

Package ‘fmx’

May 8, 2026

Type Package

Title Finite Mixture Parametrization

Version 0.1.3

Date 2025-03-15

Description A parametrization framework for finite mixture distribution using S4 objects. Density, cumulative density, quantile and simulation functions are defined. Currently normal, Tukey g-&-h, skew-normal and skew-t distributions are well tested. The gamma, negative binomial distributions are being tested.

License GPL-2

Imports methods, goftest, sn, VGAM, param2moment, TukeyGH77

Encoding UTF-8

Language en-US

Depends R (>= 4.4.0)

Suggests fitdistrplus, ggplot2, mixtools, mixsmsn, scales

RoxygenNote 7.3.2

NeedsCompilation no

Author Tingting Zhan [aut, cre] (ORCID:

<<https://orcid.org/0000-0001-9971-4844>>),

Inna Chervoneva [ctb] (ORCID: <<https://orcid.org/0000-0002-9104-4505>>)

Maintainer Tingting Zhan <tingtingzhan@gmail.com>

Repository CRAN

Date/Publication 2025-03-15 22:20:06 UTC

Contents

fmx-package	2
approxdens	2
as.fmx	3
as.fmx.fitdist	4

as.fmx.mixEM	4
as.fmx.Normal	5
as.fmx.Skew.normal	6
as.fmx.Skew.t	7
as.fmx.t	8
dfmx	9
fmx	12
fmx-class	13
fmx_constraint	14
Kolmogorov_dist	15
mlogis	16
moment2fmx	17
moment_fmx	18

Index	19
--------------	-----------

fmx-package	<i>Finite Mixture Parametrization</i>
-------------	---------------------------------------

Description

A parametrization framework for finite mixture distribution using S4 objects.

Density, cumulative density, quantile and simulation functions are defined.

Currently normal, Tukey g -&- h , skew-normal and skew- t distributions are well tested. The gamma, negative binomial distributions are being tested.

Author(s)

Maintainer: Tingting Zhan <tingtingzhan@gmail.com> ([ORCID](#))

Other contributors:

- Inna Chervoneva <Inna.Chervoneva@jefferson.edu> ([ORCID](#)) [contributor]

approxdens	<i>Empirical Density Function</i>
------------	-----------------------------------

Description

..

Usage

approxdens(x, ...)

Arguments

x [numeric vector](#), observations
... additional parameters of [density.default](#)

Details

[approx](#) inside [density.default](#)
another 'layer' of [approxfun](#)

Value

Function [approxdens\(\)](#) returns a [function](#).

Examples

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

as.fmx

Turn Various Objects to [fmx](#) Class

Description

Turn various objects created in other R packages to [fmx](#) class.

Usage

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

Arguments

x an R object
... additional parameters, see **Arguments** in individual S3 dispatches

Details

Various mixture distribution estimates obtained from other R packages are converted to [fmx](#) class, so that we could take advantage of all methods defined for [fmx](#) objects.

Value

S3 generic function [as.fmx\(\)](#) returns an [fmx](#) object.

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

Description

To convert `fitdist` objects (from package `fitdistrplus`) to `fmx` class.

Usage

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

Arguments

```
x                    fitdist object
...                   ..
```

Value

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

Examples

```
library(fitdistrplus)
# ?fitdist
data(endosulfan, package = 'fitdistrplus')
ATV <- subset(endosulfan, group == 'NonArthroInvert')$ATV
log10ATV <- log10(ATV)
fln <- fitdist(log10ATV, distr = 'norm')
(fln2 <- as.fmx(fln))
hist.default(log10ATV, freq = FALSE)
curve(dfmx(x, dist = fln2), xlim = range(log10ATV), add = TRUE)
```

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

Description

To convert `mixEM` objects (from package `mixtools`) to `fmx` class.

Currently only the returned value of `normalmixEM` and `gammamixEM` are supported

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)
(wait = as.fmx(normalmixEM(faithful$waiting, k = 2)))
hist.default(faithful$waiting, freq = FALSE)
curve(dfmx(x, dist = wait), xlim = range(faithful$waiting), add = TRUE)
```

as.fmx.Normal	Convert Normal fit from Rhref https://CRAN.R-project.org/package=mixsmsn to <i>fmx</i>
---------------	---

Description

To convert Normal object (from package `mixsmsn`) to `fmx` class.

Usage

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

Arguments

x	'Normal' object, returned from <code>smsn.mix</code> 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)
hist(x, freq = FALSE)
curve(dfmx(x, dist = m2a), xlim = range(x), add = TRUE)

```

as.fmx.Skew.normal *Convert Skew.normal Object to fmx*

Description

To convert `Skew.normal` object (from package **mixsmsn**) to `fmx` class.

