Package ‘ashr’

August 22, 2023

Encoding UTF-8
Type Package
Maintainer Peter Carbonetto <pcarbo@uchicago.edu>
Version 2.2-63
Date 2023-08-21
Title Methods for Adaptive Shrinkage, using Empirical Bayes
Description The R package ‘ashr’ implements an Empirical Bayes approach for large-scale hypothesis testing and false discovery rate (FDR) estimation based on the methods proposed in M. Stephens, 2016, "False discovery rates: a new deal", <DOI:10.1093/biostatistics/kxw041>. These methods can be applied whenever two sets of summary statistics---estimated effects and standard errors---are available, just as ‘qvalue’ can be applied to previously computed p-values. Two main interfaces are provided: ash(), which is more user-friendly; and ash.workhorse(), which has more options and is geared toward advanced users. The ash() and ash.workhorse() also provides a flexible modeling interface that can accommodate a variety of likelihoods (e.g., normal, Poisson) and mixture priors (e.g., uniform, normal).
Depends R (>= 3.1.0)
Imports Matrix, stats, graphics, Rcpp (>= 0.10.5), truncnorm, mixsqp, SQUAREM, etrunct, invgamma
Suggests testthat, knitr, rmarkdown, ggplot2, REBayes
LinkingTo Rcpp
License GPL (>= 3)
NeedsCompilation yes
URL https://github.com/stephens999/ashr
BugReports https://github.com/stephens999/ashr/issues
VignetteBuilder knitr
RoxygenNote 7.1.2
Author  Matthew Stephens [aut],
       Peter Carbonetto [aut, cre],
       Chaoxing Dai [ctb],
       David Gerard [aut],
       Mengyin Lu [aut],
       Lei Sun [aut],
       Jason Willwerscheid [aut],
       Nan Xiao [aut],
       Mazon Zeng [ctb]

Repository  CRAN

Date/Publication  2023-08-21 23:50:03 UTC

R topics documented:

ash .......................................................... 4
ashci ..................................................... 9
ashr ....................................................... 10
ash_pois .................................................. 10
calc_loglik ............................................... 11
calc_logLR ................................................ 12
calc_mixmean ............................................ 12
calc_mixsd ............................................... 13
calc_null_loglik ....................................... 13
calc_null_vloglik ...................................... 14
calc_vloglik ............................................. 14
calc_vlogLR ............................................. 15
cdf.ash ................................................. 15
cdf_conv ................................................. 16
cdf_post .................................................. 16
compute_lfsr ........................................... 17
comp_cdf ................................................ 17
comp_cdf_conv .......................................... 18
comp_cdf_conv.normalmix .............................. 18
comp_cdf_conv.unimix ................................ 19
comp_cdf_post ......................................... 19
comp_dens ............................................... 20
comp_dens_conv ........................................ 20
comp_dens_conv.normalmix ............................ 21
comp_dens_conv.unimix ................................ 21
comp_mean ............................................... 22
comp_mean.normalmix ................................. 22
comp_mean.tnormalmix ................................. 23
comp_mean2 ............................................. 23
comp_postmean .......................................... 24
comp_postmean2 ........................................ 24
comp_postprob .......................................... 25
comp_poststd ............................................ 25
R topics documented:

comp_sd ................................................................. 26
comp_sd.normalmix .................................................... 26
comp_sd.normalmix .................................................... 27
cxxMixSquarem .......................................................... 27
dens .............................................................. 28
dens_conv ............................................................ 28
dlogf ................................................................. 29
estimate_mixprop ........................................................ 29
gen_etrunFUN ........................................................... 30
get_density ............................................................ 31
get_lfsr ................................................................. 31
get_post_sample ........................................................ 33
igmix .............................................................. 33
lik_binom ............................................................. 34
lik_logF ............................................................... 35
lik_normal ............................................................. 35
lik_normalmix ........................................................... 36
lik_pois ............................................................... 36
lik_t ................................................................. 37
loglik_conv ........................................................... 37
loglik_conv.default ................................................... 38
log_comp_dens_conv .................................................... 38
log_comp_dens_conv.normalmix ....................................... 39
log_comp_dens_conv.unimix .......................................... 39
mixcdf .............................................................. 40
mixcdf.default ........................................................ 40
mixEM ............................................................... 41
mixIP ............................................................... 42
mixmean2 ............................................................. 43
mixprop .............................................................. 43
mixSQP ............................................................... 44
mixVBEM ............................................................. 44
my_e2truncbeta ........................................................ 45
my_e2truncgamma ...................................................... 46
my_e2trunctnorm ...................................................... 46
my_e2runct ........................................................... 47
my_etruncbeta ........................................................ 47
my_etrunctxgamma .................................................... 48
my_etrunclnorm ...................................................... 48
my_etrunct ........................................................... 49
my_vtrunctnorm ....................................................... 50
ncomp ............................................................... 51
ncomp.default ........................................................ 51
normalmix ............................................................ 51
pcdf_post .............................................................. 52
plogf ............................................................... 53
plot.ash .............................................................. 53
Adaptive Shrinkage

Description

Implements Empirical Bayes shrinkage and false discovery rate methods based on unimodal prior distributions.

