Package ‘mashr’

December 7, 2022

Type Package
Encoding UTF-8
Title Multivariate Adaptive Shrinkage
Version 0.2.69
Date 2022-12-07
Description Implements the multivariate adaptive shrinkage (mash) method of Urbut et al (2019) <DOI:10.1038/s41588-018-0268-8> for estimating and testing large numbers of effects in many conditions (or many outcomes). Mash takes an empirical Bayes approach to testing and effect estimation; it estimates patterns of similarity among conditions, then exploits these patterns to improve accuracy of the effect estimates. The core linear algebra is implemented in C++ for fast model fitting and posterior computation.

URL https://github.com/stephenslab/mashr

BugReports https://github.com/stephenslab/mashr/issues
License BSD_3_clause + file LICENSE
Copyright file COPYRIGHTS
SystemRequirements C++11
Depends R (>= 3.3.0), ashr (>= 2.2-22)
Imports assertthat, utils, stats, plyr, rmeta, Rcpp (>= 0.12.11), mvtnorm, abind, softImpute
LinkingTo Rcpp, RcppArmadillo, RcppGSL (>= 0.3.8)
Suggests MASS, REBayes, corrplot (>= 0.90), testthat, kableExtra, knitr, rmarkdown, profmem, flashier, ebnm

Additional_repositories https://stephenslab.github.io/flashier-drat

VignetteBuilder knitr
NeedsCompilation yes
Biarch true
RoxygenNote 7.1.2
Author Matthew Stephens [aut], Sarah Urbut [aut], Gao Wang [aut], Yuxin Zou [aut], Yunqi Yang [ctb], Sam Roweis [cph], David Hogg [cph], Jo Bovy [cph], Peter Carbonetto [aut, cre]

Maintainer Peter Carbonetto <peter.carbonetto@gmail.com>

Repository CRAN

Date/Publication 2022-12-07 21:20:06 UTC

R topics documented:

- contrast_matrix ................................................. 3
- cov_canonical .................................................. 3
- cov_ed ............................................................. 4
- cov_flash .......................................................... 5
- cov_pca ............................................................ 6
- cov_udi ............................................................. 7
- estimate_null_correlation_simple ............................. 8
- extreme_deconvolution ......................................... 8
- get_estimated_pi .................................................. 11
- get_log10bf ....................................................... 12
- get_n_significant_conditions ................................. 13
- get_pairwise_sharing ........................................... 13
- get_pairwise_sharing_from_samples .......................... 14
- get_samples ....................................................... 15
- get_significant_results ....................................... 16
- mash ............................................................... 16
- mash_1by1 ......................................................... 19
- mash_compute_loglik ............................................ 19
- mash_compute_posterior_matrices ............................. 20
- mash_compute_vloglik ......................................... 21
- mash_estimate_corr_em ........................................ 22
- mash_plot_meta .................................................. 24
- mash_set_data .................................................... 24
- mash_update_data ............................................... 25
- simple_sims ...................................................... 26
- simple_sims2 ..................................................... 27
- sim_contrast1 .................................................... 27
- sim_contrast2 .................................................... 28

Index 29
contrast_matrix

Create contrast matrix

Description

Create contrast matrix

Usage

contrast_matrix(R, ref, name = 1:R)

Arguments

R the number of column for the contrast matrix
ref the reference group. It could be a number between 1,..., R, R is number of conditions, or the name of reference group. If there is no reference group, it can be the string 'mean'.
name a length R vector contains the name for conditions

Examples

contrast_matrix(5, 'mean')

cov_canonical

Compute a list of canonical covariance matrices

Description

Compute a list of canonical covariance matrices

Usage

cov_canonical(
  data,
  cov_methods = c("identity", "singletons", "equal_effects", "simple_het")
)

Arguments

data a mash data object, eg as created by mash_set_data
cov_methods a vector of strings indicating the matrices to be used: "identity" for the identity (effects are independent among conditions); "singletons" for the set of matrices with just one non-zero entry x_{jj} = 1 (j=1,...,R); (effect specific to condition j); "equal_effects" for the matrix of all 1s (effects are equal among conditions); "simple_het" for a set of matrices with 1s on the diagonal and all off-diagonal elements equal to 0.25, 0.5 or 0.75; see cov_simple_het for details; (effects are correlated among conditions).
Details

The default is that this function computes covariance matrices corresponding to the "bmalite" models.

Value

a list of covariance matrices

Examples

data = mash_set_data(Bhat = cbind(c(1,2),c(3,4)), Shat = cbind(c(1,1),c(1,1)))
cov_canonical(data)
cov_canonical(data,"singletons")
cov_canonical(data,c("id","sing")) # can use partial matching of names

cov_ed

Perform "extreme deconvolution" (Bovy et al) on a subset of the data

Description

Perform "extreme deconvolution" (Bovy et al) on a subset of the data

Usage

cov_ed(data, Ulist_init, subset = NULL, algorithm = c("bovy", "teem"), ...)

