

# Package ‘nor1mix’

January 2, 2012

**Title** Normal (1-d) Mixture Models (S3 Classes and Methods)

**Version** 1.1-3

**Date** 2011-04-14

**Author** Martin Mächler

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

**Description** Onedimensional Normal Mixture Models Classes, for, e.g., density estimation or clustering algorithms research and teaching; providing the widely used Marron-Wand densities, see ?MarronWand.

**Suggests** cluster

**License** GPL (>= 2)

**Depends** R (>= 2.8.0), stats, graphics

**Encoding** latin1

**Repository** CRAN

**Date/Publication** 2011-04-14 16:45:49

## R topics documented:

clus2norMix . . . . .	2
dnorMix . . . . .	3
llnorMix . . . . .	4
MarronWand . . . . .	6
norMix . . . . .	7
plot.norMix . . . . .	9
pnorMix . . . . .	10
r.norMix . . . . .	12
rnorMix . . . . .	13
sort.norMix . . . . .	14

<b>Index</b>	<b>15</b>
--------------	-----------

---

`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**

<code>gr</code>	a grouping/clustering vector with values in $\{1, \dots, K\}$ ; possibly a <a href="#">factor</a> .
<code>x</code>	numeric vector of (original) data (of the same length as <code>gr</code> ).
<code>name</code>	name for <a href="#">norMix()</a> 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	<i>Normal Mixture Density</i>
---------	-------------------------------

---

**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**

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

<code>x</code>	the abscissa values.
<code>y</code>	the density values $f(x)$ as for <code>dnorMix()</code> .

`dpnorMix()` returns a list with components

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

**See Also**

[rnormMix](#) for random number generation, and [normMix](#) for the construction and further methods, particularly `plot.normMix` 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(-4,5, length=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$d, dnorMix(x, MW.nm7), tol=1e-12),
          all.equal(dp$p, pnorMix(x, MW.nm7), tol=1e-12))
```

llnorMix

*Likelihood and Parametrization of 1D Normal Mixtures***Description**

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

llnorMix(p, \*) computes the log likelihood, \ where as

obj <- par2norMix(p) and

p <- nM2par(obj)

map to and from [norMix](#) objects obj and parameter vector p in our parametrization.

**Usage**

```
llnorMix(p, x, m = (length(p) + 1)/3)
```

```
par2norMix(p, name = sprintf("{from %s}", deparse(substitute(p))[1]))
```

```
nM2par(obj)
```

**Arguments**

p	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.
name	(for par2norMix(:)) a name for the "norMix" object that is returned.
obj	a "norMix" object, see <a href="#">norMix</a> .

**Details**

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

For a  $k$ -component mixture, we map to and from a parameter vector  $\theta$  ( $=$  p as  $\mathbf{R}$ -vector) of length  $3k - 1$ . For mixture density

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

we logit-transform the  $\pi_j$  (for  $j \geq 2$ ) and log-transform the  $\sigma_j$ , such that  $\theta$  is partitioned into

$\rho[1:(k-1)]$ :  $\rho[j] = \text{logit}(\pi_{j+1})$  and  $\pi_1$  is given implicitly as  $\pi_1 = 1 - \sum_{j=2}^k \pi_j$ .

$\rho[k:(2k-1)]$ :  $\rho[k-1+j] = \mu_j$ , for  $j=1:k$ .

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

## Value

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

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

`nM2par()` returns the parameter vector  $\theta$  of length  $3k - 1$ .

## 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))`.

## 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 \code{obj} (see below)

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

## Consistency check :
stopifnot(all.equal(pp, nM2par(par2norMix(pp)), tol= 1e-15),
          all.equal(obj, par2norMix(nM2par(obj)),
                    check.attributes=FALSE, tol=1e-15),
          ## Direct computation of log-likelihood:
          all.equal(sum(dnorMix(x, obj, log=TRUE)),
                    llnorMix(pp, x), tol= 1e-15) )
```

## 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 are translated from Steve Marron's Matlab code at <http://www.stat.unc.edu/postscript/papers/marron/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 Statistics* **20**, 712–736.

For number 16,  
Janssen, Marron, Verb..., Sarle (1995) ....

**Examples**

```

MW.nm10
plot(MW.nm14)

## These are defined as norMix() calls in ../R/zMarrWand-dens.R
nms <- ls(pat="^MW.nm", "package:nor1mix")
nms <- nms[order(as.numeric(substring(nms,6)))]
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])
  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*sqrt(o[, "sig2"])),
          max(o[, "mu"] + 3*sqrt(o[, "sig2"])) )
}))

## 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 correspondingly,  $s2_{\{nm2\}} = 1.2^2 * s2_{\{paper\}}$ 
## 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.)

```

## 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), 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, ...)
```

## Arguments

<code>mu</code>	numeric vector of length $K$ , say, specifying the means $\mu$ of the $K$ normal components.
<code>sig2</code>	numeric vector of length $K$ , specifying the variances $\sigma^2$ of the $K$ normal components.
<code>w</code>	numeric vector of length $K$ , specifying the mixture proportions $\pi_j$ of the normal components, $j = 1, \dots, K$ . Defaults to equal proportions
<code>name</code>	optional name tag of the result (used for printing).
<code>long.name</code>	logical indicating if the name attribute should use punctuation and hence be slightly larger than by default.
<code>obj, x</code>	an object of class <code>norMix</code> .
<code>...</code>	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 `mu` vector of means and `var.norMix()` (not a method, call the function explicitly!) the `sig2` vector of variances.

For further methods see below.

## Value

`norMix` returns objects of class "norMix" which are currently implemented as 3-column matrix with column names `mu`, `sig2`, and `w`, and further attributes. The user should rarely need to access the underlying structure directly.

## Note

For *estimation* of the parameters of a normal mixture distribution, I recommend using other R packages, such as **flexmix**.

**Author(s)**

Martin Maechler

**See Also**

[dnorMix](#) for the density, [pnorMix](#) for the cumulative distribution and the quantile function ([qnorMix](#)), and [rnormMix](#) for random numbers and [plot.norMix](#), the plot method.

[MarronWand](#) has the Marron-Wand densities as normal mixtures.

**Examples**

```
ex <- norMix(mu = c(1,2,5))# s^2 = 1, equal proportions
ex
plot(ex)# looks like a mixture of only 2

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

plot.norMix

*Plotting Methods for 'norMix' Objects***Description**

The plot and lines methods for [norMix](#) objects draw the normal mixture density, optionally additionally with a fitted normal density.

