pars,
  w = dist@w,
  ...,
  lower.tail = TRUE,
  log.p = FALSE
)

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

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

Arguments

`x, q`	numeric vector, quantiles, `NA_real_` value(s) allowed.
`dist`	fmx object, a finite mixture distribution
`distname, K, pars, w`	auxiliary parameters, whose default values are determined by argument `dist`. The user-specified vector of `w` does not need to sum up to 1; `w/sum(w)` will be used internally.
`...`	additional parameters
`log, log.p`	logical scalar. If `TRUE`, probabilities are given as $\log(p)$.
`lower.tail`	logical scalar. If `TRUE` (default), probabilities are $Pr(X \leq x)$, otherwise, $Pr(X > x)$.

`p`	numeric vector, probabilities.
`interval`	length two numeric vector, interval for root finding, see vuniroot
`n`	integer scalar, number of observations.

Details

A computational challenge in function `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 function `logLik.fmx()`. 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.

Function `qfmx()` gives the quantile function, by numerically solving pfmx. 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

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

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

Function `qfmx()` returns an unnamed numeric 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.

Function `rfmx()` generates random deviates of an fmx object.

Examples

```
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))
}
```

distArgs — *Name(s) of Formal Argument(s) of Distribution*

Description

To obtain the name(s) of distribution parameter(s).

Usage

```
distArgs(distname)
```

Arguments

distname	character scalar, name of distribution

Value

Function `distArgs()` returns a character vector.

See Also

formalArgs

distType — *Distribution Type*

Description

..

Usage

```
distType(type = c("discrete", "nonNegContinuous", "continuous"))
```

Arguments

`type`	character scalar

Value

distType returns a character vector.

`dist_logtrans` *Distribution Parameters that needs to have a log-transformation*

Description

..

Usage

```
dist_logtrans(distname)
```

Arguments

`distname`	character scalar, name of distribution

Value

dist_logtrans returns an integer scalar

`drop1_fmx` *Drop or Add One Parameter from fmx Object*

Description

Fit fmx models with a single parameters being added or dropped.

Usage

```
## S3 method for class 'fmx'
drop1(object, ...)

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

Arguments

`object`	fmx object
`...`	additional parameters, currently not in use.

Details

..

Value

drop1.fmx and add1.fmx return a list of fmx objects, in the reverse order of model selection.

Note

Note that drop1.fmx and add1.fmx do *not* return an anova table, like other `stats:::drop.*` or `stats:::add1.*` functions do.

See Also

step

Examples

```
# donttest to save time

(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))
system.time(m0 <- QLMDe(x2, distname = 'GH', K = 2L, constraint = c('g1', 'g2', 'h1', 'h2')))
system.time(m1 <- QLMDe(x2, distname = 'GH', K = 2L, constraint = c('g1', 'h2')))
system.time(m2 <- QLMDe(x2, distname = 'GH', K = 2L)) # ~2 secs

d1 = drop1(m1)
d1 # NULL
d2 = drop1(m2)
vapply(d2, FUN = getTeX, FUN.VALUE = '')

a0 = add1(m0)
vapply(a0, FUN = getTeX, FUN.VALUE = '')
a1 = add1(m1)
vapply(a1, FUN = getTeX, FUN.VALUE = '')
```

`fmx` *Create Finite Mixture Distribution*

Description

..

Usage

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

Arguments

`distname`	character scalar
`w`	(optional) numeric vector. Does not need to sum up to 1; `w/sum(w)` will be used internally.
`...`	mixture distribution parameters. See function `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.

Value

Function `fmx()` returns an fmx object which specifies the parameters of a finite mixture distribution.

Examples

```
(e1 = fmx('norm', mean = c(0,3), sd = c(1,1.3), w = c(1, 1)))
isS4(e1) # TRUE
slotNames(e1)

(e2 = fmx('GH', A = c(0,3), g = c(.2, .3), h = c(.2, .1), w = c(2, 3)))

(e3 = fmx('GH', A = 0, g = .2, h = .2)) # one-component Tukey
```

`fmx-class` *Specification of fmx Class*

Description

Parameters and type of distribution of a one-dimensional finite mixture.

Slots

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

`pars` 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 columns match the formal arguments of the corresponding distribution, e.g., `'mean'` and `'sd'` for normal mixture, or `'A'`, `'B'`, `'g'` and `'h'` for Tukey's g-&-h mixture.

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

`data` (optional) numeric vector, the one-dimensional observations

`data.name`	(optional) character scalar, a human-friendly name of observations
`epdf`	(optional) empirical probability density function returned by approxfun
`vcov_internal`	(optional) variance-covariance matrix of the internal (i.e., unconstrained) estimates
`vcov`	(optional) variance-covariance matrix of the mixture distribution (i.e., constrained) estimates
`probs`	(optional) numeric vectors of probabilities, where the quantiles could be calculated
`Kolmogorov,CramerVonMises,KullbackLeibler`	(optional) numeric scalars

fmx2dbl *Reparameterization of fmx Object*

Description

To convert the parameters of fmx object into unrestricted parameters.

Usage

```
fmx2dbl(
  x,
  distname = x@distname,
  pars = x@pars,
  K = dim(pars)[1L],
  w = x@w,
  ...
)
```

Arguments

`x`	fmx object
`distname`	character scalar, default `x@distname`
`pars`	numeric matrix, default `x@pars`
`K`	integer scalar, default value from `x`
`w`	numeric vector, default `x@w`
`...`	additional parameters, not currently used

Details

For the first parameter

- $A_1 \rightarrow A_1$
- $A_2 \rightarrow A_1 + \exp(\log(d_1))$
- $A_k \rightarrow A_1 + \exp(\log(d_1)) + \cdots + \exp(\log(d_{k-1}))$

For mixing proportions to multinomial logits.

For `'norm'`: `sd` -> `log(sd)` for `'GH'`: `B -> log(B), h -> log(h)`

Value

Function `fmx2dbl()` returns a numeric vector

See Also

`dbl2fmx()`

`fmxS` *fmxS: Multiple fmx objects*

Description

..

Usage