Usage

```

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

```

Arguments

<code>x</code>	'Skew.normal' object, returned from <code>smsn.mix</code> with parameter <code>family = 'Skew.normal'</code> .
<code>data</code>	numeric vector
<code>...</code>	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(unnname(arg1), unnname(arg2), unnname(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))
hist(x, freq = FALSE)
curve(dfmx(x, dist = m1a), xlim = range(x), add = TRUE)

```

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

Description

To convert Skew.t object (from package **mixsmsn**) to **fmx** class.

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(unnname(arg1), unnname(arg2), unnname(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)
hist(x, freq = FALSE)
curve(dfmx(x, dist = m3a), xlim = range(x), add = TRUE)
(l3a = logLik(m3a))
stopifnot(all.equal.numeric(AIC(l3a), m3$aic), all.equal.numeric(BIC(l3a), m3$bic))
```

as.fmx.t *Convert t fit from* [Rhrefhttps://CRAN.R-project.org/package=mixsmsn](https://CRAN.R-project.org/package=mixsmsn) *to fmx*

Description

To convert t object (from package **mixsmsn**) to **fmx** class.

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(unnname(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)
# as.fmx(m4, data = x) # not ready yet!!

```

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 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 vuniroot2 and 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 *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`.

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

Note

Function `qnorm` returns an unnamed **vector** of quantiles, although `quantile` returns a named **vector** of quantiles.

Examples

```
library(ggplot2)

(e1 = fmx('norm', mean = c(0,3), sd = c(1,1.3), w = c(1, 1)))
curve(dfmx(x, dist = e1), xlim = c(-3,7))
ggplot() + geom_function(fun = dfmx, args = list(dist = e1)) + xlim(-3,7)
ggplot() + geom_function(fun = pfmx, args = list(dist = e1)) + xlim(-3,7)
hist(rfmx(n = 1e3L, dist = e1), main = '1000 obs from e1')

x = (-3):7
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)))
ggplot() + geom_function(fun = dfmx, args = list(dist = e2)) + xlim(-3,7)

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', g = .2, h = .01)) # one-component Tukey
ggplot() + geom_function(fun = dfmx, args = list(dist = e3)) + xlim(-3,5)
set.seed(124); r1 = rfmx(1e3L, dist = e3);
set.seed(124); r2 = TukeyGH77::rGH(n = 1e3L, g = .2, h = .01)
stopifnot(identical(r1, r2)) # 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))
```

```

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))
library(ggplot2)
(p = ggplot() +
  geom_function(fun = pfmx, args = list(dist = a1), mapping = aes(color = 'g2=h1=0')) +
  geom_function(fun = pfmx, args = list(dist = a2), mapping = aes(color = 'g2=0')) +
  xlim(3,15) +
  scale_y_continuous(labels = scales::percent) +
  labs(y = NULL, color = 'models') +
  coord_flip())
p + theme(legend.position = 'none')

# to use [rfmx] without \pkg{fmx}
(d = fmx(distname = 'GH', A = c(-1,1), B = c(.9,1.1), g = c(.3, -.2), h = c(.1, .05), w = c(2,3)))
d@pars
set.seed(14123); x = rfm(x = 1e3L, dist = d)
set.seed(14123); x_raw = rfm(x = 1e3L,
  distname = 'GH', K = 2L,
  pars = rbind(
    c(A = -1, B = .9, g = .3, h = .1),
    c(A = 1, B = 1.1, g = -.2, h = .05)
  ),
  w = c(.4, .6)
)
stopifnot(identical(x, x_raw))

```

fmx

*Create **fmx** Object for Finite Mixture Distribution*

Description

To create **fmx** object for finite mixture distribution.

Usage

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

Arguments

<code>distname</code>	character scalar
<code>w</code>	(optional) numeric vector . Does not need to sum up to 1; $w/\text{sum}(w)$ will be used internally.
<code>...</code>	mixture distribution parameters. See function dGH for the names and default values of Tukey <i>g</i> -&- <i>h</i> distribution parameters, or dnorm for the names and default values of normal distribution parameters.

Value

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

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

fmx Class: Finite Mixture Parametrization

Description

An S4 object to specify the parameters and type of distribution of a one-dimensional finite mixture distribution.

Slots

`distname` **character** scalar, name of parametric distribution of the mixture components. Currently, normal ('norm') and Tukey *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 *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 the observations

`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

`Kolmogorov, CramerVonMises, KullbackLeibler` (optional) **numeric** scalars

fmx_constraint	<i>Parameter Constraint(s) of Mixture Distribution</i>
----------------	--

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

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

Value

Function `fmx_constraint()` returns the indices of internal parameters (only applicable to Tukey *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
```

Kolmogorov_dist	<i>One-Sample Kolmogorov Distance</i>
-----------------	---------------------------------------

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

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

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.

