Package 'nor1mix'

November 14, 2023

Title Normal aka Gaussian 1-d Mixture Models

Version 1.3-2

Date 2023-11-12

Description Onedimensional Normal (i.e. Gaussian) Mixture Models (S3) Classes, for, e.g., density estimation or clustering algorithms research and teaching; providing the widely used Marron-Wand densities. Efficient random number generation and graphics. Fitting to data by efficient ML (Maximum Likelihood) or traditional EM estimation.

Imports stats, graphics

Suggests cluster, copula

URL https://curves-etc.r-forge.r-project.org/,
https://r-forge.r-project.org/R/?group_id=846,
https://r-forge.r-project.org/scm/viewvc.php/pkg/nor1mix/?root=curves-etc,
svn://svn.r-forge.r-project.org/svnroot/curves-etc/pkg/nor1mix

BugReports https://r-forge.r-project.org/R/?group_id=846

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation no

Author Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>),
Friedrich Leisch [ctb] (norMixEM(),
<https://orcid.org/0000-0001-7278-1983>),
Erik Jørgensen [ctb] (pnorMix(), qnorMix())

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Repository CRAN

Date/Publication 2023-11-14 14:10:02 UTC

R topics documented:

nor1mix-package 2
clus2norMix 3

dnorMix 4
llnorMix 6
MarronWand 8
norMix 10
norMix2call 12
norMixFit 14
plot.norMix 16
pnorMix 17
r.norMix 19
rnorMix 20
sort.norMix 21

Index **22**

`nor1mix-package` *Normal aka Gaussian 1-d Mixture Models*

Description

Onedimensional Normal (i.e. Gaussian) Mixture Models (S3) Classes, for, e.g., density estimation or clustering algorithms research and teaching; providing the widely used Marron-Wand densities. Efficient random number generation and graphics. Fitting to data by efficient ML (Maximum Likelihood) or traditional EM estimation.

Details

The DESCRIPTION file:

Package:	nor1mix
Title:	Normal aka Gaussian 1-d Mixture Models
Version:	1.3-2
Date:	2023-11-12
Authors@R:	c(person("Martin", "Maechler", role = c("aut","cre"), email = "maechler@stat.math.ethz.ch", comment = c(OR
Description:	Onedimensional Normal (i.e. Gaussian) Mixture Models (S3) Classes, for, e.g., density estimation or clustering
Imports:	stats, graphics
Suggests:	cluster, copula
URL:	https://curves-etc.r-forge.r-project.org/, https://r-forge.r-project.org/R/?group_id=846, https://r-forge.r-project.o
BugReports:	https://r-forge.r-project.org/R/?group_id=846
License:	GPL (>= 2)
Encoding:	UTF-8
Author:	Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Friedrich Leisch [ctb] (norMixEM(), <
Maintainer:	Martin Maechler <maechler@stat.math.ethz.ch>

Index of help topics:

```
MarronWand              Marron-Wand Densities as 'norMix' Objects
clus2norMix             Transform Clustering / Grouping to Normal
```

	Mixture
dnorMix	Normal Mixture Density
llnorMix	Likelihood, Parametrization and EM-Steps For 1D Normal Mixtures
nor1mix-package	Normal aka Gaussian 1-d Mixture Models
norMix	Mixtures of Univariate Normal Distributions
norMix2call	Transform "norMix" object into Call, Expression or Function
norMixEM	EM and MLE Estimation of Univariate Normal Mixtures
plot.norMix	Plotting Methods for 'norMix' Objects
pnorMix	Normal Mixture Cumulative Distribution and Quantiles
r.norMix	Ratio of Normal Mixture to Corresponding Normal
rnorMix	Generate 'Normal Mixture' Distributed Random Numbers
sort.norMix	Sort Method for "norMix" Objects

Note that direct Maximum Likelihood ML (via `optim()`) is typically much faster converging (and more reliably detecting convergence correctly), notably thanks to a smart re-parametrization: use `norMixMLE()`.

Author(s)

Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Friedrich Leisch [ctb] (norMixEM(), <https://orcid.org/0000-0001-7278-1983>), Erik Jørgensen [ctb] (pnorMix(), qnorMix())

Maintainer: Martin Maechler <maechler@stat.math.ethz.ch>

See Also

The Marron-Wand examples of normal (gaussian) mixtures `MarronWand`.

Multivariate distributions from copulas `Mvdc` from the **copula** package can use `norMix` marginals.

Examples

```
example(dnorMix)
```

`clus2norMix` *Transform Clustering / Grouping to Normal Mixture*

Description

Simple transformation of a clustering or grouping to a normal mixture object (class `"norMix"`, see, `norMix`.

Usage

```
clus2norMix(gr, x, name = deparse(sys.call()))
```

Arguments

gr	a grouping/clustering vector with values in $\{1, .., K\}$; possibly a factor.
x	numeric vector of (original) data (of the same length as gr).
name	name for norMix() object; constructed from the call by default.