```
fmxS(...)
```

Arguments

`...`	multiple fmx objects, or objects convertible to fmx class via function `as.fmx()`

Value

Function `fmxS()` returns an fmxS object.

Slots

`.Data` list of fmx objects

`data` numeric vector

`data.name` character scalar

Examples

```
library(fitdistrplus)
set.seed(1234); x = rnorm(n = 1e3L)
f1 = fitdist(x, distr = 'norm')
f2 = fitdist(x, distr = 'GH', start = as.list.default(letterValue(x)))
aa = fmxS(a = f1, b = f2)
summary(aa)
autoplot(aa, type = 'density')
autoplot(aa, type = 'distribution')

a1 = fmx('GH', A = c(7,9), B = c(.8, 1.2), g = c(.3, 0), h = c(0, .1), w = c(1, 1))
a2 = fmx('GH', A = c(6,9), B = c(.8, 1.2), g = c(-.3, 0), h = c(.2, .1), w = c(4, 6))
a = fmxS(a1, a2)
(p = autoplot(a, type = 'distribution') + coord_flip())
```

```
p + labs(x = 'new xlab', y = 'new ylab')
p + theme(legend.position = 'none')
```

`fmx_cluster` *Naive Estimates of Finite Mixture Distribution via Clustering*

Description

Naive estimates for finite mixture distribution fmx via clustering.

Usage

```
fmx_cluster(
  x,
  K,
  distname = c("GH", "norm", "sn"),
  constraint = character(),
  ...
)
```

Arguments

`x`	numeric vector, observations
`K`	integer scalar, number of mixture components
`distname`	character scalar, name of parametric distribution of the mixture components
`constraint`	character vector, parameters (g and/or h for Tukey's g-&-h mixture) to be set at 0. See function `fmx_constraint()` for details.
`...`	additional parameters, currently not in use

Details

First of all, if the specified number of components $K \geq 2$, trimmed k-means clustering with re-assignment 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 function `fmx_cluster()`.
- 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 $K = 1$.

Value

Function `fmx_cluster()` returns an fmx object.

See Also

`letterValue()`

`fmx_constraint` *Parameter Constraint(s) of Mixture Distribution*

Description

Determine the parameter constraint(s) of a finite mixture distribution fmx, 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@pars)[1L],
  pars = dist@pars
)
```

Arguments

`dist`	(optional) fmx object
`distname`	character scalar, name of distribution (see fmx), default value determined by `dist`
`K`	integer scalar, number of components, default value determined by `dist`
`pars`	double matrix, distribution parameters of a finite mixture distribution (see fmx), default value determined by `dist`

Value

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

Examples

```
(d0 = fmx('GH', A = c(1,4), g = c(.2,.1), h = c(.05,.1), w = c(1,1)))
(c0 = fmx_constraint(d0))
user_constraint(character(), distname = 'GH', K = 2L) # equivalent

(d1 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(0,.1), w = c(1,1)))
(c1 = fmx_constraint(d1))
user_constraint(c('g2', 'h1'), distname = 'GH', K = 2L) # equivalent

(d2 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(.15,.1), w = c(1,1)))
(c2 = fmx_constraint(d2))
user_constraint('g2', distname = 'GH', K = 2L) # equivalent
```

`fmx_diagnosis` *Diagnoses for fmx Estimates*

Description

Diagnoses for fmx estimates.

Usage

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

KullbackLeibler_fmx(object, data = object@data, ...)

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

Arguments

`object`	fmx object, or an R object convertible to an fmx object
`data`	double vector, observed data. Default is `object@data`, the data used for estimation.
`...`	additional parameters, currently not in use

Details

Function `Kolmogorov_fmx()` calculates Kolmogorov distance.

Function `KullbackLeibler_fmx()` calculates Kullback-Leibler divergence. The R code is adapted from `LaplacesDemon::KLD`.

Function `CramerVonMises_fmx()` calculates Cramer-von Mises quadratic distance (via cvm.test).

Value

Functions `Kolmogorov_fmx()`, `KullbackLeibler_fmx()`, `CramerVonMises_fmx()` all return numeric scalars.

See Also

`dgof::cvmf.test`

fmx_hybrid — *Best Naive Estimates for Finite Mixture Distribution*

Description

Best estimates for finite mixture distribution fmx.

Usage

```
fmx_hybrid(x, test = c("logLik", "CvM", "KS"), ...)
```

Arguments

`x`	numeric vector, observations
`test`	character scalar, criteria for selecting the optimal estimates. See **Details**.
`...`	additional parameters of fmx_normix and fmx_cluster

Details

fmx_hybrid compares the Tukey's g-&-h mixture estimate provided by fmx_cluster and the normal mixture estimate by fmx_normix, and select the one either with maximum likelihood (`test = 'logLik'`, default), with minimum Cramer-von Mises distance (`test = 'CvM'`) or with minimum Kolmogorov distance (`Kolmogorov_fmx()`).

Value

fmx_hybrid returns an fmx object.