Arguments

data a mash data object
Ulist_init a named list of covariance matrices to use to initialize ED; default is to use matrices from PCs
subset a subset of data to be used when ED is run (set to NULL for all the data)
algorithm algorithm to run ED
... other arguments to be passed to ED algorithm, see extreme_deconvolution for algorithm 'bovy', or teem_wrapper for algorithm 'teem'

Details

Runs the extreme deconvolution algorithm from Bovy et al (Annals of Applied Statistics) to estimate data-driven covariance matrices. It can be initialized with, for example running cov_pca with, say, 5 PCs.
Examples

data = mash_set_data(Bhat = cbind(c(1,2),c(3,4)), Shat = cbind(c(1,1),c(1,1)))
U_pca = cov_pca(data,2)
U_x = apply(data$Bhat, 2, function(x) x - mean(x))
U_xx = t(U_x) %*% U_x / nrow(U_x)
cov_ed(data,c(U_pca, list(xx = U_xx)))

---

cov_flash  
Perform Empirical Bayes Matrix Factorization using flashr, and return  
a list of candidate covariance matrices

Description

Perform Empirical Bayes Matrix Factorization using flashr, and return a list of candidate covariance matrices

Usage

cov_flash(
  data,
  factors = c("default", "nonneg"),
  subset = NULL,
  remove_singleton = FALSE,
  tag = NULL,
  output_model = NULL,
  greedy_args = list(),
  backfit_args = list()
)

Arguments

data A "mash" data object.
factors If factors = "default", the factors and loadings are both unconstrained. If factors = "nonneg", the factors are constrained to be non-negative, and the loadings are unconstrained.
subset Data samples (rows) used to estimate the covariances. Set to NULL to use all the data.
remove_singleton If remove_singleton = TRUE, factors corresponding to singleton matrices will be removed from the output.
tag How to name the covariance matrices.
output_model The fitted flash model will be saved to this file (using saveRDS).
greedy_args List containing additional parameters passed to flashier::flash.add.greedy.
backfit_args List containing additional parameters passed to flashier::flash.backfit.
cov_pca

Value

A list of covariance matrices.

Examples

# See https://stephenslab.github.io/mashr/articles/flash_mash.html
# for an example

data = mash_set_data(Bhat = cbind(c(1,2),c(3,4)), Shat = cbind(c(1,1),c(1,1)))
cov_pca(data, 2)

cov_pca | Perform PCA on data and return list of candidate covariance matrices

Description

Perform PCA on data and return list of candidate covariance matrices

Usage

cov_pca(data, npc, subset = NULL)

Arguments

data | a mash data object
npc | the number of PCs to use
subset | indices of the subset of data to use (set to NULL for all data)

Value

Returns a list of covariance matrices: the npc rank-one covariance matrices based on the first npc PCs, and the rank npc covariance matrix

Examples

data = mash_set_data(Bhat = cbind(c(1,2),c(3,4)), Shat = cbind(c(1,1),c(1,1)))
cov_pca(data, 2)
cov_udi

Compute a list of covariance matrices corresponding to the "Unassociated", "Directly associated" and "Indirectly associated" models

Description

Compute a list of covariance matrices corresponding to the "Unassociated", "Directly associated" and "Indirectly associated" models

Usage

cov_udi(data, model = udi_model_matrix(n_conditions(data)))

Arguments

data
  a mash data object, eg as created by mash_set_data

model
  a model matrix with R columns, where R is the number of conditions in the data; each row should be a vector of length R with elements "U","D" and "I" indicating whether each effect is Unassociated, Directly associated or Indirectly associated

Details

If model is specified then this returns the covariance matrices for those models. The default creates all possible models. For a description of the "Unassociated", "Directly associated" and "Indirectly associated" models see Stephens M (2013), A unified framework for Association Analysis with Multiple Related Phenotypes, PloS ONE.

Value

a named list of covariance matrices

Examples

data = mash_set_data(Bhat = cbind(c(1,2),c(3,4)), Shat = cbind(c(1,1),c(1,1)))
cov_udi(data)
cov_udi(data,c('I','D'))
estimate_null_correlation_simple

Estimate null correlations (simple)

Description

Estimates a null correlation matrix from data using simple z score threshold

Usage

estimate_null_correlation_simple(data, z_thresh = 2, est_cor = TRUE)

Arguments

data a mash data object, eg as created by mash_set_data
z_thresh the z score threshold below which to call an effect null
est_cor whether to estimate correlation matrix (TRUE) or the covariance matrix (FALSE).

Details

Returns a simple estimate of the correlation matrix (or covariance matrix) among conditions under the null. Specifically, the simple estimate is the empirical correlation (or covariance) matrix of the z scores for those effects that have (absolute) z score < z_thresh in all conditions.