Value

A call to norMix() with (mu, sig2, w) set to the empirical values of the groups (as defined by split(x,gr).

Note

Via this function, any simple clustering algorithm (such pam) can be used as simple mixture model fitting procedure.

Author(s)

Martin Maechler, Dec. 2007

See Also

norMix; further pam() (or clara()) from package **cluster** for sensible clusterings.

Examples

```
x9 <- rnorMix(500, MW.nm9)
require("cluster")
pxc <- pam(x9, k=3)
plot(pxc, which = 2)# silhouette

(nm.p9 <- clus2norMix(pxc$clustering, x9))
plot(nm.p9, p.norm=FALSE)
lines(MW.nm9, col="thistle")
```

---

dnorMix *Normal Mixture Density*

Description

Evaluate the density function of the normal mixture specified as norMix object.

Usage

```
dnorMix (x, obj,        log = FALSE)

dnorMixL(obj, x = NULL, log = FALSE, xlim = NULL, n = 511)
dpnorMix(x, obj, lower.tail = TRUE)
```

Arguments

obj	an object of class norMix.
x	numeric vector with abscissa values where to evaluate the density (and probability, for dpnorMix()). For dnorMixL() by default, when NULL, it is constructed from n (and xlim if that is specified).
log	logical indicating *log*-density values should be returned.
xlim	range of abscissa values, used if x == NULL. By default, xlim is taken as mean plus/minus 3 standard deviations of the normal mixture.
n	number of abscissa values to generate if x is not specified.
lower.tail	logical; if TRUE (default), probabilities are $P[X \le x]$, otherwise, $P[X > x]$.

Value

dnorMix(x) returns the numeric vector of density values $f(x)$, logged if log is TRUE.

dnorMixL() returns a list with components

x	the abscissa values.
y	the density values $f(x)$ as for dnorMix().

dpnorMix() returns a list with components

d	the density values $f(x)$ as for dnorMix().
p	the probability values $F(x)$ as for pnorMix().

See Also

rnorMix for random number generation, and norMix for the construction and further methods, particularly plot.norMix which makes use dnorMix.

Examples

```
ff <- dnorMixL(MW.nm7)
str(ff)
plot(ff, type = "h", ylim = c(0,1)) # rather use plot(ff, ...)

x <- seq.int(-4,5, length.out = 501)
dp <- dpnorMix(x, MW.nm7)
lines(x, dp$d, col = "tomato", lwd=3)
lines(x, dp$p, col = 3, lwd=2)# does not fit y-wise
stopifnot(all.equal(dp$d, dnorMix(x, MW.nm7), tolerance=1e-12),
          all.equal(dp$p, pnorMix(x, MW.nm7), tolerance=1e-12))
```

`llnorMix` *Likelihood, Parametrization and EM-Steps For 1D Normal Mixtures*

Description

These functions work with an almost unconstrained parametrization of univariate normal mixtures.

`llnorMix(p, *)` computes the log likelihood,

`obj <- par2norMix(p)` maps parameter vector `p` to a `norMix` object `obj`,

`p <- nM2par(obj)` maps from a `norMix` object `obj` to parameter vector `p`,

where `p` is always a parameter vector in our parametrization.

Partly for didactical reasons, the following functions provide the basic ingredients for the EM algorithm (see also `norMixEM`) to parameter estimation:

`estep.nm(x, obj, p)` computes 1 E-step for the data `x`, given *either* a `"norMix"` object `obj` or parameter vector `p`.

`mstep.nm(x, z)` computes 1 M-step for the data `x` and the probability matrix `z`.

`emstep.nm(x, obj)` computes 1 E- and 1 M-step for the data `x` and the `"norMix"` object `obj`.

where again, `p` is a parameter vector in our parametrization, `x` is the (univariate) data, and `z` is a $n \times m$ `matrix` of (posterior) conditional probabilities, and θ is the full parameter vector of the mixture model.

Usage

```
llnorMix(p, x, m = (length(p) + 1)/3, trafo = c("clr1", "logit"))

par2norMix(p, trafo = c("clr1", "logit"), name = )

nM2par(obj, trafo = c("clr1", "logit"))

 estep.nm(x, obj, par)
 mstep.nm(x, z)
emstep.nm(x, obj)
```

Arguments

`p, par`	numeric vector: our parametrization of a univariate normal mixture, see details.
`x`	numeric: the data for which the likelihood is to be computed.
`m`	integer number of mixture components; this is not to be changed for a given `p`.
`trafo`	`character` string specifying the transformation of the component weight `w` m-vector (mathematical notation in `norMix`: $\pi_j, j = 1, \ldots, m$) to an $m-1$-dimensional unconstrained parameter vector in our parametrization. `"logit"` has been hard-wired upto **nor1mix** version 1.2-3, and has been replaced *as default* in 2019 for **nor1mix** version 1.2-4 by `"clr1"` which is more symmetric and basically Aitchinson's **c**entered **l**og **r**atio, see also CRAN package **compositions**' `clr()`.

name	(for par2norMix():) a name for the "norMix" object that is returned, uses a smart default.
obj	a "norMix" object, see norMix.
z	a $n \times m$ matrix of (posterior) conditional probabilities, $z_{ij} = P(x_i \in C_j|\theta)$, where C_j denotes the j-th group ("cluster").