Usage

```
ash(
  betahat,
  sebetahat,
  mixcompdist = c("uniform", "halfuniform", "normal", "+uniform", "-uniform",
  "halfnormal"),
  df = NULL,
  ...
)
```

```
ash.workhorse(  
  betahat,  
  sebetahat,  
  method = c("fdr", "shrink"),  
  mixcompdist = c("uniform", "halfuniform", "normal", "+uniform", "-uniform",
  "halfnormal"),  
  optmethod = c("mixSQP", "mixIP", "cxxMixSquarem", "mixEM", "mixVBEM", "w_mixEM"),  
  df = NULL,  
)
nullweight = 10,  
pointmass = TRUE,  
prior = c("nullbiased", "uniform", "unit"),  
mixsd = NULL,  
gridmult = sqrt(2),  
outputlevel = 2,  
g = NULL,  
fixg = FALSE,  
mode = 0,  
alpha = 0,  
grange = c(-Inf, Inf),  
control = list(),  
lik = NULL,  
weights = NULL,  
pi_thresh = 1e-10
)

Arguments

betahat a p vector of estimates
sebetahat a p vector of corresponding standard errors
mixcompdist distribution of components in mixture used to represent the family G. Depending on the choice of mixture component, the family G becomes more or less flexible. Options are:

uniform G is (approximately) any symmetric unimodal distribution
normal G is (approximately) any scale mixture of normals
halfuniform G is (approximately) any unimodal distribution
+uniform G is (approximately) any unimodal distribution with support constrained to be greater than the mode.
-uniform G is (approximately) any unimodal distribution with support constrained to be less than the mode.
halfnormal G is (approximately) any scale mixture of truncated normals where the normals are truncated at the mode

If you are happy to assume a symmetric distribution for effects, you can use "uniform" or "normal". If you believe your effects may be asymmetric, use "halfuniform" or "halfnormal". If you want to allow only positive/negative effects use "+uniform"/"-uniform". The use of "normal" and "halfnormal" is permitted only if df=NULL.

df appropriate degrees of freedom for (t) distribution of (betahat-beta)/sebetahat; default is NULL which is actually treated as infinity (Gaussian)

... Further arguments of function ash to be passed to ash.workhorse.

method specifies how ash is to be run. Can be "shrinkage" (if main aim is shrinkage) or "fdr" (if main aim is to assess false discovery rate or false sign rate (fsr)). This is simply a convenient way to specify certain combinations of parameters: "shrinkage" sets pointmass=FALSE and prior="uniform"; "fdr" sets pointmass=TRUE and prior="nullbiased".
optmethod specifies the function implementing an optimization method.

nullweight scalar, the weight put on the prior under "nullbiased" specification, see prior

pointmass Logical, indicating whether to use a point mass at zero as one of components for a mixture distribution.

prior string, or numeric vector indicating Dirichlet prior on mixture proportions: "nullbiased", c(nullweight,1,...,1), puts more weight on first component; "uniform" is c(1,1,...,1); "unit" is (1/K,...,1/K), for optmethod = mixVBEM version only.

mixsd Vector of standard deviations for underlying mixture components.

gridmult the multiplier by which the default grid values for mixsd differ by one another. (Smaller values produce finer grids.)

outputlevel Determines amount of output. There are several numeric options: 0 = just fitted g; 1 = also PosteriorMean and PosteriorSD; 2 = everything usually needed; 3 = also include results of mixture fitting procedure (including matrix of log-likelihoods used to fit mixture). 4 and 5 are reserved for outputting additional data required by the (in-development) flashr package. The user can also specify the output they require in detail (see Examples).

g The prior distribution for beta. Usually this is unspecified (NULL) and estimated from the data. However, it can be used in conjunction with fixg=TRUE to specify the g to use (e.g. useful in simulations to do computations with the "true" g). Or, if g is specified but fixg=FALSE, the g specifies the initial value of g used before optimization, (which also implicitly specifies mixcompdist).

fixg If TRUE, don’t estimate g but use the specified g - useful for computations under the "true" g in simulations.

mode either numeric (indicating mode of g) or string "estimate", to indicate mode should be estimated, or a two dimension numeric vector to indicate the interval to be searched for the mode.

alpha Numeric value of alpha parameter in the model.

grange Two dimension numeric vector indicating the left and right limit of g. Default is c(-Inf, Inf).

control A list of control parameters passed to optmethod.

lik Contains details of the likelihood used; for general ash. Currently, the following choices are allowed: normal (see function lik_normal()); binomial likelihood (see function lik_binom); likelihood based on logF error distribution (see function lik_logF); mixture of normals likelihood (see function lik_normalmix); and Poisson likelihood (see function lik_pois).

weights a vector of weights for observations; use with optmethod = "w_mixEM"; this is currently beta-functionality.

pi_thresh a threshold below which to prune out mixture components before computing summaries (speeds up computation since empirically many components are usually assigned negligible weight). The current implementation still returns the full fitted distribution; this only affects the posterior summaries.
Details

The ash function provides a number of ways to perform Empirical Bayes shrinkage estimation and false discovery rate estimation. The main assumption is that the underlying distribution of effects is unimodal. Novice users are recommended to start with the examples provided below.

In the simplest case the inputs to ash are a vector of estimates (betahat) and their corresponding standard errors (sebetahat), and degrees of freedom (df). The method assumes that for some (unknown) "true" vector of effects beta, the statistic (betahat[j]-beta[j])/sebetahat[j] has a $t$ distribution on $df$ degrees of freedom. (The default of df=NULL assumes a normal distribution instead of a t.)

By default the method estimates the vector beta under the assumption that beta ~ g for a distribution g in G, where G is some unimodal family of distributions to be specified (see parameter `mixcompdist`). By default is to assume the mode is 0, and this is suitable for settings where you are interested in testing which beta[j] are non-zero. To estimate the mode see parameter `mode`.

As is standard in empirical Bayes methods, the fitting proceeds in two stages: i) estimate g by maximizing a (possibly penalized) likelihood; ii) compute the posterior distribution for each beta[j] | betahat[j],sebetahat[j] using the estimated g as the prior distribution.

A more general case allows that beta[j]/sebetahat[j]^alpha | sebetahat[j] ~ g.

Value

ash returns an object of `class"ash", a list with some or all of the following elements (determined by outputlevel)

- `fitted_g` fitted mixture
- `loglik` log P(D|fitted_g)
- `logLR` log[P(D|fitted_g)/P(D|beta==0)]
- `result` A dataframe whose columns are:
  - `NegativeProb` A vector of posterior probability that beta is negative.
  - `PositiveProb` A vector of posterior probability that beta is positive.
  - `lfsr` A vector of estimated local false sign rate.
  - `lfdr` A vector of estimated local false discovery rate.
  - `qvalue` A vector of q values.
  - `svalue` A vector of s values.
  - `PosteriorMean` A vector consisting the posterior mean of beta from the mixture.
  - `PosteriorSD` A vector consisting the corresponding posterior standard deviation.
- `call` a call in which all of the specified arguments are specified by their full names
- `data` a list containing details of the data and models used (mostly for internal use)
- `fit_details` a list containing results of mixture optimization, and matrix of component log-likelihoods used in this optimization

Functions

- `ash.workhorse`: Adaptive Shrinkage with full set of options.
See Also

ashci for computation of credible intervals after getting the ash object return by ash()

Examples

beta = c(rep(0,100),rnorm(100))
sebetahat = abs(rnorm(200,0,1))
betahat = rnorm(200,beta,sebetahat)
beta.ash = ash(betahat, sebetahat)
names(beta.ash)
head(beta.ash$result) # the main dataframe of results
head(get_pm(beta.ash)) # get_pm returns posterior mean
head(get_lfsr(beta.ash)) # get_lfsr returns the local false sign rate
graphics::plot(betahat,get_pm(beta.ash),xlim=c(-4,4),ylim=c(-4,4))

## Not run:
# Why is this example included here? -Peter
CIMatrix=ashci(beta.ash,level=0.95)
print(CIMatrix)
## End(Not run)

# Illustrating the non-zero mode feature.
betahat=betahat+5
beta.ash = ash(betahat, sebetahat)
graphics::plot(betahat,get_pm(beta.ash))
betan.ash=ash(betahat, sebetahat,mode=5)
graphics::plot(betahat,get_pm(betan.ash))
summary(betan.ash)

# Running ash with different error models
beta.ash1 = ash(betahat, sebetahat, lik = lik_normal())
beta.ash2 = ash(betahat, sebetahat, lik = lik_t(df=4))
e = rnorm(100)+log(rf(100,df1=10,df2=10)) # simulated data with log(F) error
e.ash = ash(e,1,lik=lik_logF(df1=10,df2=10))

# Specifying the output
beta.ash = ash(betahat, sebetahat, output = c("fitted_g","logLR","lfsr"))

#Running ash with a pre-specified g, rather than estimating it
beta = c(rep(0,100),rnorm(100))
sebetahat = abs(rnorm(200,0,1))
betahat = rnorm(200,beta,sebetahat)
ture_g = normalmix(c(0.5,0.5),c(0,0),c(0,1)) # define true g
## Passing this g into ash causes it to i) take the sd and the means
## for each component from this g, and ii) initialize pi to the value
## from this g.
beta.ash = ash(betahat, sebetahat,g=true_g,fixg=TRUE)

# running with weights
beta.ash = ash(betahat, sebeta, optmethod="w_mixEM",
               weights = c(rep(0.5,100),rep(1,100)))

# Different algorithms can be used to compute maximum-likelihood
# estimates of the mixture weights. Here, we illustrate use of the
# EM algorithm and the (default) SQP algorithm.
set.seed(1)
betahat <- c(8.115,9.027,9.289,10.097,9.463)
sebeta <- c(0.6157,0.4129,0.3197,0.3920,0.5496)
fit.em <- ash(betahat,sebeta,mixcompdist = "normal",optmethod = "mixEM")
fit.sqp <- ash(betahat,sebeta,mixcompdist = "normal",optmethod = "mixSQP")
range(fit.em$fitted$pi - fit.sqp$fitted$pi)

ashci

Credible Interval Computation for the ash object

Description

Given the ash object returned by the main function ash, this function computes a posterior credible
interval (CI) for each observation. The ash object must include a data component to use this function
(which it does by default).

Usage

ashci(
  a,
  level = 0.95,
  betaindex,
  lfsr_threshold = 1,
  tol = 0.001,
  trace = FALSE
)

Arguments

a  the fitted ash object
level  the level for the credible interval, (default=0.95)
betaindex  a vector consisting of locations of betahat where you would like to compute the credible interval
lfsr_threshold  a scalar, if specified then computes CIs only for observations more significant than that threshold.
tol  passed to uniroot; indicates desired accuracy.
trace  a logical variable denoting whether some of the intermediate results of iterations should be displayed to the user. Default is FALSE.
Details

Uses uniroot to find credible interval, one at a time for each observation. The computation cost is linear in number of observations.

Value

A matrix, with 2 columns, ith row giving CI for ith observation

Examples

```r
beta = c(rep(0,20),rnorm(20))
sebetahat = abs(rnorm(40,0,1))
betahat = rnorm(40,beta,sebetahat)
beta.ash = ash(betahat, sebetahat)
CImatrix=ashci(beta.ash,level=0.95)
CImatrix1=ashci(beta.ash,level=0.95,betaindex=c(1,2,5))
CImatrix2=ashci(beta.ash,level=0.95,lfsr_threshold=0.1)
```

ashr

Description

The main function in the ashr package is `ash`, which should be examined for more details. For simplicity only the most commonly-used options are documented under `ash`. For expert or interested users the documentation for function `ash.workhorse` provides documentation on all implemented options.

ash_pois

Description

Uses Empirical Bayes to fit the model

\[ y_j | \lambda_j \sim Poi(c_j \lambda_j) \]

with

\[ h(\lambda_j) \sim g() \]

where \( h \) is a specified link function (either "identity" or "log" are permitted).

Usage