Examples

simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
estimate_null_correlation_simple(data)

density estimation using Gaussian mixtures in the presence of noisy, heterogeneous and incomplete data

Description

We present a general algorithm to infer a d-dimensional distribution function given a set of heterogeneous, noisy observations or samples. This algorithm reconstructs the error-deconvolved or 'underlying' distribution function common to all samples, even when the individual samples have unique error and missing-data properties. The underlying distribution is modeled as a mixture of Gaussians, which is completely general. Model parameters are chosen to optimize a justified, scalar objective function: the logarithm of the probability of the data under the error-convolved model, where the error convolution is different for each data point. Optimization is performed by an Expectation Maximization (EM) algorithm, extended by a regularization technique and 'split-and-merge' procedure. These extensions mitigate problems with singularities and local maxima, which are often encountered when using the EM algorithm to estimate Gaussian density mixtures.
Usage

```r
extreme_deconvolution(
    ydata,  # [ndata,dy] matrix of observed quantities
    ycovar, # [ndata,dy] / [ndata,dy,dy] / [dy,dyndata] matrix, list or 3D array of observational
              # error covariances (if [ndata,dy] then the error correlations are assumed to vanish)
    xamp,   # [ngauss] array of initial amplitudes (*not* [1,ngauss])
    xmean,  # [ngauss,dx] matrix of initial means
    xcovar, # [ngauss,dx,dx] list of matrices of initial covariances
    projection = NULL,  # [ndata,dy,dx] list of projection matrices
    weight = NULL,  # [ndata] array of weights to be applied to the data points
    fixamp = NULL,  # (default=None) None, True/False, or list of bools
    fixmean = NULL,  # (default=None) None, True/False, or list of bools
    fixcovar = NULL,  # (default=None) None, True/False, or list of bools
    tol = 1e-06,  # (double, default=1.e-6) tolerance for convergence
    maxiter = 1e+09,  # (long, default= 10**9) maximum number of iterations to perform
    w = 0,  # (double, default=0.) covariance regularization parameter (of the conjugate prior)
    logfile = NULL,  # basename for several logfiles (_c.log has output from the c-routine; _loglike.log
                      # has the log likelihood path of all the accepted routes, i.e. only parts which
                      # increase the likelihood are included, during splitmerge)
    splitnmerge = 0,  # (int, default=0) depth to go down the splitmerge path
    maxsnm = FALSE,  # (Bool, default=False) use the maximum number of split 'n' merge steps, K*(K-1)*(K-2)/2
    likeonly = FALSE,  # use the maximum number of split 'n' merge steps, K*(K-1)*(K-2)/2
    logweight = FALSE  # logweight = FALSE
)
```

Arguments

- `ydata`: [ndata,dy] matrix of observed quantities
- `ycovar`: [ndata,dy] / [ndata,dy,dy] / [dy,dyndata] matrix, list or 3D array of observational error covariances (if [ndata,dy] then the error correlations are assumed to vanish)
- `xamp`: [ngauss] array of initial amplitudes (*not* [1,ngauss])
- `xmean`: [ngauss,dx] matrix of initial means
- `xcovar`: [ngauss,dx,dx] list of matrices of initial covariances
- `projection`: [ndata,dy,dx] list of projection matrices
- `weight`: [ndata] array of weights to be applied to the data points
- `fixamp`: (default=None) None, True/False, or list of bools
- `fixmean`: (default=None) None, True/False, or list of bools
- `fixcovar`: (default=None) None, True/False, or list of bools
- `tol`: (double, default=1.e-6) tolerance for convergence
- `maxiter`: (long, default= 10**9) maximum number of iterations to perform
- `w`: (double, default=0.) covariance regularization parameter (of the conjugate prior)
- `logfile`: basename for several logfiles (_c.log has output from the c-routine; _loglike.log has the log likelihood path of all the accepted routes, i.e. only parts which increase the likelihood are included, during splitmerge)
- `splitnmerge`: (int, default=0) depth to go down the splitmerge path
- `maxsnm`: (Bool, default=False) use the maximum number of split 'n' merge steps, K*(K-1)*(K-2)/2
**likeonly** (Bool, default=False) only compute the total log likelihood of the data

**logweight** (bool, default=False) if True, weight is actually log(weight)

### Value

- **avgloglikedata**: avgloglikedata after convergence
- **xamp**: updated xamp
- **xmean**: updated xmean
- **xcovar**: updated xcovar

### Author(s)

Jo Bovy, David W. Hogg, & Sam T. Roweis

### References

Inferring complete distribution functions from noisy, heterogeneous and incomplete observations
Jo Bovy, David W. Hogg, & Sam T. Roweis, Submitted to AOAS (2009) [arXiv/0905.2979]