Details

We use a parametrization of a (finite) m-component univariate normal mixture which is particularly apt for likelihood maximization, namely, one whose parameter space is *almost* a full $\mathbb{R}^M$, $M = 3m - 1$.

For an m-component mixture, we map to and from a parameter vector θ (== p as R-vector) of length $3m - 1$. For mixture density

$$\sum_{j=1}^{m} \pi_j \phi((t - \mu_j)/\sigma_j)/\sigma_j,$$

we transform the π_j (for $j \in 1, \ldots, m$) via the transform specified by trafo (see below), and log-transform the σ_j. Consequently, θ is partitioned into

p[1:(m-1)]: For

trafo = "logit": p[j]= logit(π_{j+1}) and π_1 is given implicitly as $\pi_1 = 1 - \sum_{j=2}^{m} \pi_j$.

trafo = "clr1": (**c**entered **l**og **r**atio, omitting 1st element): Set $\ell_j := \ln(\pi_j)$ for $j = 1, \ldots, m$, and p[j]= $\ell_{j+1} - 1/m \sum_{j'=1}^{m} \ell_{j'}$ for $j = 1, \ldots, m-1$.

p[m:(2m-1)]: p[m-1+ j]= μ_j, for j=1:m.

p[2m:(3m-1)]: p[2*m-1+ j] = $\log(\sigma_j)$, i.e., $\sigma_j^2 = exp(2 * p[. + j])$.

Value

llnorMix() returns a number, namely the log-likelihood.

par2norMix() returns "norMix" object, see norMix.

nM2par() returns the parameter vector θ of length $3m - 1$.

estep.nm() returns z, the matrix of (conditional) probabilities.

mstep.nm() returns the model parameters as a list with components w, mu, and sigma, corresponding to the arguments of norMix(). (and see the 'Examples' on using do.call(norMix, *) with it.)

emstep.nm() returns an *updated* "norMix" object.

Author(s)

Martin Maechler

See Also

norMix, logLik. Note that the log likelihood of a "norMix" object is directly given by sum(dnorMix(x, obj, log=TRUE)).

To fit, using the EM algorithm, rather use norMixEM() than the e.step, m.step, or em.step functions.

Note that direct likelihood maximization, i.e., MLE, is typically considerably more efficient than the EM, and typically converges well with our parametrization, see norMixMLE.

Examples

```
(obj <- MW.nm10) # "the Claw" -- m = 6 components
length(pp <- nM2par(obj)) # 17 == (3*6) - 1
par2norMix(pp)
## really the same as the initial 'obj' above

## Log likelihood (of very artificial data):
llnorMix(pp, x = seq.int(-2, 2, length.out = 1000))
set.seed(47)## of more realistic data:
x <- rnorMix(1000, obj)
llnorMix(pp, x)

## Consistency check :  nM2par()  and  par2norMix()  are inverses
all.EQ <- function(x,y, tol = 1e-15, ...) all.equal(x,y, tolerance=tol, ...)
stopifnot(all.EQ(pp, nM2par(par2norMix(pp))),
          all.EQ(obj, par2norMix(nM2par(obj)),
                    check.attributes=FALSE),
          ## Direct computation of log-likelihood:
          all.EQ(sum(dnorMix(x, obj, log=TRUE)),
                    llnorMix(pp, x))  )

## E- and M- steps : ------------------------------
rE1 <- estep.nm(x, obj)
rE2 <- estep.nm(x, par=pp) # the same as rE1
z <- rE1
str( rM <-  mstep.nm(x, z))
   (rEM <- emstep.nm(x, obj))

stopifnot(all.EQ(rE1, rE2),
          all.EQ(rEM, do.call(norMix, c(rM, name=""))))
```

MarronWand *Marron-Wand Densities as 'norMix' Objects*

Description

The fifteen density examples used in Marron and Wand (1992)'s simulation study have been used in quite a few subsequent studies, can all be written as normal mixtures and are provided here for convenience and didactical examples of normal mixtures. Number 16 has been added by Jansen et al.

Usage

```
MW.nm1  # Gaussian
MW.nm2  # Skewed
MW.nm2.old # Skewed(old)
MW.nm3  # Str Skew
MW.nm4  # Kurtotic
MW.nm5  # Outlier
MW.nm6  # Bimodal
MW.nm7  # Separated (bimodal)
MW.nm8  # Asymmetric Bimodal
MW.nm9  # Trimodal
MW.nm10 # Claw
MW.nm11 # Double Claw
MW.nm12 # Asymmetric Claw
MW.nm13 # Asymm. Double Claw
MW.nm14 # Smooth   Comb
MW.nm15 # Discrete Comb
MW.nm16 # Distant Bimodal
```

Author(s)

Martin Maechler

Source

They have been translated from Steve Marron's Matlab code,

now at

https://marronwebfiles.sites.oasis.unc.edu/OldResearch/parameters/nmpar.m, however for number 2, the Matlab code had `MW.nm2.old`; and I've defined `MW.nm2` as from the Annals paper; see also the last example below.