```r
ash_pois(y, scale = 1, link = c("identity", "log"), ...)
```
Arguments

- `y` vector of Poisson observations.
- `scale` vector of scale factors for Poisson observations: the model is \( y[j] \sim \text{Pois}(\text{scale}[j] \times \lambda[j]) \).
- `link` string, either "identity" or "log", indicating the link function.
- `...` other parameters to be passed to ash

Details

The model is fit in two stages: i) estimate \( g \) by maximum likelihood (over the set of symmetric unimodal distributions) to give estimate \( \hat{g} \); ii) Compute posterior distributions for \( \lambda_j \) given \( y_j, \hat{g} \). Note that the link function \( h \) affects the prior assumptions (because, e.g., assuming a unimodal prior on \( \lambda \) is different from assuming unimodal on \( \log \lambda \)), but posterior quantities are always computed for the \( \lambda \) and *not* \( h(\lambda) \).

Examples

```r
beta = c(rep(0,50),rexp(50))
y = rpois(100,beta) # simulate Poisson observations
y.ash = ash_pois(y,scale=1)
```

Description

Return the log-likelihood of the data for a given \( g() \) prior

Usage

```r
calc_loglik(g, data)
```

Arguments

- `g` the fitted \( g \), or an ash object containing \( g \)
- `data` a data object, see set_data
calc_logLR  
*Compute loglikelihood ratio for data from ash fit*

**Description**  
Return the log-likelihood ratio of the data for a given g() prior

**Usage**  
```r
calc_logLR(g, data)
```

**Arguments**  
- **g**  
  the fitted g, or an ash object containing g  
- **data**  
  a data object, see set_data

---

calc_mixmean  
*Generic function of calculating the overall mean of the mixture*

**Description**  
Generic function of calculating the overall mean of the mixture

**Usage**  
```r
calc_mixmean(m)
```

**Arguments**  
- **m**  
  a mixture of k components generated by normalmix() or unimix() or igmix()

**Value**  
it returns scalar, the mean of the mixture distribution.
calc_mixsd

Generic function of calculating the overall standard deviation of the mixture

Description

Generic function of calculating the overall standard deviation of the mixture

Usage

calc_mixsd(m)

Arguments

m a mixture of k components generated by normalmix() or unimix() or igmix()

Value

it returns scalar

calc_null_loglik

Compute loglikelihood for data under null that all beta are 0

Description

Return the log-likelihood of the data betahat, with standard errors betahatsd, under the null that beta==0

Usage

calc_null_loglik(data)

Arguments

data a data object; see set_data
calc_null_vloglik  Compute vector of loglikelihood for data under null that all beta are 0

Description

Return the vector of log-likelihoods of the data points under the null

Usage

calc_null_vloglik(data)

Arguments

data  a data object; see set_data

calc_vloglik  Compute vector of loglikelihood for data from ash fit

Description

Return the vector of log-likelihoods of the data betahat, with standard errors betahatsd, for a given g() prior on beta, or an ash object containing that

Usage

calc_vloglik(g, data)

Arguments

g  the fitted g, or an ash object containing g

data  a data object, see set_data
calc_vlogLR

Compute vector of loglikelihood ratio for data from ash fit

Description
Return the vector of log-likelihood ratios of the data betahat, with standard errors betahatsd, for a
given g() prior on beta, or an ash object containing that, vs the null that g() is point mass on 0

Usage
calc_vlogLR(g, data)

Arguments

  g              the fitted g, or an ash object containing g
  data           a data object, see set_data

cdf.ash

cdf method for ash object

Description
Computed the cdf of the underlying fitted distribution

Usage
cdf.ash(a, x, lower.tail = TRUE)

Arguments

  a              the fitted ash object
  x              the vector of locations at which cdf is to be computed
  lower.tail    (default=TRUE) whether to compute the lower or upper tail

Details
None
cdf_conv

description
compute cdf of mixture m convoluted with error distribution either normal of sd (s) or student t with
df v at locations x

usage
cdf_conv(m, data)

arguments
m mixture distribution with k components
data details depend on the model

cdf_post

description
evaluate cdf of posterior distribution of beta at c. m is the prior on beta, a mixture; c is location of
evaluation assumption is betahat | beta ~ t_v(beta,sebetahat)

usage
cdf_post(m, c, data)

arguments
m mixture distribution with k components
c a scalar
data details depend on model

value
an n vector containing the cdf for beta_i at c

examples
beta = rnorm(100,0,1)
betahat= beta+rnorm(100,0,1)
sebetahat=rep(1,100)
ash.beta = ash(betahat,1,mixcomdist="normal")
cdf0 = cdf_post(ash.beta$fitted_g,0,set_data(betahat,sebetahat))
graphics::plot(cdf0,1-get_pp(ash.beta))
### compute_lfsr

**Function to compute the local false sign rate**

**Description**

Function to compute the local false sign rate

**Usage**

```
compute_lfsr(NegativeProb, ZeroProb)
```

**Arguments**

- **NegativeProb**: A vector of posterior probability that beta is negative.
- **ZeroProb**: A vector of posterior probability that beta is zero.

**Value**

The local false sign rate.

### comp_cdf

**Generic function of computing the cdf for each component**

**Description**

Generic function of computing the cdf for each component

**Usage**

```
comp_cdf(m, y, lower.tail = TRUE)
```

**Arguments**

- **m**: a mixture (eg of type normalmix or unimix)
- **y**: locations at which cdf to be computed
- **lower.tail**: boolean indicating whether to report lower tail

**Value**

it returns a vector of probabilities, with length equals to number of components in m
**Description**

compute the cdf of data for each component of mixture when convolved with error distribution

**Usage**

```r
comp_cdf_conv(m, data)
```

**Arguments**

- `m` mixture distribution with k components
- `data` details depend on the model

**Value**

a k by n matrix of cdfs

---

**Description**

returns cdf of convolution of each component of a normal mixture with N(0,s^2) at x. Note that convolution of two normals is normal, so it works that way

**Usage**

```r
## S3 method for class 'normalmix'
comp_cdf_conv(m, data)
```

**Arguments**

- `m` mixture distribution with k components
- `data` a list with components x and s to be interpreted as a normally-distributed observation and its standard error

**Value**

a k by n matrix
Description

cdf of convolution of each component of a unif mixture

Usage

```r
## S3 method for class 'unimix'
comp_cdf_conv(m, data)
```

Arguments

- `m` : a mixture of class unimix
- `data` : see set_data()

Value

a k by n matrix

Description

evaluate cdf of posterior distribution of beta at c. m is the prior on beta, a mixture; c is location of evaluation assumption is betahat | beta ~ t_v(beta,sebetahat)

Usage

```r
comp_cdf_post(m, c, data)
```

Arguments

- `m` : mixture distribution with k components
- `c` : a scalar
- `data` : details depend on model

Value

a k by n matrix
Examples

```r
beta = rnorm(100,0,1)
betahat = beta+rnorm(100,0,1)
sebetahat = rep(1,100)
ash.beta = ash(betahat,1,mixcompdist="normal")
comp_dens_conv(get_fitted_g(ash.beta),0,data=set_data(beta,sebetahat))
```

Description

Generic function of calculating the component densities of the mixture

Usage

```r
comp_dens(m, y, log = FALSE)
```

Arguments

- `m`: mixture of k components generated by normalmix() or unimix() or igmix()
- `y`: is an n-vector of location
- `log`: whether to use log-scale on densities

Value

A k by n matrix of densities

Description

compute the density of data for each component of mixture when convolved with error distribution

Usage

```r
comp_dens_conv(m, data, ...)
```

Arguments

- `m`: mixture distribution with k components
- `data`: details depend on the model
- `...`: other arguments

Value

A k by n matrix of densities
comp_dens_conv.normalmix

Description
returns density of convolution of each component of a normal mixture with $N(0,s^2)$ at x. Note that convolution of two normals is normal, so it works that way.

Usage

## S3 method for class 'normalmix'
comp_dens_conv(m, data, ...)

Arguments
- m: mixture distribution with k components
- data: a list with components x and s to be interpreted as a normally-distributed observation and its standard error
- ...: other arguments (unused)

Value
a k by n matrix

comp_dens_conv.unimix  density of convolution of each component of a unif mixture

Description
density of convolution of each component of a unif mixture

Usage

## S3 method for class 'unimix'
comp_dens_conv(m, data, ...)

Arguments
- m: a mixture of class unimix
- data: see set_data()
- ...: other arguments (unused)

Value
a k by n matrix
comp_mean

Generic function of calculating the first moment of components of the mixture

Description

Generic function of calculating the first moment of components of the mixture

Usage

comp_mean(m)

Arguments

m

a mixture of k components generated by normalmix() or unimix() or igmix()

Value

it returns a vector of means.

Description

returns mean of the normal mixture

Usage

## S3 method for class 'normalmix'
comp_mean(m)

Arguments

m

a normal mixture distribution with k components

Value

a vector of length k
Description

Returns mean of the truncated-normal mixture.

Usage

