Package ‘norMmix’

September 9, 2024

Version 0.2-0

Title Direct MLE for Multivariate Normal Mixture Distributions

Description Multivariate Normal (i.e. Gaussian) Mixture Models (S3) Classes.
Fitting models to data using 'MLE' (maximum likelihood estimation) for
multivariate normal mixtures via smart parametrization using the 'LDL'
(Cholesky) decomposition, see McLachlan and Peel (2000, ISBN:9780471006268),
Celeux and Govaert (1995) <doi:10.1016/0031-3203(94)00125-6>.

Imports cluster, MASS, mvtnorm, mclust, sfsmisc

Suggests nor1mix, Matrix, testthat (>= 2.1.0), knitr, rmarkdown

License GPL (>= 3)

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation no

Author Nicolas Trutmann [aut, cre],
Martin Maechler [aut, ths] (<`https://orcid.org/0000-0002-8685-9910`>,
based on 'nor1mix')

Maintainer Nicolas Trutmann <`nicolas.trutmann@gmx.ch`>

Repository CRAN

Date/Publication 2024-09-09 06:40:03 UTC

Contents

`clr1` *Centered Log Ratio Transformation and Inverse*

Description

The centered log ratio transformation is Maechler's solution to allowing unconstrained mixture weights optimization.

It has been inspired by Aitchison's **c**entered **l**og **r**atio, see also CRAN package **compositions**' `clr()`, and typically other references on modelling proportions.

Usage

```
clr1(w)
clr1inv(lp)
```

Arguments

`w`	numeric vector of length k, say, of mixture weights, i.e., non-negative and summing to one.
`lp`	numeric vector of length $k-1$ clr-transformed weights.

Details

Aitchison's clr transformation is slightly different, as it does *not* drop one coordinate, as we do. Hence the extra '1' in the name of our version.

Value

a numeric vector of length $k-1$ or k, see above.

Author(s)

Martin Maechler

References

Aitchison, J., 1986. *The Statistical Analysis of Compositional Data* Monographs on Statistics and Applied Probability. Chapman & Hall Ltd., London (UK).

More in the CRAN package **compositions** vignette '`UsingCompositions.pdf`'

See Also

The first implementation of these was in **nor1mix**, June 2019, in its `par2norMix()` and `nM2par()` functions.

Examples

```
## Apart from error checking and very large number cases, the R implementation is simply
..clr1 <- function (w) {
  ln <- log(w)
  ln[-1L] - mean(ln)
}

## and its inverse
..clr1inv <- function(lp) {
  p1 <- exp(c(-sum(lp), lp))
  p1/sum(p1)
}

lp <- clr1( (1:3)/6 )
clr1inv(lp)
stopifnot(all.equal(clr1inv(lp), (1:3)/6))

for(n in 1:100) {
   k <- 2 + rpois(1, 3) # #{components}
   lp <- rnorm(k-1) # arbitrary unconstrained
   ## clr1() and clr1inv() are inverses :
   stopifnot(all.equal(lp, clr1(clr1inv(lp))))
}

wM <- clr1inv(c(720,720,720))
w2 <- clr1inv(c(720,718,717))
stopifnot(is.finite(wM), all.equal(wM, c(0, 1/3, 1/3, 1/3))
        , is.finite(w2), all.equal(w2, c(0, 0.84379473, 0.1141952, 0.042010066))
         )
```

`dfnMm` *Number of Free Parameters of Multivariate Normal Mixture Models*

Description

`npar()` returns an integer (vector, if `p` or `k` is) with the number of free parameters of the corresponding model, which is also the `length(.)` of the parameter vector in our parametrization, see `nMm2par()`.

Usage

```
dfnMm(k, p, model = c("EII","VII","EEI","VEI","EVI",
                 "VVI","EEE","VEE","EVV","VVV"))
```

Arguments

`k`	number of mixture components
`p`	dimension of data space, i.e., number of variables (aka “features”).
`model`	a `character` string. One of the 10 models above, see also ‘Description’.

Value

integer. degrees of freedom of a model with specified dimensions, components and model type.

Examples

```
(m <- eval(formals(dfnMm)$model)) # list of 10 models w/ differing Sigma
# A nice table for a given 'p'  and all models, all k in 1:8
sapply(m, dfnMm, k=setNames(,1:8), p = 20)
```

`dnorMmix` *Density from Multivariate Normal Mixture Distribution*

Description

Calculates the probability density function of the multivariate normal distribution.

