Package 'norMmix'

October 27, 2019

Version 0.0-1

Title Direct MLE for Multivariate Normal Mixture Distributions

Description Compute the MLE for multivariate normal mixtures via smart parametrization using the LDLt decomposition.

Author Nicolas Trutmann [aut, cre],
Martin Maechler [aut, ths]

Maintainer Nicolas Trutmann <nicolatr@student.ethz.ch>

Imports cluster, MASS, mvtnorm, mclust, mixtools, sfsmisc

Suggests Matrix, testthat (>= 2.1.0)

License GPL-3

Encoding UTF-8

LazyData true

NeedsCompilation no

Depends R (>= 3.5.0)

R topics documented:

`compplot` *composition plot*

Description

Creates a `massplot` like plot. Takes two `massbic` arrays and overlays their plots.

Usage

```
compplot(f, g, h=NULL, main = "unnamed", adj = 1/dim(f)[3],
         col = nMmcols[1:3], mar = 0.1 + c(1.4, 2, 3, 1),
         compnames = c("clara", "mclVVV", "Mclust"),
         oma=c(7,0,2.8,0),
      ...)
```

Arguments

`f`	Array as from `massbic` or `massbicm`
`g`	Array as from `massbic` or `massbicm`
`h`	Optional third array.
`main`	Character string to be used as title of the plot.
`adj`	Adjustment for alpha value of the plot color.
`col`	Color of plot.
`mar`	Margin space adjustment
`compnames`	placeholder
`oma`	Margin parameter. See par.
`...`	further arguments passed on to `matplot`

Details

The intended use for this function is to run `massbic` and `massbicm` with the same arguments, `string` and `DIR` and feeding the results into `compplot`, producing line plots, that can be compared against each other. Currently uses `rainbow(20)` to generate plot colors. Red, orange and green correspond to `f` and cyan, blue and purple to `g`.

Value

No return value. Intended side effect is the generation of a plot.

Note

While possible to be used as standalone, the function `epfl` allows for more complex generation of PDFs and is the intended use for simulations.

Author(s)

Nicolas Trutmann

See Also

`massplot`, `epfl`, `massbic`, `massbicm`

Examples

```
## TODO: add example
```

---

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'.

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)
```

epfl *evaluate and plot from file list*

Description

From a list of character vectors, will apply massbic, massbicm followed by massplot and compplot. Saves result of massbicm

Usage

```
epfl(files, savdir, subt = 11, ...)
```

Arguments

files	Expected to be of type list, containing character vectors. These are assumed to be RDS filenames produced from saveRDS. They are assumed to work with massbic, meaning they contain a list with named element fit, which is the return value of fitnMm. Furthermore, all are assumed to have the same dimensions.
savdir	String specifying directory in which files are located.
subt	Number of characters to be subtracted from string in files. The default is intended for format ending in "seed=
...	Arguments to be passed to massplot and compplot

Details

Suppose you have a directory dir containing RDS files. To create a list of sorted filenames, use for example the code provided here:

#files <- list.files(dir, pattern=".rds") # p1 <- c("pattern1a", "pattern1b") ## search patterns # p2 <- c("pattern2a", "pattern2b")

filelist <- list() # for (i in seq_along(p1)) # for (j in seq_along(p2)) # # for lack of AND matching, OR match everything else and invert # r <- grep(paste(p1[-i], p2[-j], sep="|", collapse="|"), # files, value=TRUE, invert=TRUE) # filelist[[paste0(p1[i], p2[j])]] <- r #

This will create a list of character vectors, matching filenames by p1 and p2.

Value

No return value. Intended side effects: Produces 3*length(files) PDFs named as follows: From each entry in files takes first string, subtracts subt characters and appends ".pdf","_mcl.pdf" and "_comp.pdf"

Author(s)

Nicolas Trutmann

See Also

massplot, compplot, fitnMm

Examples

```
# TODO: add example
```

`extracttimes` *Extract system time from* `fittednorMmix` *object*

Description

extracts array of `system.time` values from a `fittednorMmix` object.

Usage

```
extracttimes(object, ...)
```

Arguments

`object`	a `fittednorMmix` object.
`...`	currently unused

Details

a

Value

Array of size components x models x 5(return values of system.time) with dimnames: `k`, `models` and `proc_time`.

Author(s)

Nicolas Trutmann

See Also

`system.time`

Examples