```r
## S3 method for class 'tnormalmix'
comp_mean(m)
```

Arguments

- `m` A truncated normal mixture distribution with k components.

Value

A vector of length k.

Description

Generic function of calculating the second moment of components of the mixture

Usage

```r
comp_mean2(m)
```

Arguments

- `m` a mixture of k components generated by normalmix() or unimix() or igmix()

Value

it returns a vector of second moments.
comp_postmean

Description
output posterior mean for beta for each component of prior mixture m, given data

Usage
comp_postmean(m, data)

Arguments
m mixture distribution with k components
data details depend on the model

comp_postmean2

Description
output posterior mean-squared value given prior mixture m and data

Usage
comp_postmean2(m, data)

Arguments
m mixture distribution with k components
data details depend on the model
**comp_postprob**

**Description**
compute the posterior prob that each observation came from each component of the mixture m. Output a k by n vector of probabilities computed by weighting the component densities by pi and then normalizing.

**Usage**
```
comp_postprob(m, data)
```

**Arguments**
- `m`: mixture distribution with k components
- `data`: details depend on the model

---

**comp_postsd**

**Description**
output posterior sd for beta for each component of prior mixture m, given data.

**Usage**
```
comp_postsd(m, data)
```

**Arguments**
- `m`: mixture distribution with k components
- `data`: details depend on the model

**Examples**
```
beta = rnorm(100,0,1)
betahat = beta + rnorm(100,0,1)
ash.beta = ash(betahat,1,mixcompdist="normal")
data = set_data(betahat,rep(1,100))
comp_postmean(get_fitted_g(ash.beta),data)
comp_postsd(get_fitted_g(ash.beta),data)
comp_postprob(get_fitted_g(ash.beta),data)
```
comp_sd

Generic function to extract the standard deviations of components of the mixture

Description

Generic function to extract the standard deviations of components of the mixture

Usage

comp_sd(m)

Arguments

m a mixture of k components generated by normalmix() or unimix() or igmix()

Value

it returns a vector of standard deviations

comp_sd.normalmix

returns sds of the normal mixture

Usage

## S3 method for class 'normalmix'
comp_sd(m)

Arguments

m a normal mixture distribution with k components

Value

a vector of length k
Description

Returns standard deviations of the truncated normal mixture.

Usage

```r
## S3 method for class 'tnormalmix'
comp_sd(m)
```

Arguments

- `m` A truncated normal mixture distribution with k components.

Value

A vector of length k.

---

**cxxMixSquarem**

Brief description of function.

Description

Explain here what this function does.

Usage

```r
cxxMixSquarem(matrix_lik, prior, pi_init, control)
```

Arguments

- `matrix_lik` Description of argument goes here.
- `prior` Description of argument goes here.
- `pi_init` Description of argument goes here.
- `control` Description of argument goes here.
dens  

*Find density at y, a generic function*

---

**Description**

Find density at y, a generic function

**Usage**

dens(x, y)

**Arguments**

- **x**: A mixture of k components generated by `normalmix` or `unimix`.
- **y**: An n-vector of the location.

---

dens_conv  

dens_conv

---

**Description**

compute density of mixture m convoluted with normal of sd (s) or student t with df v at locations x

**Usage**

dens_conv(m, data)

**Arguments**

- **m**: mixture distribution with k components
- **data**: details depend on the model
**dlogf**

*The log-F distribution*

**Description**

Density function for the log-F distribution with \( df_1 \) and \( df_2 \) degrees of freedom (and optional non-centrality parameter \( ncp \)).

**Usage**

\[
dlogf(x, df_1, df_2, ncp, log = FALSE)
\]

**Arguments**

- \( x \) vector of quantiles
- \( df_1 \) degrees of freedom
- \( df_2 \) degrees of freedom
- \( ncp \) non-centrality parameter. If omitted the central F is assumed.
- \( log \) logical; if TRUE, probabilities \( p \) are given as \( \log(p) \).

**Value**

The density function.

**estimate_mixprop**

*Estimate mixture proportions of a mixture \( g \) given noisy (error-prone) data from that mixture.*

**Description**

Estimate mixture proportions of a mixture \( g \) given noisy (error-prone) data from that mixture.

**Usage**

\[
estimate_mixprop(
  data,
  g,
  prior,
  optmethod = c("mixSQP", "mixEM", "mixVBEM", "cxxMixSquarem", "mixIP", "w_mixEM"),
  control,
  weights = NULL
)
\]
Arguments

data list to be passed to log_comp_dens_conv; details depend on model

g an object representing a mixture distribution (eg normalmix for mixture of normals; unimix for mixture of uniforms). The component parameters of g (eg the means and variances) specify the components whose mixture proportions are to be estimated. The mixture proportions of g are the parameters to be estimated; the values passed in may be used to initialize the optimization (depending on the optmethod used)

prior numeric vector indicating parameters of "Dirichlet prior" on mixture proportions

optmethod name of function to use to do optimization

control list of control parameters to be passed to optmethod, typically affecting things like convergence tolerance

weights vector of weights (for use with w_mixEM; in beta)

Details

This is used by the ash function. Most users won’t need to call this directly, but is exported for use by some other related packages.

Value

list, including the final loglikelihood, the null loglikelihood, an n by k likelihood matrix with (j,k)th element equal to \( f_k(x_j) \), the fit and results of optmethod

Description

Produce function to compute expectation of truncated error distribution from log cdf and log pdf (using numerical integration)

Usage

gen_etruncFUN(lcdfFUN, lpdfFUN)

Arguments

lcdfFUN the log cdfFUN of the error distribution

lpdfFUN the log pdfFUN of the error distribution
get_density

Density method for ash object

Description
Return the density of the underlying fitted distribution

Usage
get_density(a, x)

Arguments
a  
the fitted ash object
x  
the vector of locations at which density is to be computed

Details
None

get_lfsr

Return lfsr from an ash object

Description
These functions simply return elements of an ash object, generally without doing any calculations. (So if the value was not computed during the original call to ash, eg because of how outputlevel was set in the call, then NULL will be returned.) Accessing elements in this way rather than directly from the ash object will help ensure compatibility moving forward (e.g. if the internal structure of the ash object changes during software development.)

Usage
get_lfsr(x)
get_lfdr(a)
get_svalue(a)
get_qvalue(a)
get_pvalue(a)
get_psd(a)
get_lfsr(a)
get_pp(a)
get_np(a)
get_loglik(a)
get_logLR(a)
get_fitted_g(a)
get_pi0(a)

Arguments

x an ash fit (e.g. from running ash)

a an ash fit (e.g. from running ash)

Value

A vector (ash) of local false sign rates

Functions

• get_lfsr: local false sign rate
• get_lfdr: local false discovery rate
• get_svalue: svalue
• get_qvalue: qvalue
• get_pm: posterior mean
• get_psd: posterior standard deviation
• get_pp: positive probability
• get_np: negative probability
• get_loglik: log-likelihood
• get_logLR: log-likelihood ratio
• get_fitted_g: fitted g mixture
• get_pi0: pi0, the proportion of nulls
get_post_sample

Sample from posterior

Description

Returns random samples from the posterior distribution for each observation in an ash object. A matrix is returned, with columns corresponding to observations and rows corresponding to samples.

Usage

get_post_sample(a, nsamp)

Arguments

a the fitted ash object
nsamp number of samples to return (for each observation)

Examples

beta = rnorm(100,0,1)
betahat = beta + rnorm(100,0,1)
ash.beta = ash(betahat,1,mixcompdist="normal")
post.beta = get_post_sample(ash.beta,1000)

igmix

Constructor for igmix class

Description

Creates an object of class igmix (finite mixture of univariate inverse-gammas)

Usage

igmix(pi, alpha, beta)

Arguments

pi vector of mixture proportions
alpha vector of shape parameters
beta vector of rate parameters

Details

None
lik_binom

Value
an object of class igmix

Examples

\texttt{igmix(c(0.5,0.5),c(1,1),c(1,2))}

\begin{verbatim}
lik_binom

Description

Creates a likelihood object for ash for use with Binomial error distribution

Usage

\texttt{lik_binom(y, n, link = c("identity", "logit"))}

Arguments

\begin{itemize}
  \item \texttt{y} Binomial observations
  \item \texttt{n} Binomial number of trials
  \item \texttt{link} Link function. The "identity" link directly puts unimodal prior on Binomial success probabilities \( p \), and "logit" link puts unimodal prior on \texttt{logit}(p).
\end{itemize}

Details

Suppose we have Binomial observations \( y \) where \( y_i \sim Bin(n_i, p_i) \). We either put an unimodal prior \( g \) on the success probabilities \( p_i \sim g \) (by specifying \texttt{link="identity"}) or on the logit success probabilities \( \texttt{logit}(p_i) \sim g \) (by specifying \texttt{link="logit"}). Either way, ASH with this Binomial likelihood function will compute the posterior mean of the success probabilities \( p_i \).

Examples

\begin{verbatim}
p = rbeta(100,2,2) # prior mode: 0.5
n = rpois(100,10)
y = rbinom(100,n,p) # simulate Binomial observations
ash(rep(0,length(y)),1,lik=lik_binom(y,n))
\end{verbatim}
lik_logF

Likelihood object for logF error distribution

Description

Creates a likelihood object for ash for use with logF error distribution

Usage

lik_logF(df1, df2)

Arguments

df1  first degree of freedom parameter of F distribution

df2  second degree of freedom parameter of F distribution

Examples

e = rnorm(100) + log(rf(100,df1=10,df2=10))  # simulate some data with log(F) error
ash(e,1,lik=lik_logF(df1=10,df2=10))

lik_normal

Likelihood object for normal error distribution

Description

Creates a likelihood object for ash for use with normal error distribution

Usage

lik_normal()

Examples

z = rnorm(100) + rnorm(100)  # simulate some data with normal error
ash(z,1,lik=lik_normal())
lik_normalmix  
*Likelihood object for normal mixture error distribution*

**Description**

Creates a likelihood object for ash for use with normal mixture error distribution

**Usage**