References

Marron, S. and Wand, M. (1992) Exact Mean Integrated Squared Error; *Annals of Statistcs* **20**, 712–736.

For number 16,
P. Janssen, J. S. Marron, N. Veraverbeke and W. Sarle (1995) Scale measures for bandwidth selection; *Journal of Nonparametric Statistics* **5**, 359–380. doi:10.1080/10485259508832654

Examples

```
MW.nm10
plot(MW.nm14)

## These are defined as norMix() calls in  ../R/zMarrWand-dens.R
nms <- ls(pattern = "^MW.nm", "package:nor1mix")
nms <- nms[order(as.numeric(substring(nms,6)))] # w/ warning for "2.old"
for(n in nms) {
```

```
   cat("\n",n,":\n"); print(get(n, "package:nor1mix"))
}

## Plot all of them:
op <- par(mfrow=c(4,4), mgp = c(1.2, 0.5, 0), tcl = -0.2,
          mar = .1 + c(2,2,2,1), oma = c(0,0,3,0))
for(n in nms[-17]) plot(get(n, "package:nor1mix"))
mtext("The Marron-Wand Densities", outer= TRUE, font= 2, cex= 1.6)

## and their Q-Q-plots (not really fast):
prob <- ppoints(N <- 100)
for(n in nms[-17]) # qnorMix() using monotone spline inversion ==> warning
   qqnorm(qnorMix(prob, get(n, "package:nor1mix")), main = n)
mtext("QQ-plots of Marron-Wand Densities", outer = TRUE,
      font = 2, cex = 1.6)
par(op)

## "object" overview:
cbind(sapply(nms, function(n) { o <- get(n)
      sprintf("%-18s: K =%2d; rng = [%3.1f, %2.1f]",
              attr(o, "name"), nrow(o),
              min(o["mu"] - 3*o["sigma"]),
              max(o["mu"] + 3*o["sigma"]) )
             }))


## Note that Marron-Wand (1992), p.720  give #2 as
MW.nm2
## the parameters of which at first look quite different from
MW.nm2.old
## which has been the definition in the above "Source" Matlab code.
## It's easy to see that mu_{nm2} = -.3 + 1.2   * mu_{paper},
## and correspondigly,   s2_{nm2} =       1.2^2 * s2_{paper}
## such that they are "identical" apart from scale and location:
op. <- par(mfrow=2:1, mgp= c(1.2,0.5,0), tcl= -0.2, mar=.1+c(2,2,2,1))
plot(MW.nm2)
plot(MW.nm2.old)
par(op.)
```

---

norMix *Mixtures of Univariate Normal Distributions*

Description

Objects of class norMix represent finite mixtures of (univariate) normal (aka Gaussian) distributions. Methods for construction, printing, plotting, and basic computations are provided.

Usage

```
norMix(mu, sig2 = rep(1,m), sigma = rep(1,m),
```

```
       w = NULL, name = NULL, long.name = FALSE)

is.norMix(obj)
 m.norMix(obj)
var.norMix(x, ...)
## S3 method for class 'norMix'
mean(x, ...)
## S3 method for class 'norMix'
print(x, ...)
## S3 method for class 'norMix'
x[i,j, drop=TRUE]
```

Arguments

mu	numeric vector of length K, say, specifying the means μ of the K normal components.
sig2	**deprecated!** numeric vector of length K, specifying the variances σ^2 of the K normal components. Do specify sigma instead!
sigma	numeric vector of length K, specifying the standard deviations σ of the K normal components.
w	numeric vector of length K, specifying the mixture proportions π_j of the normal components, $j = 1, \ldots, K$. Defaults to equal proportions
name	optional name tag of the result (used for printing).
long.name	logical indicating if the name attribute should use punctuation and hence be slightly larger than by default.
obj,x	an object of class norMix.
i,j,drop	for indexing, see the generic [extractor function.
...	further arguments passed to methods.

Details

The (one dimensional) normal mixtures, R objects of class "norMix", are constructed by norMix and tested for by is.norMix. m.norMix() returns the number of mixture components; the mean() method for class "norMix" returns the (theoretical / true) mean $E[X]$ and var.norMix() the true variance $E[(X - E[X])^2]$ where $X \sim \langle norm.mixt \rangle$.

The subsetting aka "extract" method (x[i,j]; for generic []—when called as x[i,]—will typically return a "norMix" object unless matrix indexing selects only one row in which case x[i, , drop=FALSE] will return the normal mixture (of one component only).

For further methods (density, random number generation, fitting, ...), see below.

