

Package: fmx (via r-universe)

August 29, 2024

Type Package

Title Finite Mixture Parametrization

Version 0.1.2

Date 2024-04-30

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 7.3.1

NeedsCompilation no

Author Tingting Zhan [aut, cre, cph]
(<<https://orcid.org/0000-0001-9971-4844>>), Inna Chervoneva
[ctb, cph] (<<https://orcid.org/0000-0002-9104-4505>>)

Maintainer Tingting Zhan <tingtingzhan@gmail.com>

Date/Publication 2024-04-30 14:20:09 UTC

Repository <https://tingtingzhan.r-universe.dev>

RemoteUrl <https://github.com/cran/fmx>

RemoteRef HEAD

RemoteSha b10d88cdacd84958d1f4a7e0c296d48b98bc43be

Contents

fmx-package	3
approxdens	3
as.fmx	4
as.fmx.fitdist	4
as.fmx.mixEM	5
as.fmx.Normal	6
as.fmx.Skew.normal	7
as.fmx.Skew.t	8
as.fmx.t	9
coef.fmx	10
confint.fmx	10
dbl2fmx	11
dfmx	12
distArgs	15
distType	15
dist_logtrans	16
fmx	16
fmx-class	17
fmx2dbl	18
fmx_constraint	19
fmx_diagnosis	20
getTeX	21
Kolmogorov_dist	21
logLik.fitdist	22
logLik.fmx	23
logLik.mixEM	24
MaP	24
mixEM_pars	25
mlogis	25
moment2fmx	27
moment_fmx	28
nobs.fitdist	28
npar.fmx	29
print.fmx	29
qfmx_interval	30
show,fmx-method	30
sort.mixEM	31
sort_mixsmsn	32
user_constraint	33
vcov.fmx	34
[,fmx,ANY,ANY,ANY-method	34

Index

36

fmx-package

Finite Mixture Parametrization

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](#)) [copyright holder]

Other contributors:

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

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

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	<i>Convert</i> Normal <i>fit</i> <i>from</i> Rhrefhttps://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 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(unnamed(arg1), unnamed(arg2), unnamed(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

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(unnamed(arg1), unnamed(arg2), unnamed(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	<i>Convert Skew.t fit from R</i> <i>hrefhttps://CRAN.R-project.org/package=mixsmsn to fmx</i>
---------------	---

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 <code>smsn.mix</code> with parameter <code>family = 'Skew.t'</code>
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	<i>Convert t fit from</i>	Rhrefhttps://CRAN.R-project.org/package=mixsmsn to <i>fmx</i>
----------	---------------------------	--

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(unnamed(arg1), unnamed(arg2), unnamed(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!!
```

 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 <i>g</i> -and- <i>h</i> 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, function [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, function [confint.fmx](#) does not return the confident intervals of for the constrained parameters.

Value

[confint.fmx](#) returns a [matrix](#)

dbl2fmx	<i>Inverse of fmx2dbl, for internal use</i>
---------	---

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 downstream function `QuantileGH::QLMDe` and unexported function `QuantileGH:::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 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,  
  interval = qfmx_interval(dist = dist),  
  ...,  
  lower.tail = TRUE,  
  log.p = FALSE  
)
```

```

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

```

Arguments

<code>x, q</code>	numeric vector , quantiles, <code>NA_real_</code> value(s) allowed.
<code>dist</code>	fmx object, a finite mixture distribution
<code>distname, K, pars, w</code>	auxiliary parameters, whose default values are determined by argument <code>dist</code> . The user-specified vector of <code>w</code> does not need to sum up to 1; $w/\text{sum}(w)$ will be used internally.
<code>...</code>	additional parameters
<code>log, log.p</code>	logical scalar. If TRUE, probabilities are given as $\log(p)$.
<code>lower.tail</code>	logical scalar. If TRUE (default), probabilities are $Pr(X \leq x)$, otherwise, $Pr(X > x)$.
<code>p</code>	numeric vector , probabilities.
<code>interval</code>	length two numeric vector , interval for root finding, see <code>vuniroot2</code> and <code>vuniroot</code>
<code>n</code>	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))

```

distArgs	<i>Name(s) of Formal Argument(s) of Distribution</i>
----------	--

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	<i>Distribution Type</i>
----------	--------------------------

Description

..

Usage

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

Arguments

type [character](#) scalar

Value

Function [distType](#) returns a [character vector](#).

dist_logtrans	<i>Distribution Parameters that needs to have a log-transformation</i>
---------------	--

Description

..

Usage