```
data(fSMI.12, package="norMmix")
extracttimes(fSMI.12)[,,1] ## user.self entry of system.time
```

`fitnMm` *Fit Several Normal Mixture Models to a Dataset*

Description

`fitnMm()` fits several multivariate normal mixture models (`norMmix`) to the data set `x`, and returns (among other info) a `list` of fitted `norMmix` objects.

Usage

```
fitnMm(x, k, models = 1:10,
            trafo = c("clr1", "logit"),
            ll = c("nmm", "mvt"),
            savdir = NULL, name = NULL,
          ...)
```

Arguments

`x`	data matrix, rows are observations and columns are variables
`k`	vector of positive integers, indicating the *number* of mixture components to fit.
`models`	vector of integers from `1:10`, indexing models from `c("EII", "VII", "EEI", "VEI", "EVI", "VVI", "EEE", "VEE", "EVV", "VVV")` which are to be fit.
`trafo`	a `character` string specifying the transform to use for the weights `w` ($= \pi_j$), currently either `"clr1"` or `"logit"`.
`ll, ...`	further arguments passed to function `norMmixMLE()`.
`savdir`	valid file path to save directory
`name`	name to be used when saving return value as RDS file

Value

`fitnMm()` returns a `list` with components

`nMm`	a `list` containing all fitted models
`models`	`character` vector of model (names) that were fitted.
`n`	number of observations of x
`p`	number of variables of x

Note

Given an object `r` of class `fittednorMmix`, for use with `massbic`, do:

saveRDS(list(fit=r))

See Also

norMmixMLE() which is called length(k) * length(models) times.

Examples

```
x <- rnorMmix(500, MW21)
fitnMm(x, 1:2, models=1:4) ## will fit models 1:4 with 1:3 components
```

---

fSMI.12 *Model selection of the SMI.12 dataset from the SMI.12 package.*

Description

Result of fitnMm(SMI.12,k=1:8,ini="clara",maxit=1e4,optREPORT=1e4)

Usage

```
data("fSMI.12")
```

Format

The format is: List of 7 $ nMm :List of 80 ..- attr(*, "dim")= int [1:2] 8 10 ..- attr(*, "dimnames")=List of 2 $ nMmtime:List of 80 ..- attr(*, "dim")= int [1:2] 8 10 ..- attr(*, "dimnames")=List of 2 $ k : int [1:8] 1 2 3 4 5 6 7 8 $ models : chr [1:10] "EII" "VII" "EEI" "VEI" ... $ n : int 141 $ p : int 20 $ x : num [1:141, 1:20] 16.1 15.7 15.7 16.1 16.6 ... ..- attr(*, "dimnames")=List of 2 - attr(*, "class")= chr "fittednorMmix"

Examples

```
data(fSMI.12)
## maybe str(fSMI.12) ; plot(fSMI.12) ...
```

---

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) )


## 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
)

if(interactive())
   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,
          trafo = c("clr1", "logit"),
          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.
`trafo`	a `character` string specifying the transform to use for the weights `w` ($= \pi_j$), currently either `"clr1"` or `"logit"`.
`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,
          trafo = c("clr1", "logit"),
          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.
`trafo`	a `character` string specifying the transform to use for the weights `w` ($= \pi_j$), currently either `"clr1"` or `"logit"`.
`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
```

Examples

```
MW210
plot(MW214)
```

massbic *extract BIC from .rds files*

Description

Given a filelist of RDS files, extracts BIC values from each file and returns them in an array.

Usage

```
massbic(string, DIR)
```

Arguments

`string`

`DIR`

Value

array of dimensions: components * models * files. Has attribute `dims` integer vector of same length as `files`. Correspond to dimension of dataset.

Author(s)

Nicolas Trutmann

See Also

`massbicm` `fitnMm` `massplot`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (string, DIR)
{
    nm1 <- readRDS(file = file.path(DIR, string[1]))
    cl <- nm1$fit$k
    mo <- nm1$fit$models
    val <- array(0, lengths(list(cl, mo, string)))
    dims <- vector(mode = "integer", length = length(string))
    for (i in 1:length(string)) {
        nm <- readRDS(file = file.path(DIR, string[i]))
        val[, , i] <- BIC(nm$fit)[[1]]
        dims[i] <- nm$fit$p
    }
    dimnames(val) <- list(components = cl, models = mo, simulation = string)
```

```
  attr(val, "dims") <- dims
  val
}
```

---

massbicm *Do mclust along .rds files from fitnMm*

Description

massbicm applies Mclust along an existing fittednorMmix object, assumed to be in an RDS file in a list, named 'fit'.

Usage