```r
lik_normalmix(pilik, sdlik)
```

**Arguments**

- `pilik`:
  - A `k` vector of mixture proportions (`k` is the number of mixture components), or a `n*k` matrix that the `j`th row is mixture proportions for `betahat_j`.

- `sdlik`:
  - A `k` vector of component-wise standard deviations, or an `n*k` matrix that the `j`th row is component-wise standard deviations for `betahat_j`.

**Examples**

```r
e = rnorm(100,0,.8)  
e[seq(1,100,by=2)] = rnorm(50,0,1.5)  
# generate e~0.5*N(0,0.8^2)+0.5*N(0,1.5^2)  
betahat = rnorm(100)+e  
ash(betahat, 1, lik=lik_normalmix(c(0.5,0.5),c(0.8,1.5)))
```

lik_pois  
*Likelihood object for Poisson error distribution*

**Description**

Creates a likelihood object for ash for use with Poisson error distribution

**Usage**

```r
lik_pois(y, scale = 1, link = c("identity", "log"))
```

**Arguments**

- `y`:
  - Poisson observations.

- `scale`:
  - Scale factor for Poisson observations: `y~Pois(scale*lambda)`.

- `link`:
  - Link function. The "identity" link directly puts unimodal prior on Poisson intensities `lambda`, and "log" link puts unimodal prior on `log(lambda)`. 

```r

e = rnorm(100,0,.8)

e[seq(1,100,by=2)] = rnorm(50,0,1.5)  
# generate e~0.5*N(0,0.8^2)+0.5*N(0,1.5^2)  
betahat = rnorm(100)+e  
ash(betahat, 1, lik=lik_normalmix(c(0.5,0.5),c(0.8,1.5)))
```
Details
Suppose we have Poisson observations $y$ where $y_i \sim \text{Poisson}(c_i \lambda_i)$. We either put an unimodal prior $g$ on the (scaled) intensities $\lambda_i \sim g$ (by specifying link="identity") or on the log intensities $\log(\lambda_i) \sim g$ (by specifying link="log"). Either way, ASH with this Poisson likelihood function will compute the posterior mean of the intensities $\lambda_i$.

Examples

```r
beta = c(rnorm(100,50,5)) # prior mode: 50
y = rpois(100,beta) # simulate Poisson observations
ash(rep(0,length(y)),1,lik=lik_pois(y))
```

lik_t

Likelihood object for t error distribution

Description
Creates a likelihood object for ash for use with t error distribution

Usage

```r
lik_t(df)
```

Arguments

- `df` degree of freedom parameter of t distribution

Examples

```r
z = rnorm(100) + rt(100,df=4) # simulate some data with t error
ash(z,1,lik=lik_t(df=4))
```

loglik_conv

Description
find log likelihood of data using convolution of mixture with error distribution

Usage