Examples

```
d1 = fmx('norm', mean = c(1, 2), sd = .5, w = c(.4, .6))
set.seed(100); hist(x1 <- rfmx(n = 1e3L, dist = d1))
fmx_normix(x1, distname = 'norm', K = 2L)
fmx_normix(x1, distname = 'GH', 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))
fmx_cluster(x2, K = 2L)
fmx_cluster(x2, K = 2L, constraint = c('g1', 'h2'))
fmx_normix(x2, K = 2L, distname = 'GH')
fmx_hybrid(x2, distname = 'GH', K = 2L)
```

fmx_normix — *Naive Estimates of Finite Mixture Distribution using Mixture of Normal*

Description

Naive estimates for finite mixture distribution fmx using mixture of normal

Usage

```
fmx_normix(x, K, distname = c("GH", "norm", "sn"), alpha = 0.05, R = 10L, ...)
```

Arguments

`x`	numeric vector, observations
`K`	integer scalar, number of mixture components
`distname`	character scalar, name of parametric distribution of the mixture components
`alpha`	numeric scalar, proportion of observations to be trimmed in trimmed k-means algorithm tkmeans
`R`	integer scalar, number of normalmixEM replicates
`...`	additional parameters, currently not in use

Details

fmx_normix ... 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 be selected

Value

fmx_normix returns an fmx object.

geom_function_args — *geom_function with Multiple Sets of Arguments*

Description

..

Usage

```
geom_function_args(args, ...)
```

Arguments

`args`	*named* list of arguments
`...`	parameters of geom_function, most importantly the parameter `fun`

Details

Function `geom_function_args()` plots *one* function using a *list of* arguments, by calling geom_function repetitively. The colour labels are the names of argument list `args`.

Value

Function `geom_function_args()` returns a list of ggplot layers.

Note

Parameter `args` of geom_function is *not* vectorized.

See geom_function, for the difference from stat_function.

Examples

```
ggplot() +
 geom_function_args(
   args = c('$\\alpha$' = 1, '$\\beta$' = 2),
   fun = function(x, a) a*x^2,
   xlim = c(-3, 3)) +
 labs(colour = 'Args')
```

`getTeX` *TeX Label (of Parameter Constraint(s)) of fmx Object*

Description

Create TeX label of (parameter constraint(s)) of fmx object

Usage

```
getTeX(dist, print_K = FALSE)
```

Arguments

`dist`	fmx object
`print_K`	logical scalar, whether to print the number of components K. Default `FALSE`.

Value

Function `getTeX()` returns a character scalar (of TeX 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)))
getTeX(d0)

(d1 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(0,.1), w = c(1,1)))
getTeX(d1)

(d2 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(.15,.1), w = c(1,1)))
getTeX(d2)
```

`Kolmogorov_dist` *One-Sample Kolmogorov Distance*

Description

To calculate the one-sample Kolmogorov distance between observations and a distribution.

Usage

```
Kolmogorov_dist(x, null, alternative = c("two.sided", "less", "greater"), ...)
```

Arguments

`x`	numeric vector, observations x
`null`	cumulative distribution function
`alternative`	character scalar, alternative hypothesis, either `'two.sided'` (default), `'less'`, or `'greater'`
`...`	additional arguments of `null`

Details

Function `Kolmogorov_dist()` is different from ks.test in the following aspects

- Ties in observations are supported. The step function of empirical distribution is inspired by ecdf. This is superior than `(0:(n - 1))/n` in ks.test.
- Discrete distribution (with discrete observation) is supported.
- Discrete distribution (with continuous observation) is not supported yet. This will be an easy modification in future.
- Only the one-sample Kolmogorov distance, not the one-sample Kolmogorov test, is returned, to speed up the calculation.

Value

Function `Kolmogorov_dist()` returns a numeric scalar.

See Also

ks.test

Examples

```
# from ?stats::ks.test
x1 = rnorm(50)
ks.test(x1+2, y = pgamma, shape = 3, rate = 2)
Kolmogorov_dist(x1+2, null = pgamma, shape = 3, rate = 2) # exactly the same

# discrete distribution
x2 <- rnbinom(n = 1e2L, size = 500, prob = .4)
suppressWarnings(ks.test(x2, y = pnbinom, size = 500, prob = .4)) # warning on ties
Kolmogorov_dist(x2, null = pnbinom, size = 500, prob = .4) # wont be the same
```

---

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

Description

Letter-value based estimation (Hoaglin, 2006) of Tukey's g-, h- and g-&-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, probabilities used for estimating g parameter. Or, use `p_g = FALSE` to implement the constraint $g = 0$ (i.e., an h-distribution is estimated).

`p_h`	double vector, probabilities used for estimating h parameter. Or, use `p_h = FALSE` to implement the constraint $h = 0$ (i.e., a g-distribution is estimated).
`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 location $\hat{A}$ and skewness $\hat{g}$

Details

letterV_g estimates parameter g using equation (10) for g-distribution and the equivalent equation (31) for g-&-h distribution.

letterV_B estimates parameter B for Tukey's g-distribution (i.e., $g \neq 0$, $h = 0$), 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. doi:10.1002/9781118150702.ch11

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))
plot(fit) # fitdistrplus:::plot.fitdist
```

`logLik.fitdist` *Log-Likelihood of fitdist Object*

Description

..

Usage

```
## S3 method for class 'fitdist'
logLik(object, ...)
```

Arguments

`object`	fitdist object
`...`	additional parameters, currently not in use

Details

Output of fitdist has elements `$loglik`, `$aic` and `$bic`, but they are simply numeric scalars. `fitdistrplus:::logLik.fitdist` simply returns these elements.

Function `logLik.fitdist()` returns a logLik object, which could be further used by AIC and BIC.

(I have written to the authors)

Value

Function `logLik.fitdist()` returns a logLik object

`logLik.fmx` *Log-Likelihood of fmx Object*

Description

..