Value

norMix returns objects of class "norMix" which are currently implemented as 3-column matrix with column names mu, sigma, and w, and further attributes. The user should rarely need to access the underlying structure directly.

Note

For *estimation* of the parameters of such a normal mixture, we provide a smart parametrization and an efficient implementation of the direct MLE or also the EM algorithm, see `norMixMLE()` which includes `norMixEM()`.

Author(s)

Martin Maechler

See Also

`dnorMix` for the density, `pnorMix` for the cumulative distribution and the quantile function (`qnorMix`), and `rnorMix` for random numbers and `plot.norMix`, the plot method.

`MarronWand` has the Marron-Wand densities as normal mixtures.

`norMixMLE()` and `norMixEM()` provide *fitting* of univariate normal mixtures to data.

Examples

```
ex <- norMix(mu = c(1,2,5))# defaults: sigma = 1, equal proportions ('w')
ex
plot(ex, p.comp = TRUE)# looks like a mixture of only 2; 'p.comp' plots components

## The 2nd Marron-Wand example, see also  ?MW.nm2
ex2 <- norMix(name = "#2 Skewed",
                mu = c(0, .5, 13/12),
     sigma = c(1, 2/3, 5/9),
 w = c(.2, .2, .6))

m.norMix  (ex2)
mean      (ex2)
var.norMix(ex2)
(e23 <- ex2[2:3,]) # (with re-normalized weights)
stopifnot(is.norMix(e23),
          all.equal(var.norMix(ex2),     719/1080, tol=1e-14),
          all.equal(var.norMix(ex ),      35/9,    tol=1e-14),
          all.equal(var.norMix(ex[2:3,]), 13/4,    tol=1e-14),
          all.equal(var.norMix(e23), 53^2/(12^3*4),tol=1e-14)
)

plot(ex2, log = "y")# maybe "revealing"
```

`norMix2call` *Transform "norMix" object into Call, Expression or Function*

Description

E.g., for taking symbolic derivatives, it may be useful to get an R `call`, `expression`, or `function` in / of x from a specific `"norMix"` object.

Usage

```
norMix2call(obj, oneArg = TRUE)
## S3 method for class 'norMix'
as.expression(x, oneArg = TRUE, ...)
## S3 method for class 'norMix'
  as.function(x, oneArg = TRUE, envir = parent.frame(), ...)
```

Arguments

obj, x	an R object of class "norMix".
oneArg	logical specifying if expressions of the form dnorm((x - mu)/sig) should be used, i.e. **one Arg**ument only, instead of dnorm(x, mu, sig).
envir	an environment; often the default is perfect.
...	potentially further arguments (not used in any examples yet).

Value

according to the function used, an R 'language' object, i.e., a call, expression, or function, respectively.

Author(s)

Martin Maechler

See Also

norMix. Note that deriv() currently only works correctly in case of the default oneArg = TRUE.

Examples

```
(cMW2 <- norMix2call(MW.nm2))
deriv(cMW2, "x")

(fMW1 <- as.function  (MW.nm1))
(eMW3 <- as.expression(MW.nm3))
stopifnot(is.call      (cMW2),  is.call(norMix2call(MW.nm2, FALSE)),
          is.function  (fMW1),  is.function  (as.function  (MW.nm4)),
          is.expression(eMW3),  is.expression(as.expression(MW.nm5))
)
```

norMixFit *EM and MLE Estimation of Univariate Normal Mixtures*

Description

These functions estimate the parameters of a univariate (finite) normal mixture using the EM algorithm or Likelihood Maximimization via optim(.., method = "BFGS").

Usage

```
norMixEM(x, m, name = NULL, sd.min = 1e-07* diff(range(x))/m,
         trafo = c("clr1", "logit"),
         maxiter = 100, tol = sqrt(.Machine$double.eps), trace = 1)

norMixMLE(x, m, name = NULL,
          trafo = c("clr1", "logit"),
          maxiter = 100, tol = sqrt(.Machine$double.eps), trace = 2)
```

Arguments

x	numeric: the data for which the parameters are to be estimated.
m	integer or factor: If m has length 1 it specifies the number of mixture components, otherwise it is taken to be a vector of initial cluster assignments, see details below.
name	character, passed to norMix. The default, NULL, uses match.call().
sd.min	number: the minimal value that the normal components' standard deviations (sd) are allowed to take. A warning is printed if some of the final sd's are this boundary.
trafo	character string specifying the transformation of the component weight w m-vector (mathematical notation in norMix: $\pi_j, j = 1, \ldots, m$) to an $(m-1)$-dimensional unconstrained parameter vector in our parametrization. See nM2par for details.
maxiter	integer: maximum number of EM iterations.
tol	numeric: EM iterations stop if relative changes of the log-likelihood are smaller than tol.
trace	integer (or logical) specifying if the iterations should be traced and how much output should be produced. The default, 1 prints a final one line summary, where trace = 2 produces one line of output per iteration.