Usage

```
dnorMmix(x, nMm)
```

Arguments

`x`	a vector or matrix of multivariate observations
`nMm`	a "`norMmix`" object

Value

Returns the density of `nMm` at point `x`. Iterates over components of the mixture and returns weighted sum of `dmvnorm`.

Author(s)

Nicolas Trutmann

See Also

`rnorMmix`

ellipsePts — *Compute Points on Bivariate Gaussian Confidence Ellipse*

Description

From 2-dimensional mean vector `mu`= μ and 2x2 covariance matrix `sigma`= Σ, compute `npoints` equi-angular points on the `1-alpha`= $1 - \alpha$ confidence ellipse of bivariate Gaussian (normal) distribution $\mathcal{N}_2(\mu, \Sigma)$.

Usage

```
ellipsePts(mu, sigma, npoints, alpha = 0.05, r = sqrt(qchisq(1 - alpha, df = 2)))
```

Arguments

`mu`	mean vector (`numeric` of length 2).
`sigma`	2x2 `matrix`, the covariance matrix.
`npoints`	integer specifying the number of points to be computed.
`alpha`	confidence level such that the ellipse should contain 1-alpha of the mass.
`r`	radius of the ellipse, typically computed from `alpha`, via the default value.

Value

a numeric matrix of dimension `npoints x 2`, containing the x-y-coordinates of the ellipse points.

Note

This has been inspired by package **mixtools**'s `ellipse()` function.

Author(s)

Martin Maechler

Examples

```
xy <- ellipsePts(c(10, 100), sigma = cbind(c(4, 7), c(7, 28)),  npoints = 20)
plot(xy, type = "b", col=2, cex=2,
     main="ellipsePts(mu = (10,100), sigma, npoints = 20)")
points(10, 100, col=3, cex=3, pch=3)
text  (10, 100, col=3, expression(mu == "mu"), adj=c(-.1, -.1))

stopifnot(is.matrix(xy), dim(xy) == c(20, 2))
```

ldl *LDL' Cholesky Decomposition*

Description

Simple (but not too simple) R implementation of the (square root free) LDL' Choleksy decomposition.

Usage

```
ldl(m)
```

Arguments

`m`	positive semi-definite square matrix, say of dimension $n \times n$.

Value

a `list` with two components

`L`	a lower triangular matrix with diagonal entries 1.
`D`	numeric vector, the *diagonal* $d_{1,1}, d_{2,2}, \ldots, d_{n,n}$ of the diagonal matrix D.

See Also

`chol()` in base R, or also a "generalized LDL" decomposition, the Bunch-Kaufman, `BunchKaufman()` in ('Recommended') package **Matrix**.