**Usage**

```
## S3 method for class 'norMix'
plot(x, type = "l", n = 511, xout = NULL, xlim = NULL,
     xlab = "x", ylab = "f(x)", main = attr(x, "name"), lwd = 1.4,
     p.norm = TRUE, 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 <code>norMix</code> .
type	character denoting type of plot, see, e.g. <a href="#">lines</a> .
n	number of points to generate if <code>xout</code> is unspecified.
xout	numeric or <code>NULL</code> 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 <a href="#">dnorMix</a> and gets a smart default if unspecified.
xlab,ylab	labels for the x and y axis with defaults.
main	main title of plot, defaulting to the <a href="#">norMix</a> 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),s=c(4,1)))

plot(MW.nm4, p.norm=FALSE, p.comp = TRUE)
## Further examples in ?norMix and ?rnormMix
```

---

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", "interspline", "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 $P[X \leq x]$ , 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 <a href="#">uniroot</a> .
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 <a href="#">m.norMix()</a> , 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  $\text{pnorMix}(\text{obj}, q) == p$ , i.e.,  $f(q) = 0$  for  $f(q) := \text{pnorMix}(\text{obj}, q) - p$ . This is a root finding problem which can be solved by [uniroot](#)(f, lower, upper, \*). If  $\text{length}(p) \leq 2$  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  $p. \leftarrow \text{pnorMix}(q., \text{obj})$  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  $\text{length}(p)$ .

**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 cumulative :
x <- seq(-4,4, length=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  $r(x) = f(x)/f_0(x)$  where  $f()$  is a normal mixture density and  $f_0$  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 <code>norMix</code> .
x	numeric vector with abscissa values where to evaluate the density. Default is constructed from <code>n</code> (and <code>xlim</code> if specified).
xlim	range of abscissa values, used if <code>x == NULL</code> . By default, <code>xlim</code> taken as mean plus/minus 3 standard deviations of the normal mixture.
n	number of abscissa values to generate if <code>x</code> 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 <code>x</code> .
y	corresponding values $r(x)$ .
f0	values of the moment matching normal density $f_0(x)$ .

**Note**

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

**Examples**

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

---

rnormMix

*Generate 'Normal Mixture' Distributed Random Numbers*


---

**Description**

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

**Usage**

```
rnormMix(n, obj)
```

**Arguments**

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

**Details**

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

**Value**

numeric vector of length  $n$ .

**See Also**

[dnormMix](#) for the density, and [rnormMix](#) for the construction and further methods.

**Examples**

```
x <- rnormMix(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**

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

**Value**

a "norMix" object like `x`.

**Examples**

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

# Index

- \*Topic **cluster**
  - clus2norMix, 2
- \*Topic **datasets**
  - MarronWand, 6
- \*Topic **distribution**
  - dnorMix, 3
  - MarronWand, 6
  - norMix, 7
  - plot.norMix, 9
  - pnorMix, 10
  - r.norMix, 12
  - rnorMix, 13
- \*Topic **hplot**
  - plot.norMix, 9
- \*Topic **models**
  - clus2norMix, 2
  - llnorMix, 4
- \*Topic **utilities**
  - sort.norMix, 14
  
- clara, 2
- class, 8
- clus2norMix, 2
  
- dnorMix, 3, 5, 9, 10, 12, 13
- dnorMixL (dnorMix), 3
- dpnorMix (dnorMix), 3
  
- factor, 2
  
- is.norMix (norMix), 7
  
- lines, 9
- lines.norMix (plot.norMix), 9
- llnorMix, 4
- logLik, 5
  
- m.norMix, 11
- m.norMix (norMix), 7
- MarronWand, 6, 9
- mean.norMix (norMix), 7
  
- MW.nm1 (MarronWand), 6
- MW.nm10 (MarronWand), 6
- MW.nm11 (MarronWand), 6
- MW.nm12 (MarronWand), 6
- MW.nm13 (MarronWand), 6
- MW.nm14 (MarronWand), 6
- MW.nm15 (MarronWand), 6
- MW.nm16 (MarronWand), 6
- MW.nm2 (MarronWand), 6
- MW.nm3 (MarronWand), 6
- MW.nm4 (MarronWand), 6
- MW.nm5 (MarronWand), 6
- MW.nm6 (MarronWand), 6
- MW.nm7 (MarronWand), 6
- MW.nm8 (MarronWand), 6
- MW.nm9 (MarronWand), 6
  
- nM2par (llnorMix), 4
- norMix, 2–5, 7, 9, 10, 13, 14
  
- pam, 2
- par2norMix (llnorMix), 4
- plot.norMix, 3, 9, 9
- pnorm, 11
- pnorMix, 9, 10, 11
- print.norMix (norMix), 7
  
- qnorMix (pnorMix), 10
  
- r.norMix, 12
- rnorm, 13
- rnorMix, 3, 9, 13
  
- sort, 14
- sort.norMix, 14
- splinefun, 11
- split, 2
  
- uniroot, 11
  
- var.norMix (norMix), 7