Details

Estimation of univariate mixtures can be very sensitive to initialization. By default, norMixEM and norMixLME cut the data into m groups of approximately equal size. See examples below for other initialization possibilities.

The EM algorithm consists in repeated application of E- and M- steps until convergence. Mainly for didactical reasons, we also provide the functions `estep.nm`, `mstep.nm`, and `emstep.nm`.

The MLE, Maximum Likelihood Estimator, maximizes the likelihood using `optim`, using the same advantageous parametrization as `llnorMix`.

Value

An object of class `norMix`.

Author(s)

EM: Friedrich Leisch, originally; Martin Maechler vectorized it in m, added `trace` etc.

MLE: M.Maechler

Examples

```
## use (mu, sigma)
ex  <- norMix(mu = c(-1,2,5), sigma = c(1, 1/sqrt(2), sqrt(3)))
tools::assertWarning(verbose=TRUE,
           ## *deprecated* (using 'sig2' will *NOT* work in future!)
           ex. <- norMix(mu = c(-1,2,5), sig2 = c(1, 0.5, 3))
       )
stopifnot(all.equal(ex, ex.))
plot(ex, col="gray", p.norm=FALSE)

x <- rnorMix(100, ex)
lines(density(x))
rug(x)

## EM estimation may fail depending on random sample
ex1 <- norMixEM(x, 3, trace=2) #-> warning (sometimes)
ex1
plot(ex1)

## initialization by cut() into intervals of equal length:
ex2 <- norMixEM(x, cut(x, 3))
ex2

## initialization by kmeans():
k3 <- kmeans(x, 3)$cluster
ex3 <- norMixEM(x, k3)
ex3

## Now, MLE instead of EM:
exM <- norMixMLE(x, k3, tol = 1e-12, trace=4)
exM

## real data
data(faithful)
plot(density(faithful$waiting, bw = "SJ"), ylim=c(0,0.044))
rug(faithful$waiting)
```

```
(nmF <- norMixEM(faithful$waiting, 2))
lines(nmF, col=2)
## are three components better?
nmF3 <- norMixEM(faithful$waiting, 3, maxiter = 200)
lines(nmF3, col="forestgreen")
```

---

plot.norMix *Plotting Methods for 'norMix' Objects*

Description

The plot and lines methods for norMix objects draw the normal mixture density, optionally additonally with a fitted normal density.

Usage

```
## S3 method for class 'norMix'
plot(x, type = "l", n = 511, xout = NULL, xlim = NULL, ylim,
      xlab = "x", ylab = "f(x)", main = attr(x, "name"), lwd = 1.4,
      p.norm = !p.comp, p.h0 = TRUE, p.comp = FALSE,
      parNorm = list(col = 2, lty = 2, lwd = 0.4),
      parH0 = list(col = 3, lty = 3, lwd = 0.4),
      parComp = list(col= "blue3", lty = 3, lwd = 0.4), ...)

## S3 method for class 'norMix'
lines(x, type = "l", n = 511, xout = NULL,
      lwd = 1.4, p.norm = FALSE, parNorm = list(col = 2, lty = 2, lwd = 0.4),
      ...)
```

Arguments

x	object of class norMix.
type	character denoting type of plot, see, e.g. lines.
n	number of points to generate if xout is unspecified.
xout	numeric or NULL giving the abscissae at which to draw the density.
xlim	range of x values to use; particularly important if xout is not specified where xlim is passed to dnorMix and gets a smart default if unspecified.
ylim	range of y values to use; by default, if not specified (or containing NA), a smart default is used.
xlab,ylab	labels for the x and y axis with defaults.
main	main title of plot, defaulting to the norMix name.
lwd	line width for plotting with a non-standard default.
p.norm	logical indicating if the normal density with the same mean and variance should be drawn as well.

p.h0	logical indicating if the line $y = 0$ should be drawn.
p.comp	logical indicating if the Gaussian components should also be drawn individually.
parNorm	graphical parameters for drawing the normal density if p.norm is true.
parH0	graphical parameters for drawing the line $y = 0$ if p.h0 is true.
parComp	graphical parameters for drawing the single components if p.comp is true.
...	further arguments passed to and from methods.

Author(s)

Martin Maechler

See Also

norMix for the construction and further methods, particularly dnorMix which is used here.

Examples

```
plot(norMix(m=c(0,3), sigma = c(2,1))) # -> var = c(2^2, 1) = c(4, 1)

plot(MW.nm4, p.norm=FALSE, p.comp = TRUE)
plot(MW.nm4, p.norm=FALSE, p.comp = TRUE, ylim = c(0, 2))# now works
stopifnot(all.equal(c(0,2), par("yaxp")[1:2], tol= 1e-15))

## Further examples in  ?norMix and  ?rnorMix
```

---

pnorMix *Normal Mixture Cumulative Distribution and Quantiles*

Description

Compute cumulative probabilities or quantiles (the inverse) for a normal mixture specified as norMix object.