Examples

```
(L <- rbind(c(1,0,0), c(3,1,0), c(-4,5,1)))
D <- c(4,1,9)
FF <- L %*% diag(D) %*% t(L)
FF
LL <- ldl(FF)
stopifnot(all.equal(L, LL$L),
          all.equal(D, LL$D))

## rank deficient :
FF0 <- L %*% diag(c(4,0,9)) %*% t(L)
((L0 <- ldl(FF0))) #  !! now fixed with the  if(Di == 0) test
## With the "trick", it works:
stopifnot(all.equal(FF0,
                    L0$L %*% diag(L0$D) %*% t(L0$L)))
## [hint: the LDL' is no longer unique when the matrix is singular]

system.time(for(i in 1:10000) ldl(FF) ) # ~ 0.2 sec

(L <- rbind(c( 1, 0, 0, 0),
            c( 3, 1, 0, 0),
            c(-4, 5, 1, 0),
            c(-2,20,-7, 1)))
D <- c(4,1, 9, 0.5)
F4 <- L %*% diag(D) %*% t(L)
F4
L4 <- ldl(F4)
stopifnot(all.equal(L, L4$L),
          all.equal(D, L4$D))

system.time(for(i in 1:10000) ldl(F4) ) # ~ 0.16 sec

## rank deficient :
F4.0 <- L %*% diag(c(4,1,9,0)) %*% t(L)
((L0 <- ldl(F4.0)))
stopifnot(all.equal(F4.0,
                    L0$L %*% diag(L0$D) %*% t(L0$L)))

F4_0 <- L %*% diag(c(4,1,0,9)) %*% t(L)
((L0 <- ldl(F4_0)))
stopifnot(all.equal(F4_0,
                    L0$L %*% diag(L0$D) %*% t(L0$L)))

## Large
mkLDL <- function(n, rF = function(n) sample.int(n), rFD = function(n) 1+ abs(rF(n))) {
    L <- diag(nrow=n)
    L[lower.tri(L)] <- rF(n*(n-1)/2)
    list(L = L, D = rFD(n))
}

(LD <- mkLDL(17))

chkLDL <- function(n, ..., verbose=FALSE, tol = 1e-14) {
    LD <- mkLDL(n, ...)
    if(verbose) cat(sprintf("n=%3d ", n))
    n <- length(D <- LD$D)
    L <- LD$L
    M <- L %*% diag(D) %*% t(L)
    r <- ldl(M)
    stopifnot(exprs = {
        all.equal(M,
                  r$L %*% diag(r$D) %*% t(r$L), tol=tol)
        all.equal(L, r$L, tol=tol)
        all.equal(D, r$D, tol=tol)
    })
    if(verbose) cat("[ok]\n")
    invisible(list(LD = LD, M = M, ldl = r))
}

(chkLDL(7))


N <- 99 ## test  N  random cases
```

```
set.seed(101)
for(i in 1:N) {
    cat(sprintf("i=%3d, ",i))
    chkLDL(rpois(1, lambda = 20), verbose=TRUE)
}



system.time(chkLDL( 500)) # 0.62

try( ## this almost never "works":
    system.time(
        chkLDL( 500, rF = rnorm, rFD = function(n) 10 + runif(n))
    ) # 0.64
)
system.time(chkLDL( 600)) # 1.09
## .. then it grows quickly for (on nb-mm4)
## for n = 1000  it typically *fails*: The matrix M  is typically very ill conditioned
## does not depend much on the RNG ?


"==> much better conditioned L and hence M : "
set.seed(120)
L <- as(Matrix::tril(toeplitz(exp(-(0:999)/50))), "matrix")
dimnames(L) <- NULL
D <- 10 + runif(nrow(L))
M <- L %*% diag(D) %*% t(L)
rcond(L) # 0.010006 !
rcond(M) # 9.4956e-5
if(FALSE) # ~ 4-5 sec
   system.time(r <- ldl(M))
```

`llmvtnorm` *Log-Likelihood of Multivariate Normal Mixture Relying on* `mvtnorm::dmvnorm`

Description

Compute the log-likelihood of a multivariate normal mixture, by calling `dmvnorm()` (from package **mvtnorm**).

Usage

```
llmvtnorm(par, x, k,
          model = c("EII", "VII", "EEI", "VEI", "EVI",
                    "VVI", "EEE", "VEE", "EVV", "VVV"))
```

Arguments

`par`	parameter vector as calculated by nMm2par
`x`	numeric data `matrix` (of dimension $n \times p$).
`k`	number of mixture components.
`model`	assumed model of the distribution

Value

returns the log-likelihood (a number) of the specified model for the data (n observations) `x`.

See Also

`dmvnorm()` from package **mvtnorm**. Our own function, returning the same: `llnorMmix()`.

Examples

```
set.seed(1); x <- rnorMmix(50, MW29)
para <- nMm2par(MW29, model=MW29$model)

llmvtnorm(para, x, 2, model=MW29$model)
# [1] -236.2295
```

`llnorMmix` *Log-likelihood of parameter vector given data*

Description

Calculates log-likelihood of a dataset, tx, given a normal mixture model as specified by a parameter vector. A parameter vector can be obtained by applying `nMm2par` to a `norMmix` object.

Usage

```
llnorMmix(par, tx, k,
          model = c("EII", "VII", "EEI", "VEI", "EVI",
                    "VVI", "EEE", "VEE", "EVV", "VVV"))
```

Arguments

`par`	parameter vector
`tx`	*Transposed* numeric data matrix, i.e. `tx := t(x)` is of dimension $p \times n$; its rows are variables and columns are observations.
`k`	number of mixture components.
`model`	assumed distribution model of normal mixture

Value

returns the log-likelihood (a number) of the specified model for the data (n observations) `x`.