Usage

```
## S3 method for class 'fmx'
logLik(object, data = object@data, ...)
```

Arguments

`object`	fmx object
`data`	double vector, actual observations
`...`	place holder for S3 naming convention

Details

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.

Value

logLik.fmx returns a logLik object with an additional attribute `attr(,'logl')`.

`logLik.mixEM` *Log-Likelihood of* `'mixEM'` *Object*

Description

To obtain the log-Likelihood of `'mixEM'` object, based on **mixtools** 2020-02-05.

Usage

```
## S3 method for class 'mixEM'
logLik(object, ...)
```

Arguments

`object`	`'mixEM'` object, currently only the returned value of normalmixEM and gammamixEM are supported
`...`	additional parameters, currently not in use

Value

Function `logLik.mixEM()` returns a logLik object.

`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 Σ

Value

mahalanobis_int returns a numeric scalar.

`mixEM_pars` *Names of Distribution Parameters of `'mixEM'` Object*

Description

Names of distribution parameters of `'mixEM'` object, based on **mixtools** 2020-02-05.

Usage

```
mixEM_pars(object)
```

Arguments

`object`	`'mixEM'` object, currently only the returned value of normalmixEM and gammamixEM are supported

Value

Function `mixEM_pars()` returns a character vector

See Also

normalmixEM gammamixEM

`mlogis` *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, multinomial probabilities, adding up to 1
q	numeric vector, multinomial logits

Details

Functions `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.

Functions `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

Functions `pmlogis_first()` and `pmlogis_last()` return a vector of multinomial probabilities p.

Functions `qmlogis_first()` and `qmlogis_last()` return 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)))
```

`moment_`	*Raw, Central and Standardized Moments, and other Distribution Characteristics*

Description

Up to 4th order of raw $E(Y^n)$, central $E[(Y-\mu)^n]$ and standardized moments $E[(Y-\mu)^n]/\sigma^n$ of the random variable $Y = (X - \texttt{location})/\texttt{scale}$, as well as the distribution characteristics (e.g., mean, standard deviation, skewness and excess kurtosis) of the random variable X.

Usage

```
moment_GH(A, B, g, h)

moment_sn(xi, omega, alpha)

moment_st(xi, omega, alpha, nu)

moment_norm(mean, sd)

moment_(dist, ...)

## S3 method for class 'character'
moment_(dist = c("norm", "GH", "sn", "st"), ...)

## S3 method for class 'fmx'
moment_(dist, ...)
```

Arguments

`A, B, g, h`	numeric vectors or scalars, parameters of Tukey's gh distribution `dGH()`
`xi, omega, alpha`	numeric vectors or scalars, location, scale and slant parameters for skew-normal dsn and skew-t dst distributions
`nu`	positive numeric vector or scalar, degrees of freedom(s) of skew-t dst distribution
`mean, sd`	numeric vectors or scalars, mean and standard deviation parameters for normal dnorm distribution
`dist`	see **Usage**
`...`	distribution parameters as described in **Arguments** for `moment_.character()`, or place holder for S3 method dispatch for `moment_.fmx()`

Details

For $Y = (X - \texttt{location})/\texttt{scale}$, let $\mu = E(Y)$, then the second to fourth central moments of Y are,

$$E[(Y-\mu)^2] = E(Y^2) - 2\mu E(Y) + \mu^2 = E(Y^2) - \mu^2$$

$$E[(Y-\mu)^3] = E(Y^3) - 3\mu E(Y^2) + 3\mu^2 E(Y) - \mu^3 = E(Y^3) - 3\mu E(Y^2) + 2\mu^3$$

$$E[(Y-\mu)^4] = E(Y^4) - 4\mu E(Y^3) + 6\mu^2 E(Y^2) - 4\mu^3 E(Y) + \mu^4 = E(Y^4) - 4\mu E(Y^3) + 6\mu^2 E(Y^2) - 3\mu^4$$

The distribution characteristics of Y are,

$$\mu_Y = \mu$$

$$\sigma_Y = \sqrt{E[(Y-\mu)^2]}$$

$$\texttt{skewness}_Y = E[(Y-\mu)^3]/\sigma_Y^3$$

$$\texttt{kurtosis}_Y = E[(Y-\mu)^4]/\sigma_Y^4 - 3$$

The distribution characteristics of X are $\mu_X = \texttt{location} + \texttt{scale} \cdot \mu_Y$, $\sigma_X = \texttt{scale} \cdot \sigma_Y$, $\texttt{skewness}_X = \texttt{skewness}_Y$, and $\texttt{kurtosis}_X = \texttt{kurtosis}_Y$.

The S3 method dispatch `moment_.character()` obtains the moments and distribution characteristics from the distribution name `dist` and parameters given in `...`.

The S3 method dispatch `moment_.fmx()` obtains the moments and distribution characteristics of each mixture component of an fmx object.

Value

Functions `moment_()`, `moment_GH()`, `moment_sn()`, `moment_st()`, `moment_norm()` all return a moment object.

Slots

`distname` character scalar, name of distribution, e.g., `'norm'` for normal, `'sn'` for skew-normal, `'st'` for skew-t, and `GH` for Tukey's g-and-h distribution, following the nomenclature of dnorm, dsn, dst and `dGH()`

`location,scale` numeric vectors or scalars, location and scale parameters

`mu` numeric vector or scalar, 1st order *raw* moment $\mu = E(Y)$. Note that the 1st order central moment $E(Y-\mu)$ and standardized moment $E(Y-\mu)/\sigma$ are 0.

`raw2,raw3,raw4` numeric vectors or scalars, 2nd or higher order *raw* moments $E(Y^n)$, $n \geq 2$

`central2,central3,central4` numeric vectors or scalars, 2nd or higher order *central* moments, $\sigma^2 = E[(Y-\mu)^2]$ and $E[(Y-\mu)^n]$, $n \geq 3$

`standardized3,standardized4` numeric vectors or scalars, 3rd or higher order *standardized* moments, skewness $E[(Y-\mu)^3]/\sigma^3$ and kurtosis $E[(Y-\mu)^4]/\sigma^4$. Note that the 2nd standardized moment is 1

`mean,sd,skewness,kurtosis` numeric vectors or scalars, distribution characteristics of random variable X, such as mean, standard deviation, skewness, and excess kurtosis

Note

Potential name clash with `e1071::moment`.

References

https://en.wikipedia.org/wiki/Binomial_theorem https://en.wikipedia.org/wiki/Central_moment https://en.wikipedia.org/wiki/Standardized_moment https://en.wikipedia.org/wiki/Skewness https://en.wikipedia.org/wiki/Kurtosis

Raw moments of Tukey's GH: doi:10.1002/9781118150702.ch11

Raw moments of skew-normal: https://en.wikipedia.org/wiki/Skew_normal_distribution

Raw moments of skew-t: https://arxiv.org/abs/0911.2342

Raw moments of normal: https://en.wikipedia.org/wiki/Normal_distribution (replace with $\mu = 0$ and $\sigma = 1$)

Examples