Usage

```
pnorMix(q, obj, lower.tail = TRUE, log.p = FALSE)

qnorMix(p, obj, lower.tail = TRUE, log.p = FALSE,
        tol = .Machine$double.eps^0.25, maxiter = 1000, traceRootsearch = 0,
        method = c("interpQspline", "interpspline", "eachRoot", "root2"),
        l.interp = pmax(1, pmin(20, 1000 / m)), n.mu.interp = 100)
```

Arguments

obj	an object of class norMix.
p	numeric vector of probabilities. Note that for all methods but "eachRoot", qnorMix(p, *) works with the full vector p, typically using (inverse) interpolation approaches; consequently the result is very slightly dependent on p as a whole.
q	numeric vector of quantiles
.	
lower.tail	logical; if TRUE (default), probabilities are $P[X \le x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).
tol, maxiter	tolerance and maximal number of iterations for the root search algorithm, see method below and uniroot.
traceRootsearch	logical or integer in $\{0, 1, 2, 3\}$, determining the amount of information printed during root search.
method	a string specifying which algorithm is used for the "root search". Originally, the only method was a variation of "eachRoot", which is the default now when only very few quantiles are sought. For large m.norMix(), the default is set to "root2", currently.
l.interp	positive integer for method = "interQpspline" or "interpspline", determining the number of values in each "mu-interval".
n.mu.interp	positive integer for method = "interQpspline" or "interpspline", determining the (maximal) number of mu-values to be used as knots for inverse interpolation.

Details

Whereas the distribution function pnorMix is the trivial sum of weighted normal probabilities (pnorm), its inverse, qnorMix is computed numerically: For each p we search for q such that pnorMix(obj, q) == p, i.e., $f(q) = 0$ for f(q) := pnorMix(obj, q) - p. This is a root finding problem which can be solved by uniroot(f, lower,upper,*). If length(p) <= 2 or method = "eachRoot", this happens one for one for the *sorted* p's. Otherwise, we start by doing this for the outermost non-trivial ($0 < p < 1$) values of p.

For method = "interQpspline" or "interpspline", we now compute p. <- pnorMix(q., obj) for values q. which are a grid of length l.interp in each interval $[q_j, q_{j+1}]$, where q_j are the "X-extremes" plus (a sub sequence of length n.mu.interp of) the ordered mu[j]'s. Then, we use *montone* inverse interpolation (splinefun(q., p., method="monoH.FC")) plus a few (maximally maxiter, typically one!) Newton steps. The default, "interQpspline", additionally logit-transforms the p. values to make the interpolation more linear. This method is faster, particularly for large length(p).

Value

a numeric vector of the same length as p or q, respectively.

Author(s)

Very first version (for length-1 p,q) by Erik Jørgensen <Erik.Jorgensen@agrsci.dk>.

See Also

dnorMix for the density function.

Examples

```
MW.nm3 # the "strange skew" one
plot(MW.nm3)
## now the cumlative :
x <- seq(-4,4, length.out = 1001)
plot(x, pnorMix(x, MW.nm3), type="l", col=2)
## and some of its inverse :
pp <- seq(.1, .9, by=.1)
plot(qnorMix(pp, MW.nm3), pp)

## The "true" median of a normal mixture:
median.norMix <- function(x) qnorMix(1/2, x)
median.norMix(MW.nm3) ## -2.32
```

---

r.norMix *Ratio of Normal Mixture to Corresponding Normal*

Description

Compute $r(x) = f(x)/f0(x)$ where $f()$ is a normal mixture density and $f0$ the normal density with the same mean and variance as f.

Usage

```
r.norMix(obj, x = NULL, xlim = NULL, n = 511, xy.return = TRUE)
```

Arguments

obj	an object of class norMix.
x	numeric vector with abscissa values where to evaluate the density. Default is constructed from n (and xlim if specified).
xlim	range of abscissa values, used if x == NULL. By default, xlim taken as mean plus/minus 3 standard deviations of the normal mixture.
n	number of abscissa values to generate if x is not specified.
xy.return	logical indicating if the result should be a list or just a numeric vector, see below.

Value

It depends on `xy.return`. If it's false, a numeric vector of the same length as `x`, if true (as per default), a list that can be plotted, with components

`x`	abscissa values corresponding to argument `x`.
`y`	corresponding values $r(x)$.
`f0`	values of the moment matching normal density $f0(x)$.

Note

The ratio function is used in certain semi-parametric density estimation methods (and theory).

Examples

```
d3 <- norMix(m = 5*(0:2), w = c(0.6, 0.3, 0.1))
plot(d3)
rd3 <- r.norMix(d3)
str(rd3)
stopifnot(rd3 $ y  == r.norMix(d3, xy.ret = FALSE))
par(new = TRUE)
plot(rd3, type = "l", col = 3, axes = FALSE, xlab = "", ylab="")
axis(4, col.axis=3)
```

---

rnorMix *Generate 'Normal Mixture' Distributed Random Numbers*

Description

Generate `n` random numbers, distributed according to a normal mixture.