See Also

Our alternative function `llmvtnorm()` (which is based on `dmvnorm()` from package **mvtnorm**).

Examples

```
set.seed(1); tx <- t(rnorMmix(50, MW29))
para <- nMm2par(MW29, model=MW29$model)

llnorMmix(para, tx, 2, model=MW29$model)
# [1] -236.2295
```

`MarronWand` *Marron-Wand-like Specific Multivariate Normal Mixture 'norMmix' Objects*

Description

Nicolas Trutmann constructed multivariate versions from most of the univariate (i.e., one-dimensional) "Marron-Wand" densities as defined in CRAN package **nor1mix**, see `MarronWand` (in that package).

Usage

```
## 2-dim examples:
MW21   # Gaussian
MW22   # Skewed
MW23   # Str Skew
MW24   # Kurtotic
MW25   # Outlier
MW26   # Bimodal
MW27   # Separated (bimodal)
MW28   # Asymmetric Bimodal
MW29   # Trimodal
MW210  # Claw
MW211  # Double Claw
MW212  # Asymmetric Claw
MW213  # Asymm. Double Claw
MW214  # Smooth   Comb
MW215  # Trimodal

## 3-dim :
MW31
MW32
MW33
MW34

## 5 - dim:
MW51    # Gaussian
```

Value

A normal mixture model. The first digit of the number in the variable name encodes the dimension of the mixture; the following digits merely enumerate models, with some correlation to the complexity of the model.

Author(s)

Martin Maechler for 1D; Nicolas Trutmann for 2-D, 3-D and 5-D.

References

Marron, S. and Wand, M. (1992) Exact Mean Integrated Squared Error; *Annals of Statistcs* **20**, 712–736; doi:10.1214/aos/1176348653.

Examples

```
MW210
plot(MW214, main = "plot( MW214 )")

plot(MW51, main = paste("plot( MW51 );  name:", attr(MW51, "name")))
```

`nMm2par` *Multivariate Normal Mixture Model to parameter for MLE*

Description

From a "`norMmix`"(-like) object, return the numeric parameter vector in our MLE parametrization.

Usage

```
  nMm2par(obj,
         model = c("EII", "VII", "EEI", "VEI", "EVI",
                   "VVI", "EEE", "VEE", "EVV", "VVV"),
         meanFUN = mean.default,
         checkX = FALSE)
```

Arguments

`obj`	a `list` containing `sig`: covariance matrix array, `mu`: mean vector matrix, `w`: = weights, `k`: = number of components, `p`: = dimension
`model`	a `character` string specifying the (Sigma) model, one of those listed above.
`meanFUN`	a `function` to compute a mean (of variances typically).
`checkX`	a boolean. check for positive definiteness of covariance matrix.

Details

This transformation forms a vector from the parameters of a normal mixture. These consist of weights, means and covariance matrices.

Covariance matrices are given as D and L from the LDLt decomposition

Value

vector containing encoded parameters of the mixture. first, the centered log ratio of the weights, then the means, and then the model specific encoding of the covariances.

See Also

the *inverse* function of `nMm2par()` is `par2nMm()`.

Examples

```
A <- MW24
nMm2par(A, model = A$model)
# [1] -0.3465736  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
# [7] -2.3025851

## All MW* models in {norMmix} pkg:
pkg <- "package:norMmix"
lMW <- mget(ls(pattern = "^MW", pkg), envir=as.environment(pkg))
lM.par <- lapply(lMW, nMm2par)
## but these *do* differ  ___ FIXME __
modMW <- vapply(lMW, `[[`, "model", FUN.VALUE = "XYZ")
cbind(modMW, lengths(lM.par),  npar = sapply(lMW, npar))[order(modMW),]
```

`nor1toMmix` *Cast nor1mix object as norMmix.*

Description

Cast nor1mix object as norMmix.

Usage

```
nor1toMmix(object)
```

Arguments

`object`	A nor1mix mixture model to be coerced to norMmix.

Details

This package was designed to extend the `nor1mix` package to the case of multivariate mixture models. Therefore we include a utility function to cast 1-dimensional mixtures as defined in **nor1mix** to `norMmix`.

Value

A `norMmix` object if the appropriate S3method has been implemented.