```
dist_logtrans(distname)
```

Arguments

distname [character](#) scalar, name of distribution

Value

Function [dist_logtrans](#) returns an [integer](#) scalar

fmx	<i>Create fmx Object for Finite Mixture Distribution</i>
-----	--

Description

To create [fmx](#) object for finite mixture distribution.

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 *g-&-h* 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

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	<i>fmx</i> 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

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

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](#)

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
```

fmx_diagnosis	<i>Diagnoses for fmx Estimates</i>
---------------	------------------------------------

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 <code>object@data</code> , 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`

getTeX	<i>TeX Label (of Parameter Constraint(s)) of fmx Object</i>
--------	---

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	<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

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.

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 = pnbinom, size = 500, prob = .4)) # warning on ties
Kolmogorov_dist(x2, null = pnbinom, size = 500, prob = .4) # wont be the same
```

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.

Value

Function `logLik.fitdist` returns a [logLik](#) object, which could be further used by [AIC](#) and [BIC](#).
 (I have written to the authors)

logLik.fmx	<i>Log-Likelihood of fmx Object</i>
------------	---

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

Function `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

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

logLik.mixEM	<i>Log-Likelihood of 'mixEM' Object</i>
--------------	---

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 gam-mamixEM are supported
...	additional parameters, currently not in use

Value

Function [logLik.mixEM](#) returns a [logLik](#) object.

MaP	<i>Maximum a Posteriori clustering</i>
-----	--

Description

..

Usage

```
MaP(x, dist, ...)
```

Arguments

x	numeric vector
dist	an fmx object
...	..

Value

Function [MaP](#) returns an [integer vector](#).

Examples

```
x = rnorm(1e2L, sd = 2)
m = fmx('norm', mean = c(-1.5, 1.5), w = c(1, 2))
library(ggplot2)
ggplot() + geom_function(fun = dfmx, args = list(dist = m)) +
  geom_point(mapping = aes(x = x, y = .05, color = factor(MaP(x, dist = m)))) +
  labs(color = 'Maximum a Posteriori\nClustering')
```

mixEM_pars	<i>Names of Distribution Parameters of 'mixEM' Object</i>
------------	---

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	<i>Multinomial Probabilities & Logits</i>
--------	---

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	<i>Moment of Each Component in an fmx Object</i>
------------	--

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)
```

nobs.fitdist	<i>Number of Observations in fitdist Object</i>
--------------	---

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

<code>npar.fmx</code>	<i>Number of Parameters of <code>fmx</code> Object</i>
-----------------------	--

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

Function `npar.fmx` returns an `integer` scalar.

<code>print.fmx</code>	<i>S3 <code>print</code> of <code>fmx</code> Object</i>
------------------------	---

Description

..

Usage

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

Arguments

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

Value

Function `print.fmx` returns the input `fmx` object invisibly.

qfmx_interval	Obtain interval for <i>vuniroot2</i> for Function <i>qfmx</i>
---------------	---

Description

Obtain interval for *vuniroot2* for Function *qfmx*

Usage

```
qfmx_interval(
  dist,
  p = c(1e-06, 1 - 1e-06),
  distname = dist@distname,
  K = dim(pars)[1L],
  pars = dist@pars,
  w = dist@w,
  ...
)
```

Arguments

dist	<i>fmx</i> object
p	length-2 numeric vector
distname, K, pars, w	(optional) ignored if dist is provided
...	additional parameters, currently not used

Value

Function *qfmx_interval* returns a length-2 numeric vector.

show, fmx-method	Show <i>fmx</i> 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 <i>fmx</i> object
--------	----------------------

Value

The [show](#) method for [fmx](#) 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 <i>mu</i> 's be increasing (FALSE, default) or decreasing (TRUE)?
...	additional parameters, currently not in use

Details

[normalmixEM](#) does *not* order the location parameter

Value

Function [sort.mixEM](#) returns a 'mixEM' object.

See Also

[sort](#)

sort_mixsmsn	<i>Sort</i>	<i>Objects</i>	<i>from</i>	<i>Rhrefhttps://CRAN.R-project.org/package=mixsmsn</i>
	<i>mixsmsn by Location Parameters</i>			

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](#)

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 *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

Function `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 <i>g</i> -and- <i>h</i> mixture), or for the internal/unconstrained parameters (TRUE).
...	place holder for S3 naming convention

Details

Function `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, function `vcov.fmx` does not return the variance/covariance involving the constrained parameters.