```r
loglik_conv(m, data)
```

Arguments

- `m` mixture distribution with k components
- `data` details depend on the model
Description

The default version of `loglik_conv`.

Usage

`## Default S3 method:
loglik_conv(m, data)`

Arguments

- `m` mixture distribution with k components
- `data` data whose details depend on model

Value

A k by n matrix of log densities
Description
returns log-density of convolution of each component of a normal mixture with N(0,s^2) or s*t(v) at x. Note that convolution of two normals is normal, so it works that way

Usage
## S3 method for class 'normalmix'
log_comp_dens_conv(m, data)

Arguments
m mixture distribution with k components
data a list with components x and s to be interpreted as a normally-distributed observation and its standard error

Value
a k by n matrix

Description
log density of convolution of each component of a unif mixture

Usage
## S3 method for class 'unimix'
log_comp_dens_conv(m, data)

Arguments
m a mixture of class unimix
data see set_data()

Value
a k by n matrix of densities
mixcdf

Description

Returns cdf for a mixture (generic function)

Usage

mixcdf(x, y, lower.tail = TRUE)

Arguments

x a mixture (eg of type normalmix or unimix)
y locations at which cdf to be computed
lower.tail boolean indicating whether to report lower tail

Details

None

Value

an object of class normalmix

Examples

mixcdf(normalmix(c(0.5,0.5),c(0,0),c(1,2)),seq(-4,4,length=100))

mixcdf.default

Description

The default version of mixcdf.

Usage

## Default S3 method:
mixcdf(x, y, lower.tail = TRUE)

Arguments

x a mixture (eg of type normalmix or unimix)
y locations at which cdf to be computed
lower.tail boolean indicating whether to report lower tail
mixEM

Estimate mixture proportions of a mixture model by EM algorithm

Description

Given the individual component likelihoods for a mixture model, estimates the mixture proportions by an EM algorithm.

Usage

mixEM(matrix_lik, prior, pi_init = NULL, control = list())

Arguments

matrix_lik, a n by k matrix with (j,k)th element equal to \( f_k(x_j) \).
prior, a k vector of the parameters of the Dirichlet prior on \( \pi \). Recommended to be rep(1,k)
pi_init, the initial value of \( \pi \) to use. If not specified defaults to (1/k,...,1/k).
control A list of control parameters for the SQUAREM algorithm, default value is set to be control.default=list(K = 1, method=3, square=TRUE, step.min0=1, step.max0=1, mstep=4, kr=1, objfn.inc=1,tol=1.e-07, maxiter=5000, trace=FALSE).

Details

Fits a k component mixture model

\[
 f(x|\pi) = \sum_k \pi_k f_k(x)
\]

to independent and identically distributed data \( x_1,\ldots,x_n \). Estimates mixture proportions \( \pi \) by maximum likelihood, or by maximum a posteriori (MAP) estimation for a Dirichlet prior on \( \pi \) (if a prior is specified). Uses the SQUAREM package to accelerate convergence of EM. Used by the ash main function; there is no need for a user to call this function separately, but it is exported for convenience.

Value

A list, including the estimates (pihat), the log likelihood for each iteration (B) and a flag to indicate convergence.
Description

Given the individual component likelihoods for a mixture model, estimates the mixture proportions.

Usage

mixIP(matrix_lik, prior, pi_init = NULL, control = list(), weights = NULL)

Arguments

- matrix_lik, a n by k matrix with (j,k)th element equal to $f_k(x_j)$.
- prior, a k vector of the parameters of the Dirichlet prior on $\pi$. Recommended to be rep(1,k).
- pi_init, the initial value of $\pi$ to use. If not specified defaults to (1/k,...,1/k).
- control A list of control parameters to be passed to REBayes::KWDual
- weights weights to be assigned to the observations (an n vector)

Details

Optimizes $L(\pi) = \sum_j w_j \log(\sum_k p_k f_{jk}) + h(\pi)$

subject to $\pi_k$ non-negative and $\sum_k \pi_k = 1$. Here

$h(\pi)$

is a penalty function $h(\pi) = \sum_k (prior_k-1) \log \pi_k$. Calls REBayes::KWDual in the REBayes package, which is in turn a wrapper to the mosek convex optimization software. So REBayes must be installed to use this. Used by the ash main function; there is no need for a user to call this function separately, but it is exported for convenience.

Value

A list, including the estimates (pihat), the log likelihood for each iteration (B) and a flag to indicate convergence
mixmean2

Generic function of calculating the overall second moment of the mixture

Description
Generic function of calculating the overall second moment of the mixture

Usage
mixmean2(m)

Arguments
m a mixture of k components generated by normalmix() or unimix() or igmix()

Value
it returns scalar

mixprop
Generic function of extracting the mixture proportions

Description
Generic function of extracting the mixture proportions

Usage
mixprop(m)

Arguments
m a mixture of k components generated by normalmix() or unimix() or igmix()

Value
it returns a vector of component probabilities, summing up to 1.
mixSQP

Estimate mixture proportions of a mixture model using mix-SQP algorithm.

Description

Estimate mixture proportions of a mixture model using mix-SQP algorithm.

Usage

mixSQP(matrix_lik, prior, pi_init = NULL, control = list(), weights = NULL)

Arguments

matrix_lik  A matrix containing the conditional likelihood values, possibly normalized.
prior       A vector of the parameters of the Dirichlet prior on the mixture weights.
pi_init     The initial estimate of the mixture weights.
control     A list of settings for the mix-SQP optimization algorithm; see mixsqp for details.
weights     The weights to be assigned to the observations. Must be a vector of length equal to
             the number of rows of matrix_lik. If weights = NULL, all observations are
             assigned the same weight.

Value

A list object including the estimates (pihat) and a flag (control) indicating convergence success
or failure.

mixVBEM

Estimate posterior distribution on mixture proportions of a mixture model by a Variational Bayes EM algorithm

Description

Given the individual component likelihoods for a mixture model, estimates the posterior on the
mixture proportions by an VBEM algorithm. Used by the ash main function; there is no need for a
user to call this function separately, but it is exported for convenience.

Usage

mixVBEM(matrix_lik, prior, pi_init = NULL, control = list())
my_e2truncbeta

Arguments

matrix_lik a n by k matrix with (j,k)th element equal to $f_k(x_j)$.
prior a k vector of the parameters of the Dirichlet prior on $\pi$. Recommended to be rep(1,k)
pi_init the initial value of the posterior parameters. If not specified defaults to the prior parameters.
control A list of control parameters for the SQUAREM algorithm, default value is set to be control.default=list(K = 1, method=3, square=TRUE, step.min0=1, step.max0=1, mstep=4, k=1, objfn.inc=1,tol=1.e-07, maxiter=5000, trace=FALSE).

Details

Fits a k component mixture model

$$f(x|\pi) = \sum_k \pi_k f_k(x)$$

to independent and identically distributed data $x_1, \ldots, x_n$. Estimates posterior on mixture proportions $\pi$ by Variational Bayes, with a Dirichlet prior on $\pi$. Algorithm adapted from Bishop (2009), Pattern Recognition and Machine Learning, Chapter 10.

Value

A list, whose components include point estimates (pihat), the parameters of the fitted posterior on $\pi$ (pipost), the bound on the log likelihood for each iteration (B) and a flag to indicate convergence (converged).

my_e2truncbeta second moment of truncated Beta distribution

Description

Compute second moment of the truncated Beta.

Usage

my_e2truncbeta(a, b, alpha, beta)

Arguments

a left limit of distribution
b right limit of distribution
alpha, beta shape parameters of Beta distribution
my_e2truncgamma  second moment of truncated gamma distribution

Description
Compute second moment of the truncated gamma.

Usage
my_e2truncgamma(a, b, shape, rate)

Arguments
a  left limit of distribution
b  right limit of distribution
shape  shape of gamma distribution
rate  rate of gamma distribution

my_e2truncnorm  Expected Squared Value of Truncated Normal

Description
Computes the expected squared values of truncated normal distributions with parameters a, b, mean, and sd. Arguments can be scalars, vectors, or matrices. Arguments of shorter length will be recycled according to the usual recycling rules, but a and b must have the same length. Missing values are accepted for all arguments.

Usage
my_e2truncnorm(a, b, mean = 0, sd = 1)

Arguments
a  The lower limit for the support of the truncated normal. Can be -Inf.
b  The upper limit for the support. Can be Inf. a and b must have the same length, and each element of a should be less than or equal to the corresponding element of b.
mean  The mean of the untruncated normal.
sd  The standard deviation of the untruncated normal. Standard deviations of zero are interpreted as numerically (rather than exactly) zero, so that the square of the untruncated mean is returned if it lies within [a, b] and the square of the nearer of a and b is returned otherwise.
Value

The expected squared values of truncated normal distributions with parameters \( a, b, \text{mean}, \) and \( \text{sd} \). If any of the arguments is a matrix, then a matrix will be returned.

See Also

my_etruncnorm, my_vtruncnorm

my_e2trunct  my_e2trunct

Description

Compute second moment of the truncated t. Uses results from O'Hagan, Biometrika, 1973

Usage

my_e2trunct(a, b, df)

Arguments

a  left limit of distribution
b  right limit of distribution
df  degree of freedom of error distribution

my_etruncbeta  mean of truncated Beta distribution

Description

Compute mean of the truncated Beta.

Usage

my_etruncbeta(a, b, alpha, beta)

Arguments

a  left limit of distribution
b  right limit of distribution
alpha, beta  shape parameters of Beta distribution
my_etruncgamma

Description

Compute mean of the truncated gamma.

Usage

my_etruncgamma(a, b, shape, rate)

Arguments

a \hspace{1cm} \text{left limit of distribution}
\hline
b \hspace{1cm} \text{right limit of distribution}
\hline
shape \hspace{1cm} \text{shape of gamma distribution}
\hline
rate \hspace{1cm} \text{rate of gamma distribution}

my_etrunclogf

Description

Compute expectation of truncated log-F distribution.

Usage

my_etrunclogf(a, b, df1, df2)

Arguments

a \hspace{1cm} \text{Left limit of distribution.}
\hline
b \hspace{1cm} \text{Right limit of distribution.}
\hline
df1, df2 \hspace{1cm} \text{degrees of freedom}
my_etruncnorm

Expected Value of Truncated Normal

Description
Computes the means of truncated normal distributions with parameters \( a \), \( b \), \( \text{mean} \), and \( \text{sd} \). Arguments can be scalars, vectors, or matrices. Arguments of shorter length will be recycled according to the usual recycling rules, but \( a \) and \( b \) must have the same length. Missing values are accepted for all arguments.

Usage
my_etruncnorm(a, b, mean = 0, sd = 1)

Arguments
- \( a \): The lower limit for the support of the truncated normal. Can be \(-\infty\).
- \( b \): The upper limit for the support. Can be \( \infty \). \( a \) and \( b \) must have the same length, and each element of \( a \) should be less than or equal to the corresponding element of \( b \).
- \( \text{mean} \): The mean of the untruncated normal.
- \( \text{sd} \): The standard deviation of the untruncated normal. Standard deviations of zero are interpreted as numerically (rather than exactly) zero, so that the untruncated mean is returned if it lies within \([a, b]\) and the nearer of \( a \) and \( b \) is returned otherwise.

Value
The expected values of truncated normal distributions with parameters \( a \), \( b \), \( \text{mean} \), and \( \text{sd} \). If any of the arguments is a matrix, then a matrix will be returned.

See Also
my_e2truncnorm, my_vtruncnorm

my_etrunct

my_etrunct

Description
Compute second moment of the truncated t. Uses results from O’Hagan, Biometrika, 1973

Usage
my_etrunct(a, b, df)
Arguments

- **a**: left limit of distribution
- **b**: right limit of distribution
- **df**: degree of freedom of error distribution

---

**my_vtruncnorm**  
*Variance of Truncated Normal*

Description

Computes the variance of truncated normal distributions with parameters a, b, mean, and sd. Arguments can be scalars, vectors, or matrices. Arguments of shorter length will be recycled according to the usual recycling rules, but a and b must have the same length. Missing values are accepted for all arguments.

Usage

```r
my_vtruncnorm(a, b, mean = 0, sd = 1)
```

Arguments

- **a**: The lower limit for the support of the truncated normal. Can be -Inf.
- **b**: The upper limit for the support. Can be Inf. a and b must have the same length, and each element of a should be less than or equal to the corresponding element of b.
- **mean**: The mean of the untruncated normal.
- **sd**: The standard deviation of the untruncated normal.

Value

The variance of truncated normal distributions with parameters a, b, mean, and sd. If any of the arguments is a matrix, then a matrix will be returned.

See Also

- `my_etruncnorm`
- `my_e2truncnorm`
Description

ncomp

Usage

ncomp(m)

Arguments

m a mixture of k components generated by normalmix() or unimix() or igmix()

Description

The default version of \texttt{ncomp}.

Usage

### Default S3 method:
ncomp(m)

Arguments

m a mixture of k components generated by normalmix() or unimix() or igmix()

Constructor for \texttt{normalmix} class

Description

Creates an object of class \texttt{normalmix} (finite mixture of univariate normals)

Usage

normalmix(pi, mean, sd)
Arguments

- **pi**: vector of mixture proportions
- **mean**: vector of means
- **sd**: vector of standard deviations

Details

None

Value

an object of class normalmix

Examples