`norMmix` *Constructor for Multivariate Normal Mixture Objects*

Description

`norMmix` creates a multivariate normal (aka Gaussian) mixture object, conceptually a mixture of k multivariate (p-dimensional) Gaussians $\mathcal{N}(\mu_j, \Sigma_j)$, for $j = 1, \dots, k$.

Usage

```
norMmix(mu, Sigma = NULL, weight = rep(1/k, k), name = NULL,
        model = c("EII", "VII", "EEI", "VEI", "EVI",
                  "VVI", "EEE", "VEE", "EVV", "VVV"))
```

Arguments

`mu`	matrix of means, or a vector in which case $k = 1$ is assumed. Otherwise use `as.matrix(mu)`.
`Sigma`	NULL, number, numeric, vector (length = k), matrix (dim = p x k), or array (p x p x k). See details.
`weight`	weights of mixture model components
`name`	gives the option of naming mixture
`model`	see ‘Details’

Details

`model` must be specified by one of the (currently 10) `character` strings shown in the default. (In a future version, `model` may become *optional*).

`norMmix` as a few nifty ways of constructing simpler matrices from smaller givens. This happens according to the dimension of the given value for the Sigma argument:

0. for a single value `d` or `NULL`, `norMmix()` assumes all covariance matrices to be diagonal with entries `d` or `1`, respectively.

1. for a vector `v`, `norMmix` assumes all matrices to be diagonal with the i-th matrix having diagonal entries `v[i]`.

2. for a matrix `m`, `norMmix` assumes all matrices to be diagonal with diagonal vector `m[,i]`, i.e., it goes by columns.

3. an array is assumed to be the covariance matrices, given explicitly.

FIXME ... give "all" the details ... (from Bachelor’s thesis ???)

Value

currently, a `list` of class `"norMmix"`, with a `name` attribute and components

`model`	three-letter `character` string, specifying the Sigma-parametrization
`mu`	(p x k) matrix of component means `mu[,j]`, $j = 1, \ldots, k$.
`Sigma`	(p x p x k) array of component Covariance matrices `Sigma[,,j]`.
`weight`	p-vector of mixture probability weights; non-negative, summing to one: `sum(weight) == 1`.
`k`	integer, the number of components
`dim`	integer, the dimension p.

Author(s)

Nicolas Trutmann

References

__ TODO __

See Also

`norMmixMLE()` to fit such mixture models to data (an $n \times p$ matrix).

"Marron-Wand"-like examples (for testing, etc), such as `MW21`.

Examples

```
## Some of the "MW" objects : % --> ../R/zmarrwandnMm.R

# very simple 2d:
M21 <- norMmix(mu = cbind(c(0,0)), # 2 x 1 ==> k=2, p=1
               Sigma = 1, model = "EII")
stopifnot(identical(M21, # even simpler, Sigma = default :
                    norMmix(mu = cbind(c(0,0)), model = "EII")))

m2.2 <- norMmix(mu = cbind(c(0, 0), c(5, 0)), Sigma = c(1, 10),
                weight = c(7,1)/8, model = "VEI")

m22 <- norMmix(
    name = "one component rotated",
    mu = cbind( c(0,0) ),
    Sigma = array(c(55,9, 9,3), dim = c(2,2, 1)),
    model = "EVV")
stopifnot( all.equal(MW22, m22) )

m213 <- norMmix(
    name = "#13 test VVV",
    weight = c(0.5, 0.5),
    mu = cbind( c(0,0), c(30,30) ),
    Sigma = array(c( 1,3,3,11, 3,6,6,13 ), dim=c(2,2, 2)),
    model = "VVV")
stopifnot( all.equal(MW213, m213) )
str(m213)

m34 <- norMmix(
    name = "#4 3d VEI",
    weight = c(0.1, 0.9),
    mu = matrix(rep(0,6), 3,2),
    Sigma = array(c(diag(1:3), 0.2*diag(3:1)), c(3,3, 2)),
    model = "VVI" )
stopifnot( all.equal(MW34, m34) )
```

norMmixMLE *Maximum Likelihood Estimation for Multivariate Normal Mixtures*

Description

Direct Maximum Likelihood Estimation (MLE) for multivariate normal mixture models "`norMmix`". Starting from a `clara` (package **cluster**) clustering plus one M-step by default, or alternatively from the default start of (package) **mclust**, perform direct likelihood maximization via `optim()`.