Value

Function `vcov.fmx` returns a [matrix](#).

 [,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

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

[, 35
[, fmx, ANY, ANY, ANY-method, 34

AIC, 23
approx, 3
approxdens, 3, 3
approxfun, 3
as.fmx, 4, 4
as.fmx.fitdist, 4, 4
as.fmx.mixEM, 5, 5
as.fmx.Normal, 6, 6
as.fmx.Skew.normal, 7, 7
as.fmx.Skew.t, 8, 8
as.fmx.t, 9, 9

BIC, 23

character, 11, 15–19, 21, 22, 25, 27, 33
coef.fmx, 10, 10
confint.fmx, 10, 11
CramerVonMises_fmx, 20
CramerVonMises_fmx (fmx_diagnosis), 20
cvm.test, 20

dbl2fmx, 11, 11, 18
density.default, 3
dfmx, 12, 13
dGH, 16
dist_logtrans, 16, 16
distArgs, 15, 15
distType, 15, 16
dnorm, 16
double, 17, 19, 20, 23

ecdf, 22

fitdist, 4, 22, 23, 28
fmx, 4–11, 13, 16, 16, 17–21, 23, 24, 27–31, 33–35
fmx-class, 17
fmx-package, 3

fmx2dbl, 11, 18, 18
fmx_constraint, 19, 19
fmx_diagnosis, 20
formalArgs, 15
function, 3, 22

gammamixEM, 5, 24, 25
getTeX, 21, 21

integer, 11, 13, 16, 18, 19, 24, 29, 33, 35

Kolmogorov_dist, 21, 22
Kolmogorov_fmx, 20
Kolmogorov_fmx (fmx_diagnosis), 20
ks.test, 22
KullbackLeibler_fmx, 20
KullbackLeibler_fmx (fmx_diagnosis), 20

length, 30
list, 11
logical, 10, 13, 21, 31, 32, 34, 35
logLik, 23, 24, 29
logLik.fitdist, 22, 23
logLik.fmx, 13, 23, 23
logLik.mixEM, 24, 24

MaP, 24, 24
matrix, 11, 17–19, 34
mixEM_pars, 25, 25
mlogis, 25
moment, 28
moment2fmx, 27, 27
moment2param, 27
moment_fmx, 28, 28

nobs.fitdist, 28, 29
normal, 17
normalmixEM, 5, 24, 25, 31
npar.fmx, 29, 29
numeric, 3, 5–11, 13, 16–18, 20, 22–24, 26, 27, 30

pfmx, [13](#)
pfmx (dfmx), [12](#)
pGH, [13](#)
plogis, [25](#), [26](#)
plot.mixEM, [5](#)
pmlogis_first, [26](#)
pmlogis_first (mlogis), [25](#)
pmlogis_last, [26](#)
pmlogis_last (mlogis), [25](#)
print, [29](#)
print.fmx, [29](#), [29](#)

qfmx, [13](#), [30](#)
qfmx (dfmx), [12](#)
qfmx_interval, [30](#), [30](#)
qlogis, [25](#), [26](#)
qmlogis_first, [26](#)
qmlogis_first (mlogis), [25](#)
qmlogis_last, [26](#)
qmlogis_last (mlogis), [25](#)
qnorm, [14](#)
quantile, [14](#)

rfmx, [13](#)
rfmx (dfmx), [12](#)

show, [31](#)
show, fmx-method, [30](#)
smsn.mix, [6–9](#), [32](#)
sort, [31](#), [32](#)
sort.mixEM, [31](#), [31](#)
sort.Normal, [32](#)
sort.Normal (sort_mixsmsn), [32](#)
sort.Skew.normal, [32](#)
sort.Skew.normal (sort_mixsmsn), [32](#)
sort.Skew.t, [32](#)
sort.Skew.t (sort_mixsmsn), [32](#)
sort.t (sort_mixsmsn), [32](#)
sort_mixsmsn, [32](#)

user_constraint, [33](#), [33](#)

vcov.fmx, [34](#), [34](#)
vector, [3](#), [5–11](#), [13–18](#), [20](#), [22–27](#), [30](#), [33](#), [35](#)
vuniroot, [13](#)
vuniroot2, [13](#), [30](#)