```r
cnormalmix(c(0.5,0.5),c(0,0),c(1,2))
```

Description

“parallel” vector version of `cdf_post` where `c` is a vector, of same length as `betahat` and `sebetahat`

Usage

```r
cpdf_post(m, c, data)
```

Arguments

- **m**: mixture distribution with k components
- **c**: a numeric vector with n elements
- **data**: depends on context

Value

an n vector, whose ith element is the cdf for beta_i at c_i

Examples

```r
beta = rnorm(100,0,1)
betahat= beta+rnorm(100,0,1)
sebetahat=rep(1,100)
ash.beta = ash(betahat,1,mixcompdist="normal")
c = pcdf_post(get_fitted_g(ash.beta),beta,set_data(betahat,sebetahat))
```
The `plogf` function provides the distribution function for the log-F distribution with \(df1\) and \(df2\) degrees of freedom (and optional non-centrality parameter \(ncp\)).

**Usage**

```r
plogf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
```

**Arguments**

- `q`: vector of quantiles
- `df1, df2`: degrees of freedom
- `ncp`: non-centrality parameter. If omitted the central F is assumed.
- `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 \(\log(p)\).

**Value**

The distribution function.

The `plot.ash` method is used to plot the cumulative distribution function (CDF) of the underlying fitted distribution.

**Plot method for ash object**

**Description**

Plot the cdf of the underlying fitted distribution.

**Usage**

```r
# S3 method for class 'ash'
plot(x, ..., xmin, xmax)
```

**Arguments**

- `x`: the fitted ash object
- `...`: Arguments to be passed to methods, such as graphical parameters (see `plot`)
- `xmin`: xlim lower range, default is the lowest value of betaHat
- `xmax`: xlim upper range, default is the highest value of betaHat

**Details**

None
**plot_diagnostic**

*Diagnostic plots for ash object*

### Description

Generate several plots to diagnose the fitness of ASH on the data

### Usage

```r
plot_diagnostic(
  x,  
  plot.it = TRUE,  
  sebetahat.tol = 0.001,  
  plot.hist,  
  xmin,  
  xmax,  
  breaks = "Sturges",  
  alpha = 0.01,  
  pch = 19,  
  cex = 0.25
)
```

### Arguments

- **x**  
  the fitted ash object
- **plot.it**  
  logical. whether to plot the diagnostic result
- **sebetahat.tol**  
  tolerance to test the equality of betahat
- **plot.hist**  
  logical. whether to plot the histogram of betahat when sebetahat is not constant
- **xmin, xmax**  
  range of the histogram of betahat to be plotted
- **breaks**  
  histograms parameter (see `hist`)
- **alpha**  
  error level for the de-trended diagnostic plot
- **pch, cex**  
  plot parameters for dots

### Details

None.
pm_on_zero

Generic function to extract which components of mixture are point mass on 0

Description
Generic function to extract which components of mixture are point mass on 0

Usage
pm_on_zero(m)

Arguments
m a mixture of k components generated by normalmix() or unimix() or igmix()

Value
a boolean vector indicating which components are point mass on 0

posterior_dist Compute Posterior

Description
Return the posterior on beta given a prior (g) that is a mixture of normals (class normalmix) and observation \( \text{betahat} \ N(\beta, \text{sebetahat}) \)

Usage
posterior_dist(g, betahat, sebetahat)

Arguments
g a normalmix with components indicating the prior; works only if g has means 0
betahat (n vector of observations)
sebetahat (n vector of standard errors/deviations of observations)

Details
This can be used to obt

Value
A list, (pi1,mu1,sigma1) whose components are each k by n matrices where k is number of mixture components in g, n is number of observations in betahat
postmean

**Description**
postmean

**Usage**
```
postmean(m, data)
```

**Arguments**
- `m`: mixture distribution with k components
- `data`: details depend on the model

---

postmean2

**Description**
output posterior mean-squared value given prior mixture m and data

**Usage**
```
postmean2(m, data)
```

**Arguments**
- `m`: mixture distribution with k components
- `data`: details depend on the model

---

postsd

**Description**
output posterior sd given prior mixture m and data

**Usage**
```
postsd(m, data)
```

**Arguments**
- `m`: mixture distribution with k components
- `data`: details depend on the model
Description
returns random samples from the posterior, given a prior distribution \( m \) and \( n \) observed datapoints.

Usage
\[
\text{post\_sample}(m, \text{data}, \text{nsamp})
\]

Arguments
\[
\begin{align*}
m & \quad \text{prior distribution (eg of type \texttt{normalmix})} \\
data & \quad \text{a list with components \( x \) and \( s \), each vectors of length \( n \), to be interpreted as a normally-distributed observations and corresponding standard errors} \\
\text{nsamp} & \quad \text{number of random samples to return for each observation}
\end{align*}
\]

Details
exported, but mostly users will want to use \texttt{get\_post\_sample}

Value
an \( \text{nsamp} \) by \( n \) matrix

Description
returns random samples from the posterior, given a prior distribution \( m \) and \( n \) observed datapoints.

Usage
\[
## \text{S3 method for class}'normalmix' \\
\text{post\_sample}(m, \text{data}, \text{nsamp})
\]

Arguments
\[
\begin{align*}
m & \quad \text{mixture distribution with k components} \\
data & \quad \text{a list with components \( x \) and \( s \) to be interpreted as a normally-distributed observation and its standard error} \\
\text{nsamp} & \quad \text{number of samples to return for each observation}
\end{align*}
\]
### post_sample.unimix

**Description**
returns random samples from the posterior, given a prior distribution \( m \) and \( n \) observed datapoints.

**Usage**
```r
## S3 method for class 'unimix'
post_sample(m, data, nsamp)
```

**Arguments**
- `m`: mixture distribution with \( k \) components
- `data`: a list with components `x` and `s` to be interpreted as a normally-distributed observation and its standard error
- `nsamp`: number of samples to return for each observation

**Value**
a \( n \) by \( nsamp \) matrix

### print.ash

**Print method for ash object**

**Description**
Print the fitted distribution of beta values in the EB hierarchical model

**Usage**
```r
## S3 method for class 'ash'
print(x, ...)
```

**Arguments**
- `x`: the fitted ash object
- `...`: not used, included for consistency as an S3 generic/method.

**Details**
None
Description

prunes out mixture components with low weight

Usage

prune(m, thresh = 1e-10)

Arguments

m, What is this argument?
thresh the threshold below which components are removed

qval.from.lfdr Function to compute q values from local false discovery rates

Description

Computes q values from a vector of local fdr estimates

Usage

qval.from.lfdr(lfdr)

Arguments

lfdr, a vector of local fdr estimates

Details

The q value for a given lfdr is an estimate of the (tail) False Discovery Rate for all findings with a smaller lfdr, and is found by the average of the lfdr for all more significant findings. See Storey (2003), Annals of Statistics, for definition of q value.

Value

vector of q values
**set_data**

*Takes raw data and sets up data object for use by ash*

**Description**

Takes raw data and sets up data object for use by ash

**Usage**

```r
set_data(betahat, sebetahat, lik = NULL, alpha = 0)
```

**Arguments**

- `betahat`: vector of betahats
- `sebetahat`: vector of standard errors
- `lik`: a likelihood (see e.g., `lik_normal()`)
- `alpha`: specifies value of alpha to use (model is for `betahat/sebetahat^alpha | sebetahat`)

**Details**

The data object stores both the data, and details of the model to be used for the data. For example, in the generalized version of ash the cdf and pdf of the likelihood are stored here.

**Value**

data object (list)

---

**summary.ash**

*Summary method for ash object*

**Description**

Print summary of fitted ash object

**Usage**

```r
## S3 method for class 'ash'
summary(object, ...)
```

**Arguments**

- `object`: the fitted ash object
- `...`: not used, included for consistency as an S3 generic/method.
**tnormalmix**

*Constructor for tnormalmix class*

**Description**

Creates an object of class tnormalmix (finite mixture of truncated univariate normals).

**Usage**

```r
tnormalmix(pi, mean, sd, a, b)
```

**Arguments**

- **pi**: Cector of mixture proportions (length k say).
- **mean**: Vector of means (length k).
- **sd**: Vector of standard deviations (length k).
- **a**: Vector of left truncation points of each component (length k).
- **b**: Vector of right truncation points of each component (length k).

**Value**

An object of class “tnormalmix”.

**Examples**