Usage

```
rnorMix(n, obj)
```

Arguments

`n`	the number of random numbers desired.
`obj`	an object of class `norMix`.

Details

For a mixture of m, i.e., `m.norMix(obj)`, components, generate the number in each component as multinomial, and then use `rnorm` for each.

Note that the these integer (multinomial) numbers are generated via `sample()`, which is by `.Random.seed`, notably from `RNGkind(sample.kind = ..)` which changed with R version 3.6.0.

Value

numeric vector of length n.

See Also

dnorMix for the density, and norMix for the construction and further methods.

Examples

```
x <- rnorMix(5000, MW.nm10)
hist(x)# you don't see the claw
plot(density(x), ylim = c(0,0.6),
     main = "Estim. and true 'MW.nm10' density")
lines(MW.nm10, col = "orange")
```

---

sort.norMix *Sort Method for "norMix" Objects*

Description

Sorting a "norMix" object (see norMix), sorts along the mu values; i.e., for the default decreasing = FALSE the resulting x[,"mu"] are sorted from left to right.

Usage

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

Arguments

x	an object of class "norMix".
decreasing	logicial indicating if sorting should be up or down.
...	further arguments passed to sort(x[,"mu"],*).

Value

a "norMix" object like x.

Examples

```
sort(MW.nm9)
stopifnot(identical(MW.nm2, sort(MW.nm2)))
```

Index

∗ **cluster**
clus2norMix, 3
∗ **datasets**
MarronWand, 8
∗ **distribution**
dnorMix, 4
MarronWand, 8
norMix, 10
plot.norMix, 16
pnorMix, 17
r.norMix, 19
rnorMix, 20
∗ **hplot**
plot.norMix, 16
∗ **math**
norMix2call, 12
∗ **models**
clus2norMix, 3
llnorMix, 6
∗ **package**
nor1mix-package, 2
∗ **utilities**
sort.norMix, 21
.Random.seed, *20*
[, *11*
[.norMix *(*norMix*)*, 10

as.expression.norMix *(*norMix2call*)*, 12
as.function.norMix *(*norMix2call*)*, 12

call, *12, 13*
character, *6*, *14*
clara, *4*
class, *11*, *13*
clus2norMix, 3
cut, *14*

deriv, *13*
dnorMix, 4, *8*, *12*, *16, 17*, *19*, *21*
dnorMixL *(*dnorMix*)*, 4
do.call, *7*
dpnorMix *(*dnorMix*)*, 4

emstep.nm, *15*
emstep.nm *(*llnorMix*)*, 6
environment, *13*
estep.nm, *15*
estep.nm *(*llnorMix*)*, 6
expression, *12, 13*

factor, *4*
function, *12, 13*

is.norMix *(*norMix*)*, 10

lines, *16*
lines.norMix *(*plot.norMix*)*, 16
list, *7*
llnorMix, 6, *15*
logical, *13*
logLik, *8*

m.norMix, *18*
m.norMix *(*norMix*)*, 10
MarronWand, *3*, 8, *12*
match.call, *14*
matrix, *6, 7*
mean.norMix *(*norMix*)*, 10
mstep.nm *(*llnorMix*)*, 6
Mvdc, *3*
MW.nm1 *(*MarronWand*)*, 8
MW.nm10 *(*MarronWand*)*, 8
MW.nm11 *(*MarronWand*)*, 8
MW.nm12 *(*MarronWand*)*, 8
MW.nm13 *(*MarronWand*)*, 8
MW.nm14 *(*MarronWand*)*, 8
MW.nm15 *(*MarronWand*)*, 8
MW.nm16 *(*MarronWand*)*, 8
MW.nm2 *(*MarronWand*)*, 8
MW.nm3 *(*MarronWand*)*, 8
MW.nm4 *(*MarronWand*)*, 8

MW.nm5 (MarronWand), 8
MW.nm6 (MarronWand), 8
MW.nm7 (MarronWand), 8
MW.nm8 (MarronWand), 8
MW.nm9 (MarronWand), 8

nM2par, *14*
nM2par (llnorMix), 6
nor1mix (nor1mix-package), 2
nor1mix-package, 2
norMix, *3–8*, 10, *12–17*, *21*
norMix2call, 12
norMixEM, *6*, *8*
norMixEM (norMixFit), 14
norMixFit, 14
norMixMLE, *3*, *8*, *12*
norMixMLE (norMixFit), 14

optim, *3*, *14*, *15*

pam, *4*
par2norMix (llnorMix), 6
plot.norMix, *5*, *12*, 16
pnorm, *18*
pnorMix, *12*, 17, *18*
print.norMix (norMix), 10

qnorMix (pnorMix), 17

r.norMix, 19
RNGkind, *20*
rnorm, *20*
rnorMix, *5*, *12*, 20

sample, *20*
sort, *21*
sort.norMix, 21
splinefun, *18*
split, *4*

uniroot, *18*

var.norMix (norMix), 10