Usage

```
norMmixMLE(x, k,
           model = c("EII", "VII", "EEI", "VEI", "EVI",
                     "VVI", "EEE", "VEE", "EVV", "VVV"),
           initFUN = claraInit,
           ll = c("nmm", "mvt"),
           keep.optr = TRUE, keep.data = keep.optr,
           method = "BFGS", maxit = 100, trace = 2,
           optREPORT = 10, reltol = sqrt(.Machine$double.eps),
     ...)

claraInit(x, k, samples = 128,
          sampsize = ssClara2kL, trace)
mclVVinit(x, k, ...)

ssClara2kL(n, k, p)
```

Arguments

`x`	numeric [n x p] matrix
`k`	positive number of components
`model`	a `character` string, specifying the model (for the `k` covariance matrices) to be assumed.

`initFUN`	a `function`, that takes arguments `x` and `k` and returns a clustering index; a vector of length p =`ncol(x)`, with entries in `1:k`.
`ll`	a string specifying the method to be used for the likelihood computation; the default, `"nmm"` uses `llnorMmix()`, whereas `"mvt"` uses `llmvtnorm()` which is based on the MV normal density from package **mvtnorm**.
`keep.optr`, `keep.data`	`logical`, each indicating of the optimization result (from `optim()`, currently), or the data `x` respectively, should be saved as part of the result (function 'value', see also below).
`method`, `maxit`, `optREPORT`, `reltol`	arguments for tuning the optimizer `optim(*, method=method, control = list(...))`.
`trace`	**in** `norMmixMLE()`**:** passed to `optim(*, control=..)`, see above. **in** `claraInit()`**:** a non-negative integer indicating how much `clara()` calls should be traced.
`...`	**in** `norMmixMLE()`**:** passed to `optim(*, control=..)`, see above. **in** `mclVVVinit()`**:** further arguments passed to (package **mclust**) function `hcVVV()`.
`samples`	the number of subsamples to take in `clara()`, package **cluster**, see its help.
`sampsize`	the sample size to take in `clara()`, package **cluster**. Here, can be a positive integer *or*, as by default, a `function` with arguments `(n,k,p)`.
`n`, `p`	matrix dimensions `nrow(x)` and `ncol(x)`.

Details

By default, `initFUN=claraInit`, uses `clara()` and one M-step from EM-algorithm to initialize parameters after that uses general optimizer `optim()` to calculate the MLE.

To silence the output of `norMmixMLE`, set `optREPORT` very high and `trace` to 0. For details on output behavior, see the "details" section of `optim`.

Value

`norMmixMLE` returns an object of `class` `"norMmixMLE"` which is a `list` with components

`norMmix`	the `"norMmix"` object corresponding to the specified model and the fitted (MLE) parameter vector.
`optr`	(if `keep.optr` is true:) the [r]eturn value of optimization, i.e., currently, `optim()`.
`npar`	the number of free parameters, a function of $(p, k, model)$.
`n`	the sample size, i.e., the number of observations or rows of `x`.
`cond`	the result of (the hidden function) `parcond(..)`, that is the ratio of sample size over parameter count.
`x`	(if `keep.optr` is true:) the $n \times p$ data matrix.

Examples

```
MW214
set.seed(105)
x <- rnorMmix(1000, MW214)

## Fitting, assuming to know the true model (k=6, "VII")
fm1  <- norMmixMLE(x, k = 6, model = "VII", initFUN=claraInit)
fm1 # {using print.norMmixMLE() method}
fm1M <- norMmixMLE(x, k = 6, model = "VII", initFUN=mclVVVinit)

## Fitting "wrong" overparametrized model: typically need more iterations:
fmW <- norMmixMLE(x, k = 7, model = "VVV", maxit = 200, initFUN=claraInit)
## default maxit=100 is often too small    ^^^^^^^^^^^


x <- rnorMmix(2^12, MW51)
fM5 <- norMmixMLE(x, k = 4) # k = 3 is sufficient
fM5
c(logLik = logLik(fM5), AIC = AIC(fM5), BIC = BIC(fM5))
plot(fM5, show.x=FALSE)
plot(fM5, lwd=3, pch.data=".")

# this takes several seconds
 fM5big <- norMmixMLE(x, model = "VVV", k = 4, maxit = 300) # k = 3 is sufficient
 summary(warnings())
 fM5big ; c(logLik = logLik(fM5big), AIC = AIC(fM5big), BIC = BIC(fM5big))
 plot(fM5big, show.x=FALSE)
```