```
library(ggplot2)

moment_(dist = 'norm', mean = 1.234, sd = .58)

## Not run:  # requires Tingting's \pkg{QuantileGH}
A = 3; B = 1.5; g = .7; h = .1
moment_(dist = 'GH', A = A, B = B, g = 0, h = h)
moment_(dist = 'GH', A = A, B = B, g = g, h = 0)
moment_(dist = 'GH', A = A, B = B, g = g, h = h)
## End(Not run)

xi = 2; omega = 1.3; alpha = 3; nu = 6

ggplot() + geom_function(fun = sn::dsn, args = list(
  xi = xi, omega = omega, alpha = alpha
), xlim = c(0, 6))
moment_(dist = 'sn', xi, omega, alpha)

ggplot() + geom_function(fun = sn::dst, args = list(
  xi = xi, omega = omega, alpha = alpha, nu = nu
), xlim = c(0, 6))
moment_(dist = 'st', xi, omega, alpha, nu)
```

`nobs.fitdist` *Number of Observations in fitdist Object*

Description

..

Usage

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

Arguments

`object`	fitdist object
`...`	additional parameters, currently not in use

Value

Function `nobs.fitdist()` returns an integer scalar

`nobs.fmx` *Number of Observations in fmx Object*

Description

..

Usage

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

Arguments

`object`	fmx object
`...`	place holder for S3 naming convention

Details

nobs.fmx returns the sample size of the observations used in QLMDe estimation, or `integer(0)` for distribution-only fmx object

Value

nobs.fmx returns an integer scalar.

See Also

nobs

`npar.fmx` *Number of Parameters of fmx Object*

Description

..

Usage

```
npar.fmx(dist)
```

Arguments

`dist`	fmx object

Details

Also the degree-of-freedom in logLik, as well as `stats:::AIC.logLik` and `stats:::BIC.logLik`

Value

npar.fmx returns an integer scalar.

`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_t double vector, the target value(s), missing value not allowed
`current`	length-n_c double vector, the value(s) to be compared with `target`, missing value not allowed
`tolerance`	positive double scalar, default `sqrt(.Machine$double.eps)`
`...`	potential parameters, currently not in use

Details

Function `outer_allequal()` is different from all.equal.numeric, such that

- only compares between double, not complex, values
- element-wise comparison is performed
- a logical scalar is always returned for each element-wise comparison.

Value

Function `outer_allequal()` returns an $n_c \times n_t$ logical matrix indicating whether the length-n_c vector `current` is element-wise near-equal to the length-n_t vector `target` within the pre-specified `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))
```

`print.fmx` *S3 print of fmx Object*

Description

..

Usage

```
## S3 method for class 'fmx'
print(x, ...)
```

Arguments

`x`	an fmx object
`...`	additional parameters, not currently in use

Value

print.fmx returns the input fmx object invisibly.

See Also

print

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("GH", "norm", "sn"),
  K,
  data.name = deparse1(substitute(x)),
  constraint = character(),
  probs = 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 scalar, 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 scalar, 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 function `fmx_constraint()` for details.
`probs`	numeric vector, percentiles at where the sample and theoretical quantiles are to be matched. See function `QLMDp()` for details.
`init`	character scalar for the method of initial values selection, or an fmx object of the initial values. See function `fmx_hybrid()` for more details.
`tol, maxiter`	see function `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

Function `QLMDe()` returns an fmx object.

See Also

`fmx_hybrid()`

Examples

```
hist(x1 <- CK5[[1L]])
QLMDe(x1, distname = 'GH', K = 2L)
```

QLMDe_stepK — *Forward Selection of the Number of Components K*

Description

To compare gh-parsimonious models of Tukey's g-&-h mixtures with different number of components K (up to a user-specified K_{max}) and select the optimal number of components.

Usage

```
QLMDe_stepK(
  x,
  distname = c("GH", "norm"),
  data.name = deparse1(substitute(x)),
  Kmax = 3L,
  test = c("BIC", "AIC"),
  direction = c("forward", "backward"),
  ...
)
```

Arguments

`x`	numeric vector, observations
`distname, data.name`	character scalars, see parameters of the same names in function `QLMDe()`
`Kmax`	integer scalar K_{max}, maximum number of components to be considered. Default `3L`

`test`	character scalar, criterion to be used, either Akaike's information criterion AIC, or Bayesian information criterion BIC (default).
`direction`	character scalar, direct of selection in function `step_fmx()`, either `'forward'` (default) or `'backward'`
`...`	additional parameters

Details

Function `QLMDe_stepK()` compares the gh-parsimonious models with different number of components K, and selects the optimal number of components using BIC (default) or AIC.