### Examples

```r
ydata <-
c(2.62434536, 0.38824359, 0.47182825, -0.07296862, 1.86540763,
-1.30153870, 2.74481176, 0.23879310, 1.31983910, 0.75062962,
2.46210794, -1.06014071, 0.67758280, 0.61594565, 2.13376944,
-0.09989127, 0.82757179, 0.12214158, 1.04221375, 1.58281521,
-0.10061918, 2.14472371, 1.90159072, 1.50249434, 1.90085595,
0.31627214, 0.87710977, 0.06423057, 0.73211192, 1.53835547,
0.30833925, 0.60324647, 0.31282730, 0.15479436, 0.32875387,
0.98735450, -0.11731035, 1.23441570, 2.65980218, 1.74204416,
0.88816445, 0.11237104, 0.25284171, 2.69245460, 1.95080775,
0.36390435, 1.19091548, 3.10025514, 1.12015895, 1.61720311,
1.30017032, 0.64775015, -0.14251820, 0.65065728, 0.79110577,
1.58662319, 1.83898341, 1.93110208, 1.28558733, 1.88514116,
0.24560206, 2.25286816, 1.51292982, 0.70190717, 1.48851815,
0.92448289, 2.13162939, 2.51981682, 3.18557541, -0.39649633,
-0.44411380, 0.49553414, 1.1603707, 1.87616892, 1.31563495,
-1.0228122, 0.69379599, 1.82797464, 1.23009474, 1.76201118,
0.77767186, 0.79924193, 1.1856139, 1.41005165, 1.19289972,
1.11900865, 0.32933771, 1.37756379, 1.12182127, 2.12948391,
2.1981788, 1.18561642, 0.62471505, 0.36126959, 1.4239435,
1.07734007, 0.65614632, 1.04359686, 0.37999916, 1.69803203,
0.55287144, 2.24250778, 1.40349164, 1.59357852, -0.09491185,
1.16938243, 1.74055645, 0.04629940, 0.73378149, 1.83261455,
-0.37311732, 1.31515939, 1.84616065, 0.14048406, 1.35054598,
-0.31228341, 0.96104494, -0.61577236, 2.12141771, 1.40890054,
0.97538394, 0.22488338, 2.27375593, 2.96718175, -0.85798186,
2.23616403, 2.62765075, 1.33801170, -0.19928603, 1.86334532,
0.81967970, 0.39687937, -0.23005814, 1.55053758, 1.79280687,
0.37646927, 1.52057634, -0.1443139, 1.80186103, 1.04657630,
0.81343023, 0.89825413, 1.86888616, 1.75041164, 1.52946532,

```
get_estimated_pi

Return the estimated mixture proportions

Description

Return the estimated mixture proportions

Usage

get_estimated_pi(m, dimension = c("cov", "grid", "all"))

Arguments

m

dimension

the mash result

indicates whether you want the mixture proportions for the covariances, grid, or all

Details

If the fit was done with `usepointmass=TRUE` then the first element of the returned vector will correspond to the null, and the remaining elements to the non-null covariance matrices. Suppose the fit was done with $K$ covariances and a grid of length $L$. If `dimension= cov` then the
returned vector will be of length $K$ (or $K+1$ if ‘usepointmass=TRUE’). If ‘dimension=grid’ then the returned vector will be of length $L$ (or $L+1$). If ‘dimension=all’ then the returned vector will be of length $LK$ (or $LK+1$). The names of the vector will be informative for which combination each element corresponds to.

Value

a named vector containing the estimated mixture proportions.

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
get_estimated_pi(m)
```

---

**get_log10bf**  
*Return the Bayes Factor for each effect*

**Description**

Return the Bayes Factor for each effect

**Usage**

```r
get_log10bf(m)
```

**Arguments**

- `m`  
  the mash result (from joint or 1by1 analysis); must have been computed using `usepointmass=TRUE`

**Value**

if `m` was fitted using `usepointmass=TRUE` then returns a vector of the log10(bf) values for each effect. That is, the jth element `lb[j]` is log10(Pr(Bj | g=gnullnonnull)/Pr(Bj | g = 0)) where gnullnonnull is the non-null part of ghat. Otherwise returns NULL.

**Examples**

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
get_log10bf(m)
```
**get_n_significant_conditions**

*Count number of conditions each effect is significant in*

**Description**

Count number of conditions each effect is significant in

**Usage**

```r
get_n_significant_conditions(
  m, 
  thresh = 0.05, 
  conditions = NULL, 
  sig_fn = get_lfsr
)
```

**Arguments**

- `m` the mash result (from joint or 1by1 analysis)
- `thresh` indicates the threshold below which to call signals significant
- `conditions` which conditions to include in check (default to all)
- `sig_fn` the significance function used to extract significance from mash object; eg could be ashr::get_lfsr or ashr::get_lfdr

**Value**

a vector containing the number of significant conditions

**Examples**

```r
simdata = simple_sims(50, 5, 1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
get_n_significant_conditions(m)
```

**get_pairwise_sharing**

*Compute the proportion of (significant) signals shared by magnitude in each pair of conditions, based on the posterior mean*

**Description**

Compute the proportion of (significant) signals shared by magnitude in each pair of conditions, based on the posterior mean
Usage

get_pairwise_sharing(m, factor = 0.5, lfsr_thresh = 0.05, FUN = identity)

Arguments

m the mash fit
factor a number in [0,1] the factor within which effects are considered to be shared
lfsr_thresh the lfsr threshold for including an effect in the assessment
FUN a function to be applied to the estimated effect sizes before assessing sharing. The most obvious choice beside the default 'FUN=identity' would be 'FUN=abs' if you want to ignore the sign of the effects when assessing sharing.