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

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)))

```

moment2fmx

Creates [fmx](#) Object with Given Component-Wise Moments

Description

Creates [fmx](#) Object with Given Component-Wise Moments

Usage

```
moment2fmx(distname, w, ...)
```

Arguments

distname	character scalar
w	numeric vector
...	numeric scalars, some or all of mean, sd, skewness and kurtosis (length will be recycled), see moment2param

Value

Function [moment2fmx\(\)](#) returns a [fmx](#) object.

Examples

```

m = c(-1.5, 1.5)
s = c(.9, 1.1)
sk = c(.2, -.3)
kt = c(.5, .75)
w = c(2, 3)
(d1 = moment2fmx(distname='GH', w=w, mean=m, sd=s, skewness=sk, kurtosis=kt))
moment_fmx(d1)
(d2 = moment2fmx(distname='st', w=w, mean=m, sd=s, skewness=sk, kurtosis=kt))
moment_fmx(d2)
library(ggplot2)
ggplot() +
  geom_function(aes(color = 'GH'), fun = dfmx, args = list(dist=d1), n = 1001) +
  geom_function(aes(color = 'st'), fun = dfmx, args = list(dist=d1), n = 1001) +
  xlim(-5, 6)
# two curves looks really close, but actually not identical
x = rfmx(n = 1e3L, dist = d1)
range(dfmx(x, dist = d1) - dfmx(x, dist = d2))

```

moment_fmx

Moment of Each Component in an [fmx](#) Object

Description

To find moments of each component in an [fmx](#) object.

Usage

```
moment_fmx(object)
```

Arguments

object an [fmx](#) object

Details

Function [moment_fmx\(\)](#) calculates the [moments](#) and distribution characteristics of each mixture component of an S4 [fmx](#) object.

Value

Function [moment_fmx\(\)](#) returns a [moment](#) object.

Examples

```

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

```

Index

approx, [3](#)
approxdens, [2](#)
approxdens(), [3](#)
approxfun, [3](#)
as.fmx, [3](#)
as.fmx(), [3](#)
as.fmx.fitdist, [4](#)
as.fmx.fitdist(), [4](#)
as.fmx.mixEM, [4](#)
as.fmx.mixEM(), [5](#)
as.fmx.Normal, [5](#)
as.fmx.Normal(), [5](#)
as.fmx.Skew.normal, [6](#)
as.fmx.Skew.normal(), [6](#)
as.fmx.Skew.t, [7](#)
as.fmx.Skew.t(), [7](#)
as.fmx.t, [8](#)
as.fmx.t(), [8](#)

character, [12–15](#), [17](#)

density.default, [3](#)
dfmx, [9](#)
dfmx(), [10](#), [11](#)
dGH, [12](#)
dnorm, [12](#)
double, [13](#), [14](#)

ecdf, [15](#)

fitdist, [4](#)
fmx, [3–8](#), [10–12](#), [12](#), [13](#), [14](#), [17](#), [18](#)
fmx(), [13](#)
fmx-class, [13](#)
fmx-package, [2](#)
fmx_constraint, [14](#)
fmx_constraint(), [14](#)
function, [3](#), [15](#)

gammamixEM, [4](#), [5](#)

integer, [10](#), [14](#)

Kolmogorov_dist, [15](#)
Kolmogorov_dist(), [15](#)
ks.test, [15](#)

logical, [10](#)
logLik.fmx(), [10](#)

matrix, [13](#), [14](#)
mlogis, [16](#)
moment, [18](#)
moment2fmx, [17](#)
moment2fmx(), [17](#)
moment2param, [17](#)
moment_fmx, [18](#)
moment_fmx(), [18](#)

normal, [13](#)
normalmixEM, [4](#)
numeric, [3](#), [5–8](#), [10–13](#), [15–17](#)

pfmx, [11](#)
pfmx (dfmx), [9](#)
pfmx(), [11](#)
pGH, [11](#)
plogis, [16](#)
plot.mixEM, [5](#)
pmlogis_first, [16](#)
pmlogis_first (mlogis), [16](#)
pmlogis_last, [16](#)
pmlogis_last (mlogis), [16](#)

qfmx, [11](#)
qfmx (dfmx), [9](#)
qfmx(), [11](#)
qlogis, [16](#)
qmlogis_first, [16](#)
qmlogis_first (mlogis), [16](#)
qmlogis_last, [16](#)
qmlogis_last (mlogis), [16](#)

qnorm, [11](#)

quantile, [11](#)

rfmx (dfmx), [9](#)

rfmx(), [11](#)

smsn.mix, [5–8](#)

vector, [3](#), [5–8](#), [10–13](#), [15–17](#)

vuniroot, [10](#)

vuniroot2, [10](#)