The forward selection starts with finding the gh-parsimonious model (via function `step_fmx()`) at $K = 1$. Let the current number of component be K^c. We compare the gh-parsimonious models of $K^c + 1$ and K^c component, respectively, using BIC or AIC. If K^c is preferred, then the forward selection is stopped, and K^c is considered the optimal number of components. If K^c+1 is preferred, then the forward selection is stopped if $K^c + 1 = K_{max}$, otherwise update K^c with $K_c + 1$ and repeat the previous steps.

Value

Function `QLMDe_stepK()` returns an object of S3 class `'stepK'`, which is a list of selected models (in reversed order) with attribute(s) `'direction'` and `'test'`.

Examples

```
hist(x1 <- CK5[[1L]])
QLMDe_stepK(x1, distname = 'GH', Kmax = 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, to`	numeric scalar, minimum and maximum of the equidistant (in probability or quantile) probabilities. Default `.05` and `.95`, respectively
`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 function `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 = after_stat(density)), bins = 30L, colour = 'white', alpha = .1)

(p1 = QLMDp()) # equidistant in probabilities
autoplot(d2, probs = p1) + p_hist

(p2 = QLMDp(equidistant = 'quantile', x = x2)) # equidistnat in quantiles
autoplot(d2, probs = p2) + 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(probs, d)
```

Arguments

`probs`	numeric vector, cumulative probabilities at the given quantiles
`d`	numeric vector, probability densities at the given quantiles

Details

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

Function `quantile_vcov()` must not be used in a compute-intensive way.

Value

Function `quantile_vcov()` returns the variance-covariance matrix of quantiles based on Mosteller (1946).

References

Frederick Mosteller. On Some Useful "Inefficient" Statistics (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

Function `reAssign.tkmeans()` returns an `'reAssign_tkmeans'` object, which inherits from tkmeans class.

Note

Either kmeans or tkmeans is slow for big `x`.

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')
```

`show,fmx-method` *Show fmx 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 object

Value

The show method for fmx object does not have a returned value.

`show,moment-method` *Show moment*

Description

..

Usage

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

Arguments

`object`	moment

Value

The show method for moment object does not have a returned value.

`sort.mixEM` *Sort* `'mixEM'` *Object by First Parameters*

Description

To sort a `'mixEM'` object by its first parameters, i.e., μ's for normal mixture, α's for γ-mixture, etc.

Usage

```
## S3 method for class 'mixEM'
sort(x, decreasing = FALSE, ...)
```

Arguments

`x`	`'mixEM'` object
`decreasing`	logical scalar. Should the sort by mu's be increasing (`FALSE`, default) or decreasing (`TRUE`)?
`...`	additional parameters, currently not in use

Details

normalmixEM does *not* order the location parameter

Value

sort.mixEM returns a `'mixEM'` object.

See Also

sort

`sort_mixsmsn` *Sort Objects from* R*hrefhttps://CRAN.R-project.org/package=mixsmsn***mixsmsn** *by Location Parameters*

Description

To sort an object returned from package **mixsmsn** by its location parameters

Usage

```
## S3 method for class 'Skew.normal'
sort(x, decreasing = FALSE, ...)

## S3 method for class 'Normal'
sort(x, decreasing = FALSE, ...)

## S3 method for class 'Skew.t'
sort(x, decreasing = FALSE, ...)

## S3 method for class 't'
sort(x, decreasing = FALSE, ...)
```

Arguments

`x`	`'Normal'`, `'Skew.normal'`, `'Skew.t'` object
`decreasing`	logical scalar. Should the sort the location parameter be increasing (`FALSE`, default) or decreasing (`TRUE`)?
`...`	additional parameters, currently not in use

Details

smsn.mix does *not* order the location parameter

Value

Function `sort.Normal()` returns a `'Normal'` object.

Function `sort.Skew.normal()` returns a `'Skew.normal'` object.

Function `sort.Skew.t()` returns a `'Skew.t'` object.

See Also

sort

step_fmx — *Forward Selection of gh-parsimonious Model with Fixed Number of Components K*

Description

To select the gh-parsimonious mixture model, i.e., with some g and/or h parameters equal to zero, conditionally on a fixed number of components K.

Usage

```
step_fmx(
  object,
  test = c("BIC", "AIC"),
  direction = c("forward", "backward"),
  ...
)
```

Arguments

`object`	fmx object
`test`	character scalar, criterion to be used, either Akaike's information criterion AIC, or Bayesian information criterion BIC (default).
`direction`	character scalar, `'forward'` (default) or `'backward'`
`...`	additional parameters, currently not in use

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 object of S3 class `'step_fmx'`, which is a list of selected models (in reversed order) with attribute(s) `'direction'` and `'test'`.

See Also

step

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)
`B`	double scalar, scale parameter $B > 0$, default $B = 1$ (as parameter `sd` of dnorm)
`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 function `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

Function `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 `gk::dgh`.

Function `pGH()` gives the distribution function, only taking scalar arguments and vector quantiles `q`. This function takes about 1/10 time of `gk::pgh` and `OpVaR::pgh` functions.

Function `qGH()` gives the quantile function, only taking scalar arguments and vector probabilities `p`. This function takes about 1/2 time of `gk::qgh` and 1/10 time of `OpVaR::qgh` functions.

Function `rGH()` generates random deviates, only taking scalar arguments.

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

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

See Also

`OpVaR::dgh` `gk::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 function `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

`outer_allequal()` duplicated.default unique.default

Examples

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

`user_constraint` *Formalize User-Specified Constraint of fmx Object*

Description

Formalize user-specified constraint of fmx object

Usage