Details

For each pair of tissues, first identify the effects that are significant (by lfsr<lfsr_thresh) in at least one of the two tissues. Then compute what fraction of these have an estimated (posterior mean) effect size within a factor ‘factor’ of one another. The results are returned as an R by R matrix.

Examples

simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
get_pairwise_sharing(m) # sharing by magnitude (same sign)
get_pairwise_sharing(m, factor=0) # sharing by sign
get_pairwise_sharing(m, FUN=abs) # sharing by magnitude when sign is ignored
Arguments

m the mash fit with samples from posteriors
factor a number in [0,1] the factor within which effects are considered to be shared
lfsr_thresh the lfsr threshold for including an effect in the assessment
FUN a function to be applied to the estimated effect sizes before assessing sharing. The most obvious choice beside the default 'FUN=identity' would be 'FUN=abs' if you want to ignore the sign of the effects when assessing sharing.

Details

For each pair of conditions, compute the fraction of effects that are within a factor ‘factor’ of one another. The results are returned as an R by R matrix.

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data), posterior_samples=5, algorithm='R')
get_pairwise_sharing_from_samples(m) # sharing by magnitude (same sign)
get_pairwise_sharing_from_samples(m, factor=0) # sharing by sign
get_pairwise_sharing_from_samples(m, FUN=abs) # sharing by magnitude when sign is ignored
```

---

get_samples

Return samples from a mash object

Description

Return samples from a mash object

Usage

get_samples(m)

Arguments

m The mash fit.

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data), posterior_samples=5, algorithm='R')
get_samples(m)
```
get_significant_results

Find effects that are significant in at least one condition

Description

Find effects that are significant in at least one condition

Usage

get_significant_results(m, thresh = 0.05, conditions = NULL, sig_fn = get_lfsr)

Arguments

- **m**: the mash result (from joint or 1by1 analysis)
- **thresh**: indicates the threshold below which to call signals significant
- **conditions**: which conditions to include in check (default to all)
- **sig_fn**: the significance function used to extract significance from mash object; eg could be ashr::get_lfsr or ashr::get_lfdr. (Small values must indicate significant.)

Value

a vector containing the indices of the significant effects, by order of most significant to least

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
get_significant_results(m)
```

mash

Apply mash method to data

Description

Apply mash method to data
mash

Usage

mash(
  data,
  Ulist = NULL,
  gridmult = sqrt(2),
  grid = NULL,
  normalizeU = TRUE,
  usepointmass = TRUE,
  g = NULL,
  fixg = FALSE,
  prior = c("nullbiased", "uniform"),
  nullweight = 10,
  optmethod = c("mixSQP", "mixIP", "mixEM", "cxxMixSquarem"),
  control = list(),
  verbose = TRUE,
  add.mem.profile = FALSE,
  algorithm.version = c("Rcpp", "R"),
  pi_thresh = 1e-10,
  A = NULL,
  posterior_samples = 0,
  seed = 123,
  outputlevel = 2,
  output_lfdr = FALSE
)

Arguments

data a mash data object containing the Bhat matrix, standard errors, alpha value; created using mash_set_data or mash_set_data_contrast
Ulist a list of covariance matrices to use (see normalizeU for rescaling these matrices)
gridmult scalar indicating factor by which adjacent grid values should differ; close to 1 for fine grid
grid vector of grid values to use (scaling factors omega in paper)
normalizeU whether or not to normalize the U covariances to have maximum of 1 on diagonal
usepointmass whether to include a point mass at 0, corresponding to null in every condition
g the value of g obtained from a previous mash fit - an alternative to supplying Ulist, grid and usepointmass
fixg if g is supplied, allows the mixture proportions to be fixed rather than estimated; e.g., useful for fitting mash to test data after fitting it to training data
prior indicates what penalty to use on the likelihood, if any
nullweight scalar, the weight put on the prior under “nullbiased” specification, see “prior”.
optmethod name of optimization method to use
control A list of control parameters passed to optmethod.
verbose If TRUE, print progress to R console.
add.mem.profile
If TRUE, print memory usage to R console (requires R library ‘profmem’).

algorithm.version
Indicates whether to use R or Rcpp version

pi_thresh
threshold below which mixture components are ignored in computing posterior summaries (to speed calculations by ignoring negligible components)

A
the linear transformation matrix, \( Q \times R \) matrix. This is used to compute the posterior for \( Ab \).

posterior.samples
the number of samples to be drawn from the posterior distribution of each effect.

seed
A random number seed to use when sampling from the posteriors. It is used when posterior.samples > 0.

outputlevel
controls amount of computation / output; 1: output only estimated mixture component proportions, 2: and posterior estimates, 3: and posterior covariance matrices, 4: and likelihood matrices

output_lfdr
If output_lfdr = TRUE, output local false discovery rate estimates. The lfdr tends to be sensitive to mis-estimated covariance matrices, and generally we do not recommend using them; we recommend using the local false sign rate (lfsr) instead, which is always returned, even when output_lfdr = TRUE.

Value
a list with elements result, loglik and fitted_g

Examples