```
massbicm(string, DIR)
```

Arguments

string

DIR

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (string, DIR)
{
    nm <- readRDS(file.path(DIR, string[1]))
    cl <- nm$fit$k
    mo <- nm$fit$models
    valm <- array(0, lengths(list(cl, mo, string)))
    dims <- vector(mode = "integer", length = length(string))
    for (i in 1:length(string)) {
        nm <- readRDS(file.path(DIR, string[i]))
        x <- nm$fit$x
        valm[, , i] <- mclust::Mclust(x, G = cl, modelNames = mo)$BIC
        dims[i] <- nm$fit$p
    }
    dimnames(valm) <- list(components = cl, models = mo, files = string)
    attr(valm, "dims") <- dims
    -valm
  }
```

massplot *plot from massbic*

Description

plots the result of massbic and massbicm

Usage

```
massplot(f, main = "unnamed", adj = exp(-0.002 * size),
         col = nMmcols[1], mar = 0.1 + c(1.4, 2, 3, 1), ...)
```

Arguments

f	Result of massbic
main	Character string, title of plot.
adj	Alpha adjustment for plot color.
col	Plot color.
mar	A numerical vector of the form 'c(bottom, left, top, right)'.
...	Further parameters, passed to matplot

Author(s)

Nicolas Trutmann

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (f, main = "unnamed")
{
    ran <- extendrange(f)
    size <- dim(f)[3]
    cl <- as.numeric(dimnames(f)$components)
    p <- attr(f, "dims")
    adj <- exp(-0.002 * size)
    models <- mods()
    op <- sfsmisc::mult.fig(mfrow = c(4, 5), main = main, mar = 0.1 +
        c(2, 4, 4, 1))
    for (i in 1:10) {
        if (!is.null(p)) {
            matplot(f[, i, ], lty = 1, col = adjustcolor(rainbow(10)[i],
                adj), type = "l", ylim = ran, main = models[i])
            axis(3, at = seq_along(cl), labels = npar(cl, p[1],
```

```
            models[i]))
        }
        else {
            matplot(f[, i, ], lty = 1, col = adjustcolor(rainbow(10)[i],
                adj), main = models[i], type = "l", ylim = ran)
        }
    }
    for (i in 1:10) {
        boxplot(t(f[, i, ]), lty = 1, col = adjustcolor(rainbow(10)[i],
            0.4), main = models[i], type = "l", ylim = ran)
    }
    par(op$old.par)
}
```

---

nc2p *Wrapper function for* nMm2par

Description

nc2p returns same as nMm2par, using object$model and trafo="clr1" as defaults.

Usage

```
nc2p(object)
```

Arguments

object	an "norMmix" object.

Value

the same as nMm2par. Real valued vector of length 1.

```
nMm2par(object,trafo="clr1",model=object$model)
```

Examples

```
str(MW213)
nc2p(MW213)
#  [1]  0.0000000  0.0000000  0.0000000 30.0000000 30.0000000  0.3465736
#  [7]  0.5493061  0.3465736 -0.5493061  3.0000000  2.0000000
nMm2par(MW213, trafo="clr1", model=MW213$model)
#  [1]  0.0000000  0.0000000  0.0000000 30.0000000 30.0000000  0.3465736
#  [7]  0.5493061  0.3465736 -0.5493061  3.0000000  2.0000000
```

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, trafo = c("clr1", "logit"),
        model = c("EII", "VII", "EEI", "VEI", "EVI",
                  "VVI", "EEE", "VEE", "EVV", "VVV"),
        meanFUN = mean)
```

Arguments

obj	a list containing sig: covariance matrix array, mu: mean vector matrix, w: = weights, k: = number of components, p: = dimension
trafo	a character string specifying the transform to use for the weights w ($= \pi_j$), currently either "clr1" or "logit".
model	a character string specifying the (Sigma) model, one of those listed above.
meanFUN	a function to compute a mean (of variances typically).

Details

This transformation forms a vector from the parameters of a normal mixture. These consist of weights, means and covariance matrices. Weights are transformed according to 'trafo' param; means are unchanged.

Cov mats are given as D and L from the LDLt decomposition

See Also

nc2p simplified wrapper function.

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

Examples

```
A <- MW24
nMm2par(A, trafo = "clr1", model = A$model)
# [1] -0.3465736  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
# [7] -2.3025851
```

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 $j = 1, \ldots, k$.