```
user_constraint(x, distname, K)
```

Arguments

`x`	character vector, constraint(s) to be imposed. For example, for a two-component Tukey's g-&-h mixture, `c('g1', 'h2')` indicates $g_1 = h_2 = 0$ given $A_1 < A_2$, i.e., the g-parameter for the first component (with smaller location value) and the h-parameter for the second component (with larger mean value) are to be constrained as 0.
`distname`	character scalar, name of distribution
`K`	integer scalar, number of components

Value

user_constraint returns the indices 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')`).

Examples

```
(d0 = fmx('GH', A = c(1,4), g = c(.2,.1), h = c(.05,.1), w = c(1,1)))
(c0 = fmx_constraint(d0))
user_constraint(distname = 'GH', K = 2L, x = character()) # equivalent

(d1 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(0,.1), w = c(1,1)))
(c1 = fmx_constraint(d1))
user_constraint(distname = 'GH', K = 2L, x = c('g2', 'h1')) # equivalent

(d2 = fmx('GH', A = c(1,4), g = c(.2,0), h = c(.15,.1), w = c(1,1)))
(c2 = fmx_constraint(d2))
user_constraint(distname = 'GH', K = 2L, x = 'g2') # equivalent
```

`vcov.fmx` *Variance-Covariance of fmx Object*

Description

..

Usage

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

Arguments

`object`	fmx 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

Details

vcov.fmx 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'`). When the distribution has constraints on one or more parameters, vcov.fmx does not return the variance/covariance involving the constrained parameters.

Value

vcov.fmx returns a matrix.

`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

Function `vuniroot2()`, different from vuniroot, does

- accept `NA_real_` as element(s) of y
- handle the case when the analytic 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

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

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 Object

Description

Taking subset of components in fmx object

Usage

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

Arguments

x	fmx object
i	integer or logical vector, the row indices of *components* to be chosen, see [

Details

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

Value

An fmx object consisting of a subset of components. information about the observations (e.g. slots `@data` and `@data.name`), will be lost.

Examples