```r
Bhat = matrix(rnorm(100),ncol=5) # create some simulated data
Shat = matrix(rep(1,100),ncol=5)
data = mash_set_data(Bhat,Shat, alpha=1)
U.c = cov_canonical(data)
res.mash = mash(data,U.c)

# Run mash with penalty exponent on null term equal to 100.
# See "False discovery rates: a new deal" (M. Stephens 2017),
# supplementary material S.2.5 for more details.
set.seed(1)
simdata = simple_sims(500,5,1)
data = mash_set_data(simdata$Bhat,simdata$Shat)
U.c = cov_canonical(data)
res0 = mash(data,U.c)
res1 = mash(data,U.c,prior = "nullbiased",nullweight = 101)
plot(res0$fitted_g$pi,res1$fitted_g$pi,pch = 20)
abline(a = 0,b = 1,col = "skyblue",lty = "dashed")
```

mash_1by1

Perform condition-by-condition analyses

Description

Performs simple "condition-by-condition" analysis by running ash from package ashr on data from each condition, one at a time. May be a useful first step to identify top hits in each condition before a mash analysis.

Usage

mash_1by1(data, alpha = 0, ...)

Arguments

data A list with the following two elements: Bhat an n by R matrix of observations (n units in R conditions); and Shat, an n by R matrix of standard errors (n units in R conditions).
alpha Numeric value of alpha parameter in the model. alpha = 0 for Exchangeable Effects (EE), alpha = 1 for Exchangeable Z-scores (EZ).
... optionally, other parameters to be passed to ash

Value

A list similar to the output of mash, particularly including posterior matrices.

Examples

simdata = simple_sims(50,5,1)
mash_1by1(simdata)

mash_compute_loglik

Compute loglikelihood for fitted mash object on new data.

Description

Compute loglikelihood for fitted mash object on new data.

Usage

mash_compute_loglik(g, data, algorithm.version = c("Rcpp", "R"))
Arguments

g  A mash object or the fitted_g from a mash object.
data  A set of data on which to compute the loglikelihood.
algorithm.version  Indicate R or Rcpp version

Details

The log-likelihood for each element is \( p(B_{hat,j}|Shat_j, g, \alpha) \) where \( B_{hat,j}|B_j, Shat_j \sim N(B_j, Shat_j) \) and \( B_j/\sqrt{Shat_j} | Shat_j \sim g \).

Value

The log-likelihood for data computed using g.

Examples

```
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
mash_compute_loglik(m, data)
```

mash_compute_posterior_matrices

`mash_compute_posterior_matrices`  
`Compute posterior matrices for fitted mash object on new data`

Description

Compute posterior matrices for fitted mash object on new data

Usage

```
mash_compute_posterior_matrices(
g,  
data,  
pi_thresh = 1e-10,  
algorithm.version = c("Rcpp", "R"),  
A = NULL,  
output_posterior_cov = FALSE,  
posterior_samples = 0,  
seed = 123
)
```
mash_compute_vloglik

Arguments

- `g`: a mash object or the fitted_g from a mash object.
- `data`: a set of data on which to compute the posterior matrices
- `pi_thresh`: threshold below which mixture components are ignored in computing posterior summaries (to speed calculations by ignoring negligible components)
- `algorithm.version`: Indicates whether to use R or Rcpp version
- `A`: the linear transformation matrix, Q x R matrix. This is used to compute the posterior for Ab.
- `output_posterior_cov`: whether or not to output posterior covariance matrices for all effects
- `posterior_samples`: the number of samples to be drawn from the posterior distribution of each effect.
- `seed`: a random number seed to use when sampling from the posteriors. It is used when posterior_samples > 0.

Value

A list of posterior matrices

Examples

```r
simdata = simple_sims(50, 5, 1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
mash_compute_posterior_matrices(m, data)
```

mash_compute_vloglik

Compute vector of loglikelihood for fitted mash object on new data

Description

Compute vector of loglikelihood for fitted mash object on new data

Usage