```r
tnormalmix(c(0.5,0.5),c(0,0),c(1,2),c(-10,0),c(0,10))
```

**unimix**

*Constructor for unimix class*

**Description**

Creates an object of class unimix (finite mixture of univariate uniforms)

**Usage**

```r
unimix(pi, a, b)
```
Arguments
   pi       vector of mixture proportions
   a       vector of left hand ends of uniforms
   b       vector of right hand ends of uniforms

Details
   None

Value
   an object of class unimix

Examples
   unimix(c(0.5,0.5),c(0,0),c(1,2))

vcdf_post

vcdf_post

Description
   vectorized version of cdf_post

Usage
   vcdf_post(m, c, data)

Arguments
   m       mixture distribution with k components
   c       a numeric vector
   data    depends on context

Value
   an n vector containing the cdf for beta_i at c

Examples
   beta = rnorm(100,0,1)
   betahat = beta+rnorm(100,0,1)
   sebetahat = rep(1,100)
   ash.beta = ash(betahat,1,mixcompdist="normal")
   c = vcdf_post(get_fitted_g(ash.beta),seq(-5,5,length=1000),data = set_data(betahat,sebetahat))
Estimate mixture proportions of a mixture model by EM algorithm (weighted version)

Description

Given the individual component likelihoods for a mixture model, and a set of weights, estimates the mixture proportions by an EM algorithm.

Usage

```r
w_mixEM(matrix_lik, prior, pi_init = NULL, weights = NULL, control = list())
```

Arguments

- `matrix_lik`, an n by k matrix with (j,k)th element equal to \( f_k(x_j) \).
- `prior`, a k vector of the parameters of the Dirichlet prior on \( \pi \). Recommended to be `rep(1,k)`.
- `pi_init`, the initial value of \( \pi \) to use. If not specified defaults to \((1/k,\ldots,1/k)\).
- `weights`, an n vector of weights.
- `control` A list of control parameters for the SQUAREM algorithm, default value is set to be `control.default=list(K = 1, method=3, square=TRUE, step.min0=1, step.max0=1, mstep=4, kr=1, objfn.inc=1, tol=1.e-07, maxiter=5000, trace=FALSE)`.

Details

Fits a k component mixture model

\[
    f(x|\pi) = \sum_k \pi_k f_k(x)
\]

to independent and identically distributed data \( x_1, \ldots, x_n \) with weights \( w_1, \ldots, w_n \). Estimates mixture proportions \( \pi \) by maximum likelihood, or by maximum a posteriori (MAP) estimation for a Dirichlet prior on \( \pi \) (if a prior is specified). Here the log-likelihood for the weighted data is defined as \( l(\pi) = \sum_j w_j \log f(x_j|\pi) \). Uses the SQUAREM package to accelerate convergence of EM. Used by the ash main function; there is no need for a user to call this function separately, but it is exported for convenience.

Value

A list, including the estimates (pihat), the log likelihood for each iteration (B) and a flag to indicate convergence.
Index

ash, 4, 10
ash.workhorse, 5, 10
ash_pois, 10
ashci, 8, 9
ashr, 10
ashr-package (ashr), 10

calc_loglik, 11
calc_logLR, 12
calc_mixmean, 12
calc_mixsd, 13
calc_null_loglik, 13
calc_null_vloglik, 14
calc_vloglik, 14
calc_vlogLR, 15
cdf.ash, 15
cdf_conv, 16
cdf_post, 16, 52, 62
class, 7
comp_cdf, 17
comp_cdf_conv, 18
comp_cdf_conv.normalmix, 18
comp_cdf_conv.unimix, 19
comp_cdf_post, 19
comp_dens, 20
comp_dens_conv, 20
comp_dens_conv.normalmix, 21
comp_dens_conv.unimix, 21
comp_mean, 22
comp_mean.normalmix, 22
comp_mean.tnormalmix, 23
comp_mean2, 23
comp_postmean, 24
comp_postmean2, 24
comp_postprob, 25
comp_postsd, 25
comp_sd, 26
comp_sd.normalmix, 26
comp_sd.tnormalmix, 27
compute_lfsr, 17

cxxMixSquarem, 27
dens, 28
dens_conv, 28
dlogf, 29

estimate_mixprop, 29
gen_etruncFUN, 30
get_density, 31
get_fitted_g (get_lfsr), 31
get_lfdr (get_lfsr), 31
get_lfsr, 31
get_loglik (get_lfsr), 31
get_logLR (get_lfsr), 31
get_np (get_lfsr), 31
get_p10 (get_lfsr), 31
get_pm (get_lfsr), 31
get_post_sample, 33
get_pp (get_lfsr), 31
get_psd (get_lfsr), 31
get_qvalue (get_lfsr), 31
get_svalue (get_lfsr), 31

hist, 54

igmix, 33

lik_binom, 34
lik_logf, 35
lik_normal, 35
lik_normalmix, 36
lik_pois, 36
lik_t, 37

log_comp_dens_conv, 38
log_comp_dens_conv.normalmix, 39
log_comp_dens_conv.unimix, 39
loglik_conv, 37, 38
loglik_conv.default, 38

mixcdf, 40, 40

64
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixcdf.default</td>
<td>40</td>
</tr>
<tr>
<td>mixEM</td>
<td>41</td>
</tr>
<tr>
<td>mixIP</td>
<td>42</td>
</tr>
<tr>
<td>mixmean2</td>
<td>43</td>
</tr>
<tr>
<td>mixprop</td>
<td>43</td>
</tr>
<tr>
<td>mixSQP</td>
<td>44</td>
</tr>
<tr>
<td>mixsqp</td>
<td>44</td>
</tr>
<tr>
<td>mixVBEM</td>
<td>44</td>
</tr>
<tr>
<td>my_e2truncbeta</td>
<td>45</td>
</tr>
<tr>
<td>my_e2truncgamma</td>
<td>46</td>
</tr>
<tr>
<td>my_e2truncnorm</td>
<td>46, 49, 50</td>
</tr>
<tr>
<td>my_e2trunct</td>
<td>47</td>
</tr>
<tr>
<td>my_etrunctbetav</td>
<td>47</td>
</tr>
<tr>
<td>my_etrunctgamma</td>
<td>48</td>
</tr>
<tr>
<td>my_etrunctlogf</td>
<td>48</td>
</tr>
<tr>
<td>my_etrunctnorm</td>
<td>47, 49, 50</td>
</tr>
<tr>
<td>my_etrunct</td>
<td>49</td>
</tr>
<tr>
<td>my_vtruncnorm</td>
<td>47, 49, 50</td>
</tr>
<tr>
<td>ncomp</td>
<td>51</td>
</tr>
<tr>
<td>ncomp.default</td>
<td>51</td>
</tr>
<tr>
<td>normalmix</td>
<td>28, 51</td>
</tr>
<tr>
<td>pcdf_post</td>
<td>52</td>
</tr>
<tr>
<td>plogf</td>
<td>53</td>
</tr>
<tr>
<td>plot</td>
<td>53</td>
</tr>
<tr>
<td>plot.ash</td>
<td>53</td>
</tr>
<tr>
<td>plot_diagnostic</td>
<td>54</td>
</tr>
<tr>
<td>pm_on_zero</td>
<td>55</td>
</tr>
<tr>
<td>post_sample</td>
<td>57</td>
</tr>
<tr>
<td>post_sample.normalmix</td>
<td>57</td>
</tr>
<tr>
<td>post_sample.unimix</td>
<td>58</td>
</tr>
<tr>
<td>posterior_dist</td>
<td>55</td>
</tr>
<tr>
<td>postmean</td>
<td>56</td>
</tr>
<tr>
<td>postmean2</td>
<td>56</td>
</tr>
<tr>
<td>postsd</td>
<td>56</td>
</tr>
<tr>
<td>print.ash</td>
<td>58</td>
</tr>
<tr>
<td>prune</td>
<td>59</td>
</tr>
<tr>
<td>qval.from.lfdr</td>
<td>59</td>
</tr>
<tr>
<td>set_data</td>
<td>60</td>
</tr>
<tr>
<td>summary</td>
<td>61</td>
</tr>
<tr>
<td>summary.ash</td>
<td>60</td>
</tr>
<tr>
<td>tnormalmix</td>
<td>61</td>
</tr>
<tr>
<td>unimix</td>
<td>28, 61</td>
</tr>
<tr>
<td>vcdf_post</td>
<td>62</td>
</tr>
<tr>
<td>w_mixEM</td>
<td>63</td>
</tr>
</tbody>
</table>