---

npar — *Degrees of freedom of (Fitted) Multivariate Normal Mixtures*

Description

This function is generic; method functions can be written to handle specific classes of objects. The following classes have methods written for them:

norMmix

norMmixMLE

fittednorMmix

Usage

```
npar(object, ...)
```

Arguments

`object`	any R object from the list in the 'Description'.
`...`	potentially further arguments for methods; Currently, none of the methods for the listed classes do have such.

Value

Depending on `object` :

`norMmix`	integer number.
`norMmixMLE`	integer number.
`fittednorMmix`	integer`matrix` with `dimnames` set to k and models.

Author(s)

Nicolas Trutmann

See Also

`norMmix`, `norMmixMLE`.

Examples

```
methods(npar) # list available methods

npar(MW213)
```

`par2nMm` *Transform Parameter Vector to Multivariate Normal Mixture*

Description

Transforms the (numeric) parameter vector of our MLE parametrization of a multivariate normal mixture model into the corresponding `list` of components determining the model. Additionally (partly redundantly), the dimension `p` and number of components `k` need to be specified as well.

Usage

```
par2nMm(par, p, k, model = c("EII","VII","EEI","VEI","EVI",
                              "VVI","EEE","VEE","EVV","VVV")
      , name = sprintf("model = %s , components = %s", model, k)
       )
```

Arguments

`par`	the model parameter numeric vector.
`p`	dimension of data space, i.e., number of variables (aka “features”).
`k`	the number of mixture components, a positive integer.
`model`	a `character` string, one of those listed; see `nMm2par()`’s documentation.
`name`	a `character` string naming the `norMmix` return value.

Value

returns a `list` with components

`weight`	..
`mu`	..
`Sigma`	..
`k`	..
`dim`	..

See Also

This is the inverse function of `nMm2par()`.

Examples

```
## TODO: Show to get the list, and then how to get a  norMmix() object from the list
str(MW213)
# List of 6
#  $ model : chr "VVV"
#  $ mu    : num [1:2, 1:2] 0 0 30 30
#  $ Sigma : num [1:2, 1:2, 1:2] 1 3 3 11 3 6 6 13
#  $ weight: num [1:2] 0.5 0.5
#  $ k     : int 2
#  $ dim   : int 2
#  - attr(*, "name")= chr "#13 test VVV"
#  - attr(*, "class")= chr "norMmix"

para <- nMm2par(MW213, model="EEE")
par2nMm(para, 2, 2, model="EEE")
```

`plot.norMmix` *Plot Method for "norMmix" Objects*

Description

This is the S3 method for plotting "`norMmix`" objects.

Usage

```
## S3 method for class 'norMmix'
plot(x, y=NULL, ...)
## S3 method for class 'norMmixMLE'
plot(x, y = NULL,
     show.x = TRUE,
     main = sprintf(
         "norMmixMLE(*, model=\"%s\") fit to n=%d observations in %d dim.",
         nm$model, x$nobs, nm$dim
```

```
        ),
        sub = paste0(
            sprintf("log likelihood: %g; npar=%d", x$logLik, x$npar),
         if (!is.null(opt <- x$optr)) paste("; optim() counts:", named2char(opt$counts))
        ),
        cex.data = par("cex") / 4, pch.data = 4,
        ...)

## S3 method for class 'fittednorMmix'
plot(x, main = "unnamed", plotbest = FALSE, ...)

plot2d (nMm, data = NULL,
        add = FALSE,
        main = NULL,
        sub = NULL,
        type = "l", lty = 2, lwd = if (!is.null(data)) 2 else 1,
        xlim = NULL, ylim = NULL, f.lim = 0.05,
        npoints = 250, lab = FALSE,
        col = Trubetskoy10[1],
        col.data = adjustcolor(par("col"), 1/2),
        cex.data = par("cex"), pch.data = par("pch"),
        fill = TRUE, fillcolor = col, border = NA,
        ...)
plotnd(nMm, data = NULL,
       main = NULL,
       diag.panel = NULL,
       ...)
Trubetskoy10
```