```r
mash_compute_vloglik(g, data, algorithm.version = c("Rcpp", "R"))
```

Arguments

- `g`: A mash object.
- `data`: A set of data on which to compute the loglikelihood.
- `algorithm.version`: Indicate R or Rcpp version
Details

The log-likelihood for each element is $p(Bhat_j|Shat_j, g, \alpha)$ where $Bhat_j|B_j, Shat_j \sim N(B_j, Shat_j)$ and $B_j/\sqrt{Shat_j} \sim g$ Here the value of $\alpha$ is set when setting up the data object in 'mash_set_data'. If $g$ is a mash object (safest!) then the function will check that this value matches the $\alpha$ used when fitting 'mash'. Note: as a convenience, this function can also be called with $g$ a mixture distribution with same structure as the fitted_g from a mash object. This is mostly useful when doing simulations, where you might want to compute the likelihood under the "true" $g$. When used in this way the user is responsible for making sure that the $g$ makes sense with the alpha set in data.

Value

The vector of log-likelihoods for each data point computed using $g$.

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
mash_compute_vloglik(m, data)
```

---

**mash_estimate_corr_em**  
Fit mash model and estimate residual correlations using EM algorithm

Description

Estimates a residual correlation matrix from data using an ad hoc EM algorithm.

Usage

```r
mash_estimate_corr_em(
data,
Ulist,
init,
max_iter = 30,
tol = 1,
est_cor = TRUE,
track_fit = FALSE,
prior = c("nullbiased", "uniform"),
details = TRUE,
...)
```
mash_estimate_corr_em

Arguments

- **data**: a mash data object, eg as created by `mash_set_data`
- **Ulist**: a list of covariance matrices to use
- **init**: the initial value for the residual correlation. If it is not given, we use result from `estimate_null_correlation_simple`
- **max_iter**: maximum number of iterations to perform
- **tol**: convergence tolerance
- **est_cor**: whether to estimate correlation matrix (TRUE) or the covariance matrix (FALSE)
- **track_fit**: add an attribute `trace` to output that saves current values of all iterations
- **prior**: indicates what penalty to use on the likelihood, if any
- **details**: whether to return details of the model, if it is TRUE, the mash model, the number of iterations and the value of objective functions will be returned
- **...**: other parameters pass to `mash`

Details

Returns the estimated residual correlation matrix among conditions. We estimate the residual correlation matrix using an ad hoc EM algorithm. The update in the ad hoc M step is not guaranteed to increase the likelihood, therefore, the EM algorithm is stopped before the likelihood drops. The residual correlation matrix $V$ is estimated using the posterior second moment of the noise.

Warning: This method could take some time. The `estimate_null_correlation_simple` gives a quick approximation for the null correlation matrix.

Value

the estimated correlation matrix and the fitted mash model

| V | estimated residual correlation matrix |
| mash.model | fitted mash model |

Examples

```r
simdata = simple_sims(100,5,1)
m.1by1 = mash.1by1(mash_set_data(simdata$Bhat,simdata$Shat))
strong.subset = get_significant_results(m.1by1,0.05)
random.subset = sample(1:nrow(simdata$Bhat),20)
data.strong = mash_set_data(simdata$Bhat[strong.subset,], simdata$Shat[strong.subset,])
data.tmp = mash_set_data(simdata$Bhat[random.subset,], simdata$Shat[random.subset,])
U_pca = cov_pca(data.strong, 3)
U_ed = cov_ed(data.strong, U_pca)
Vhat = mash_estimate_corr_em(data.tmp, U_ed)
```
mash_plot_meta

Plot metaplot for an effect based on posterior from mash

Description
Plot metaplot for an effect based on posterior from mash

Usage
mash_plot_meta(m, i, xlab = "Effect size", ylab = "Condition", ...)

Arguments
m  the result of a mash fit
i  index of the effect to plot
xlab Character string specifying x-axis label.
ylab Character string specifying y-axis label.
... Additional arguments passed to metaplot.

Examples
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
m = mash(data, cov_canonical(data))
mash_plot_meta(m,1)

mash_set_data
Create a data object for mash analysis.

Description
Create a data object for mash analysis.

Usage
mash_set_data(
  Bhat,
  Shat = NULL,
  alpha = 0,
  df = Inf,
  pval = NULL,
  V = diag(ncol(Bhat)),
  zero_check_tol = .Machine$double.eps,
  zero_Bhat_Shat_reset = 0,
  zero_Bhat_Shat_reset = 0
)


Arguments

\[ Bhat \]  
An N by R matrix of observed estimates.

\[ Shat \]  
An N by R matrix of corresponding standard errors. Shat can be a scalar if all standard errors are equal. This is most useful if Bhat is a matrix of Z scores, so elements of Shat are all 1. Default is 1.

\[ alpha \]  
Numeric value of alpha parameter in the model. alpha = 0 for Exchangeable Effects (EE), alpha = 1 for Exchangeable Z-scores (EZ). Default is 0. Please refer to equation (3.2) of M. Stephens 2016, Biostatistics for a discussion on alpha.

\[ df \]  
An N by R matrix of corresponding degrees of freedom of the t-statistic Bhat/Shat. Can be a scalar if all degrees of freedom are equal. Default is inf (for large samples).