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. should mu be a vector it will assume k=1 to circumvent this behavoiur use as.matrix(mu) beforehand
Sigma	array of covariance matrices
weight	weights of mixture model components
name	gives the option of naming mixture
model	see desc

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

```
# TODO
```

---

norMmixMLE *Maximum Likelihood Estimation for Multivariate Normal Mixture Models*

Description

Direct Maximum Likelihood Estimation (MLE) for multivariate normal mixture models "norMmix". Starting from a clara (package **cluster**) clustering plus one M-step, 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"),
           ini = c("clara", "mclVVV"),
           trafo = c("clr1", "logit"),
           ll = c("nmm", "mvt"),
           method = "BFGS", maxit = 100, trace = 2,
           optREPORT = 10, reltol = sqrt(.Machine$double.eps),
           samples = 128, sampsize = ssClaraL, traceClara = 0,
     ...)

ssClaraL(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.
ini	a character string specifying the initialization step.
trafo	a character string specifying the transform to use for the weights w ($= \pi_j$), currently either "clr1" or "logit".
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**.

`method, maxit, trace, optREPORT, reltol, ...`
: arguments for tuning the optimizer `optim(*,method=method,control = list(...))`.

`samples, sampsize, traceClara`
: if `ini = "clara"`, arguments for `clara()` (package **cluster**). Note that `clara`'s help page emphasizes that larger and more samples should be used typically. Here, `sampsize` may be a number *or* as by default a `function(n,k,p)` determining the size of the subsamples as a function of the problem dimensionalities.

`n,p`
: matrix dimensions `nrow(x)` and `ncol(x)`.

Details

Uses clara() and one M-step from EM-algorithm to initialize parameters after that uses general optimizer optim() to calculate ML.

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`
: the [r]eturn value of `optim()`.

`npar`
: the number of free parameter, a function of $(p, k, model)$.

`n`
: the sample size, i.e., the number of observations or rows of `x`.

`cond`
: the result of `parcond(..)`, that is the ratio of sample size over parameter count.

Examples

```
str(MW214)
set.seed(105)
x <- rnorMmix(1000, MW214)
## Fitting assuming we know the true parametric model
fm1 <- norMmixMLE(x, k = 6, model = "VII")
if(interactive()) ## Fitting "wrong" overparametrized model: typically need more iterations:
fmW <- norMmixMLE(x, k = 7, model = "VVV", maxit = 200)# default maxit=100 is often too small
```

`npar` *Extract degrees of freedom from objects of the* `norMmix` *package*

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 object from the list in the Description.
...	In place to leave room for further arguments. None of the methods for the listed classes take arguments beyond object.

Value

norMmix	Integer vector of length 1.
norMmixMLE	Integer vector of length 1.
fittednorMmix	Integer matrix with dimnames set to k and models.

Author(s)

Nicolas Trutmann

Examples

```
data(fSMI.12)
npar(fSMI.12)

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")
      , trafo = c("clr1", "logit")
      , 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.
trafo	a character string specifying the transform to use for the weights w ($= \pi_j$), currently either "clr1" or "logit".
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"
# NULL
para <- nMm2par(MW213, model="EEE")
par2nMm(para, 2, 2, model="EEE")
```

plot.fittednorMmix *Plot method for the class* fittednorMmix

Description

This is the S3 method for plotting the results of fitnMm

Usage

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

Arguments

x	object of class "fittednorMmix"
main	plot title
plotbest	logical, determines whether to plot BIC values or best fitted model. See Details.
...	further arguments to be passed to plot if plotbest=TRUE, and matplot if FALSE

Details

This plot method has two main capabilities, selected by the argument plotbest. If plotbest is TRUE, then the model will be plotted using the plot.norMmix method with added points of the fitted data. And if plotbest is FALSE, then the BIC values will be plotted using matplot

See Also

fitnMm, norMmix, plot.norMmix

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, ...)
```

Arguments

x	an R object inheriting from "norMmix".
y	further data matrix, first 2 columns will be plotted by "points"
...	further arguments to be passed to "plot"

Value

`plot.norMmix` 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)
```

`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, trafo=c("clr1", "logit"))
```

Arguments

`x`	data `matrix`.
`obj`	an R object of class `"norMmix"`.
`trafo`	a `character` string specifying the transform to use for the weights, see `llnorMmix`.

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.

Examples

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

Index