Arguments

x, nMm	an R object inheriting from "`norMmix`".
y	further data matrix, first 2 columns will be plotted by "`points`"
...	further arguments to be passed to another plotting function.
show.x	Option for `plot.norMmixMLE`. Plot data points along with estimated model. Defaults to `TRUE`.
data	Data points to plot.
add	This argument is used in the internal function, `plot2d`, to control whether to create a new plot or add to an existing one. Should not be set by the user. Defaults to `FALSE`
main	Set main title. See `Usage` section for default values.
sub	Set subtitle. See `Usage` section for default values.
type	Graphing type for ellipses border. Defaults to "l".
lty	Line type to go with the `type`. See "`par`".
lwd	Line width as in `lty`.
xlim	Set explicit x limits for 2d plots.

`ylim`	As `xlim`.
`f.lim`	Percentage value for how much to extend `xlim` and `ylim`. As in the `f` argument to "`extendrange`".
`npoints`	How many points to use in the drawn ellipses. Larger values make them prettier but might affect plot times.
`lab`	Whether to print labels for mixture components. Will print "comp
`col`	Fill color for ellipses. Default is "#4363d8".
`col.data`	Color to be used for data points.
`cex.data`	See "`par`".
`pch.data`	See "`par`".
`fill`	Leave ellipses blank with outline or fill them in.
`fillcolor`	Color for infill of ellipses.
`border`	Argument to be passed to `polygon`.
`diag.panel`	Function to plot 2d projections of a higher-dimensional mixture model. Used by `plotnd`. Requires function with signature `function(x, y, data = NULL, ...)` Should not be set by the user.
`plotbest`	Used by `fittednorMmix`. Plot best fitting model using `plot.norMmix`.

Details

The plot method calls one of two auxiliary functions, one for dim=2, another for higher dimensions. The method for 2 dimensional plots also takes a `add` parameter (`FALSE` by default), which allows for the ellipses to be drawn over an existing plot.

The higher dimensional plot method relies on the `pairs.default` function to draw a lattice plot, where the panels are built using the 2 dimensional method.

`Trubetskoy10`: A vector of colors for these plots, chosen to be distinguishable and accessible for the colorblind, according to `https://sashamaps.net/2017/01/11/list-of-20-simple-distinct-colors/`, slightly rearranged, so that the first five colors stand out well on white background.

Value

`plot.norMmix` In the 2 dimensional case, returns invisibly coordinates of bounding ellipses of distribution.

Examples

```
plot(MW212) ## and add a finite sample realization:
points(rnorMmix(n=500, MW212))

## or:
x <- points(rnorMmix(n=500, MW212))
plot(MW212, x)


## Example of dim. = p > 2 :
plot(MW34)
```

rnorMmix — *Random Sample from Multivariate Normal Mixture Distribution*

Description

Draw n (p-dimensional) observations randomly from the multivariate normal mixture distribution specified by obj.

Usage

```
rnorMmix(n, obj, index = FALSE, permute = TRUE)
```

Arguments

n	sample size, non-negative.
obj	a "norMmix" object
index	Logical, store the clustering information as first column
permute	Logical, indicating if the observations should be randomly permuted after creation "cluster by cluster".

Value

n p-dimensional observations, as numeric $n \times p$ matrix.

Author(s)

Nicolas Trutmann

See Also

rmultinom

Examples

```
x <- rnorMmix(500, MW213)
plot(x)
x <- rnorMmix(500, MW213, index=TRUE)
plot(x[,-1], col=x[,1]) ## using index column to color components
```

sllnorMmix *Simple wrapper for Log-Likelihood Function or Multivariate Normal Mixture*

Description

sllnorMmix() returns a number, the log-likelihood of the data x, given a normal mixture obj.

Usage

```
sllnorMmix(x, obj)
```

Arguments

x	data matrix.
obj	an R object of class "norMmix".

Details

Calculates log-likelihood of a dataset, x, given a normal mixture model; just a simplified wrapper for llnorMmix. Removes functionality in favor of ease of use.

Value

double. See description.

Examples

```
set.seed(2019)
x <- rnorMmix(400, MW27)
sllnorMmix(x, MW27) # -1986.315
```

Index