\[ pval \]  
An N by R matrix of p-values of t-statistic Bhat/Shat. Shat and df should not be specified when pval is provided.

\[ V \]  
an R by R matrix / \([R x R x N]\) array of effect specific correlation matrix of error correlations; must be positive definite. \([So Bhat_j distributed as N(B_j,diag(Shat_j) V[^{\[\cdot\]}j] diag(Shat_j)) where _j denotes the jth row of a matrix]\). Defaults to identity.

\[ zero_check_tol \]  
a small positive number as threshold for Shat to be considered zero if any Shat is smaller or equal to this number.

\[ zero_Bhat_Shat_reset \]  
Replace zeros in Shat matrix to given value if the corresponding Bhat are also zeros.

\[ zero_Shat_reset \]  
Replace zeros in Shat matrix to given value.

Value

A data object for passing into mash functions.

Examples

```r
simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
```

Description

This function can update two parts of the mash data. The first one is setting the reference group, so the mash data can be used for commonbaseline analysis. The other one is updating the null correlation matrix.
Usage

mash_update_data(mashdata, ref = NULL, V = NULL)

Arguments

mashdata  mash data object containing the Bhat matrix, standard errors, V; created using mash_set_data
ref  the reference group. It could be a number between 1,..., R, R is number of conditions, or the name of reference group. If there is no reference group, it can be the string 'mean'.
V  an R by R matrix / [R x R x N] array of correlation matrix of error correlations

Value

a updated mash data object

Examples

simdata = simple_sims(50,5,1)
data = mash_set_data(simdata$Bhat, simdata$Shat)
mash_update_data(data, 'Var mean')

Description

Create some simple simulated data for testing purposes

Usage

simple_sims(nsamp = 100, ncond = 5, err_sd = 0.01)

Arguments

nsamp  number of samples of each type
ncond  number of conditions
err_sd  the standard deviation of the errors

Details

The simulation consists of equal numbers of four different types of effects: null, equal among conditions, present only in first condition, independent across conditions

Examples

simple_sims(100, 5)
simple_sims2

Create some more simple simulated data for testing purposes

Description

Create some more simple simulated data for testing purposes

Usage

\begin{Verbatim}
\texttt{simple_sims2}(nsamp = 100, err_sd = 0.01)
\end{Verbatim}

Arguments

\begin{itemize}
  \item \texttt{nsamp} \hspace{1cm} number of samples of each type
  \item \texttt{err_sd} \hspace{1cm} the standard deviation of the errors
\end{itemize}

Details

The simulation consists of five conditions with two types of effect, those present (and identical) in first two conditions and those present (and identical) in last three conditions

Examples

\begin{Verbatim}
\texttt{simple_sims2}(100, 5)
\end{Verbatim}

sim_contrast1

Create simplest simulation, \(cj = \mu 1\) data used for contrast analysis

Description

Create simplest simulation, \(cj = \mu 1\) data used for contrast analysis

Usage

\begin{Verbatim}
\texttt{sim_contrast1}(nsamp = 100, ncond = 5, err_sd = \sqrt{0.5})
\end{Verbatim}

Arguments

\begin{itemize}
  \item \texttt{nsamp} \hspace{1cm} number of samples of each type
  \item \texttt{ncond} \hspace{1cm} number of conditions
  \item \texttt{err_sd} \hspace{1cm} the standard deviation of the errors
\end{itemize}

Details

There is no true deviation exists in this case
Examples

```r
sim_contrast2(100,5)
```

---

**Description**

Create simulation with signal data used for contrast analysis.

**Usage**

```r
sim_contrast2(nsamp = 1000, ncond = 5, err_sd = sqrt(0.5))
```

**Arguments**

- `nsamp` Number of samples of each type.
- `ncond` Number of conditions.
- `err_sd` The standard deviation of the errors.

**Details**

The first condition is the reference group. The deviations are the difference between the subsequent conditions with the reference group. The simulation consists of 90 10 different types of deviations: equal among conditions, present only in the first subsequent condition, independent across conditions.

**Examples**

```r
sim_contrast2(100,5)
```
Index

contrast_matrix, 3
cov_canonical, 3
cov_ed, 4
cov_flash, 5
cov_pca, 6
cov_udi, 7
estimate_null_correlation_simple, 8, 23
extreme_deconvolution, 4, 8
get_estimated_pi, 11
get_log10bf, 12
get_n_significant_conditions, 13
get_pairwise_sharing, 13
get_pairwise_sharing_from_samples, 14
get_samples, 15
get_significant_results, 16
mash, 16
mash_1by1, 19
mash_compute_loglik, 19
mash_compute_posterior_matrices, 20
mash_compute_vloglik, 21
mash_estimate_corr_em, 22
mash_plot_meta, 24
mash_set_data, 24
mash_update_data, 25
metaplot, 24
saveRDS, 5
sim_contrast1, 27
sim_contrast2, 28
simple_sims, 26
simple_sims2, 27
teeem_wrapper, 4