```
(d = fmx('norm', mean = c(1, 4, 7), w = c(1, 1, 1)))
d[1:2]
```

Index

∗ **datasets**
    CK5, 13
    CycD1, 16
[, 59
[,fmx,ANY,ANY,ANY-method, 59

add1.fmx, 22
add1.fmx (drop1_fmx), 21
AIC, 35, 47, 53
algorithm, 30
all.equal.numeric, 44
anova, 22
approx, 3
approxdens, 3, 3
approxfun, 3, 10, 24
as.fmx, 4
as.fmx(), 4, 25
as.fmx.fitdist, 5
as.fmx.fitdist(), 4, 5
as.fmx.mixEM, 5
as.fmx.mixEM(), 4, 5
as.fmx.Normal, 6
as.fmx.Normal(), 6
as.fmx.Skew.normal, 7
as.fmx.Skew.normal(), 7
as.fmx.Skew.t, 8
as.fmx.Skew.t(), 8
as.fmx.t, 9
as.fmx.t(), 9
autolayer, 11
autolayer_fmx_continuous, 10, 12
autolayer_fmx_continuous(), 11
autolayer_fmx_discrete, 11, 11, 12
autoplot, 12
autoplot.fmx, 12, 12
autoplot.fmxS, 13
autoplot.fmxS(), 13

BIC, 35, 47, 53

character, 10–14, 17, 20, 21, 23–27, 29–32,
    34, 37, 40, 45–48, 53, 56
chol.default, 16
chol2inv, 16
CK5, 13
clusterList, 14
clusterList(), 14
coef.fmx, 14
coef.fmx(), 15
complex, 44
confint.fmx, 15, 15
CramerVonMises_fmx (fmx_diagnosis), 28
CramerVonMises_fmx(), 28
crossprod_inv, 16
crossprod_inv(), 16
cvm.test, 28
CycD1, 16

dbl2fmx, 17
dbl2fmx(), 17, 25
density.default, 3
dfmx, 17
dfmx(), 19
dGH (TukeyGH), 54
dGH(), 23, 39, 40, 55
dist_logtrans, 21, 21
distArgs, 20
distArgs(), 20
distType, 20, 21
dnorm, 23, 39, 40, 54
double, 13, 16, 23, 27, 28, 33–35, 43, 44,
    54–56, 58
drop1.fmx, 22
drop1.fmx (drop1_fmx), 21
drop1_fmx, 21
dsn, 39, 40
dst, 39, 40
duplicated.default, 56
duplicated_allequal, 56
duplicated_allequal (ud_allequal), 55

ecdf, *32*

fitdist, *5*, *34, 35*, *41, 42*
fmx, *4–12*, *14, 15*, *18, 19*, *21, 22*, 22, *23–31*, *35*, *40*, *42–46*, *50*, *53*, *56, 57*, *59*
fmx(), *23*
fmx-class, 23
fmx2dbl, 24
fmx2dbl(), *17*, *25*
fmx_cluster, 26, *29*
fmx_cluster(), *26*
fmx_constraint, *27*, 27
fmx_constraint(), *26*, *45*
fmx_diagnosis, 28
fmx_hybrid, *29*, 29
fmx_hybrid(), *45, 46*
fmx_normix, *29, 30*, 30
fmxS, *13*, *25*, 25
fmxS(), *25*
fmxS-class *(*fmxS*)*, 25
formalArgs, *20*
function, *3*, *10*, *24*, *32*, *58*

gammamixEM, *6*, *36, 37*
geom_function, *30, 31*
geom_function_args, 30
geom_function_args(), *31*
getTeX, 31
getTeX(), *31*
ggplot, *12, 13*, *31*

integer, *10, 11*, *14*, *17*, *19*, *21*, *24*, *26, 27*, *30*, *42, 43*, *45, 46*, *48*, *54*, *56*, *58, 59*

kmeans, *14*, *50*
Kolmogorov_dist, 32
Kolmogorov_dist(), *32*
Kolmogorov_fmx *(*fmx_diagnosis*)*, 28
Kolmogorov_fmx(), *28, 29*
ks.test, *32, 33*
KullbackLeibler_fmx *(*fmx_diagnosis*)*, 28
KullbackLeibler_fmx(), *28*

layer, *10, 11*, *31*
letterV_B, *34*
letterV_B *(*letterValue*)*, 33
letterV_B_g_h, *34*
letterV_B_g_h *(*letterValue*)*, 33
letterV_B_h, *34*
letterV_B_h *(*letterValue*)*, 33
letterV_g, *34*
letterV_g *(*letterValue*)*, 33
letterValue, 33, *34*
letterValue(), *27*
list, *11*, *13, 14*, *16, 17*, *22*, *25*, *31*, *47*, *53*
logical, *15*, *18*, *31*, *44*, *51, 52*, *54*, *56, 57*, *59*
logLik, *35, 36*, *43*
logLik.fitdist, 35
logLik.fitdist(), *35*
logLik.fmx, 35, *36*
logLik.fmx(), *19*
logLik.mixEM, 36
logLik.mixEM(), *36*

mad, *26*
mahalanobis, *36*, *49*
mahalanobis_int, 36, *37*
matrix, *15, 16*, *23, 24*, *27*, *36*, *44*, *49*, *57*
median, *26*
mixEM_pars, 37
mixEM_pars(), *37*
mlogis, 37
moment, *40*, *51*
moment-class *(*moment_*)*, 39
moment_, 39
moment_(), *40*
moment_.character(), *39, 40*
moment_.fmx(), *39, 40*
moment_GH *(*moment_*)*, 39
moment_GH(), *40*
moment_norm *(*moment_*)*, 39
moment_norm(), *40*
moment_sn *(*moment_*)*, 39
moment_sn(), *40*
moment_st *(*moment_*)*, 39
moment_st(), *40*

nobs, *42*
nobs.fitdist, 41
nobs.fitdist(), *42*
nobs.fmx, *42*, 42
normal, *23*
normalmixEM, *30*, *36, 37*, *51*
npar.fmx, *43*, 43
numeric, *3–11*, *14, 15*, *17–19*, *23–26*, *28–30*, *32*, *35–40*, *45, 46*, *48, 49*, *58*

optim, *45*

outer, *44*
outer_allequal, 43
outer_allequal(), *44*, *56*

pfmx, *19*
pfmx *(*dfmx*)*, 17
pfmx(), *19*
pGH, *19*
pGH *(*TukeyGH*)*, 54
pGH(), *55*
plogis, *37, 38*
plot.mixEM, *6*
pmlogis_first *(*mlogis*)*, 37
pmlogis_first(), *38*
pmlogis_last *(*mlogis*)*, 37
pmlogis_last(), *38*
print, *44*
print.fmx, *44*, 44

qfmx, *19*
qfmx *(*dfmx*)*, 17
qfmx(), *19*
qGH *(*TukeyGH*)*, 54
qGH(), *55*
qGH2z *(*TukeyGH*)*, 54
qGH2z(), *55*
QLMDe, *10*, *42*, 45, *47*
QLMDe(), *17*, *46*, *48*
QLMDe_stepK, 46
QLMDe_stepK(), *47*
QLMDp, *45*, 47
QLMDp(), *45*, *48*
qlogis, *37, 38*
qmlogis_first *(*mlogis*)*, 37
qmlogis_first(), *38*
qmlogis_last *(*mlogis*)*, 37
qmlogis_last(), *38*
qnorm, *19*
qr.default, *16*
qr.R, *16*
quantile, *19*, *24*
quantile_vcov, 48
quantile_vcov(), *46*, *49*

reAssign, 49
reAssign.tkmeans(), *14*, *50*
rfmx *(*dfmx*)*, 17
rfmx(), *19*
rGH *(*TukeyGH*)*, 54
rGH(), *55*

show, *50, 51*
show,fmx-method, 50
show,moment-method, 51
smsn.mix, *6–9*, *52*
sort, *52*
sort.mixEM, *51*, 51
sort.Normal *(*sort_mixsmsn*)*, 52
sort.Normal(), *52*
sort.Skew.normal *(*sort_mixsmsn*)*, 52
sort.Skew.normal(), *52*
sort.Skew.t *(*sort_mixsmsn*)*, 52
sort.Skew.t(), *52*
sort.t *(*sort_mixsmsn*)*, 52
sort_mixsmsn, 52
stat_function, *10*, *31*
step, *22*, *53*
step_fmx, *53*, 53
step_fmx(), *47*

tkmeans, *14*, *30*, *49, 50*
TukeyGH, 54

ud_allequal, 55
unique.default, *56*
unique_allequal, *56*
unique_allequal *(*ud_allequal*)*, 55
user_constraint, *56*, 56

vcov.fmx, *57*, 57
vector, *3–11*, *13–21*, *23–26*, *28–30*, *32–40*, *43–46*, *48, 49*, *54–56*, *58, 59*
vuniroot, *19*, *58*
vuniroot2, 58
vuniroot2(), *45*, *54*, *58*

z2qGH *(*TukeyGH*)*, 54
z2qGH(), *55*