Package 'gipsDA'

February 3, 2026

Title Training DA Models Utilizing 'gips'

Version 0.1.2

URL https://AntoniKingston.github.io/gipsDA/, https://github.com/AntoniKingston/gipsDA

BugReports https://github.com/AntoniKingston/gipsDA/issues

Description Extends classical linear and quadratic discriminant analysis by incorporating permutation group symmetries into covariance matrix estimation. The package leverages methodology from the 'gips' framework to identify and impose permutation structures that act as a form of regularization, improving stability and interpretability in settings with symmetric or exchangeable features. Several discriminant analysis variants are provided, including pooled and class-specific covariance models, as well as multi-class extensions with shared or independent symmetry structures. For more details about 'gips' methodology see and Graczyk et al. (2022) <doi:10.1214/22-AOS2174> and Chojecki, Morgen, Kołodziejek (2025, <doi:10.18637/jss.v112.i07>).

License GPL-3

Imports dplyr, ggplot2, gips, jsonlite, lattice, patchwork, permutations, rlang, tibble, tidyr, MASS, numbers, stringi

Suggests mockery, testthat, roxygen2

Depends R (>= 2.10)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Antoni Zbigniew Kingston [aut], Norbert Maksymilian Frydrysiak [aut, cre], Adam Przemysław Chojecki [ctb] (ORCID: <https://orcid.org/0009-0008-2902-4096>)

Maintainer Norbert Maksymilian Frydrysiak <norbert.frydrysiak@proton.me>

Repository CRAN

Date/Publication 2026-02-03 13:30:02 UTC

Contents

`find_MAP` *Find the Maximum A Posteriori Estimation*

Description

Use one of the optimization algorithms to find the permutation that maximizes a posteriori probability based on observed data. Not all optimization algorithms will always find the MAP, but they try to find a significant value.

Usage

```
find_MAP(
  g,
  max_iter = NA,
  optimizer = NA,
  show_progress_bar = TRUE,
  save_all_perms = FALSE,
  return_probabilities = FALSE
)
```

Arguments

`g`	Object of a `gipsmult` class.
`max_iter`	The number of iterations for an algorithm to perform. At least 2. For `optimizer = "BF"`, it is not used; for `optimizer = "MH"`, it has to be finite; for `optimizer = "HC"`, it can be infinite.
`optimizer`	The optimizer for the search of the maximum posteriori: • `"BF"` (the default for unoptimized `g` with `perm size <= 9`) - Brute Force; • `"MH"` (the default for unoptimized `g` with `perm size > 10`) - Metropolis-Hastings; • `"HC"` - Hill Climbing; • `"continue"` (the default for optimized `g`) - The same as the `g` was optimized by (see Examples).

See the **Possible algorithms to use as optimizers** section below for more details.

`show_progress_bar`
: A boolean. Indicate whether or not to show the progress bar:
  - When `max_iter` is infinite, `show_progress_bar` has to be `FALSE`;
  - When `return_probabilities = TRUE`, then shows an additional progress bar for the time when the probabilities are calculated.

`save_all_perms`
: A boolean. `TRUE` indicates saving a list of all permutations visited during optimization. This can be useful sometimes but needs a lot more RAM.

`return_probabilities`
: A boolean. `TRUE` can only be provided only when `save_all_perms = TRUE`. For:
  - `optimizer = "MH"` - use Metropolis-Hastings results to estimate posterior probabilities;
  - `optimizer = "BF"` - use brute force results to calculate exact posterior probabilities.

  These additional calculations are costly, so a second and third progress bar is shown (when `show_progress_bar = TRUE`).

  To examine probabilities after optimization, call `get_probabilities_from_gipsmult()`.

Details

`find_MAP()` can produce a warning when:

- the optimizer "hill_climbing" gets to the end of its `max_iter` without converging.
- the optimizer will find the permutation with smaller `n0` than `number_of_observations`

Value

Returns an optimized object of a `gipsmult` class.

Possible algorithms to use as optimizers

For every algorithm, there are some aliases available.

- `"brute_force"`, `"BF"`, `"full"` - use the **Brute Force** algorithm that checks the whole permutation space of a given size. This algorithm will find the actual Maximum A Posteriori Estimation, but it is very computationally expensive for bigger spaces. We recommend Brute Force only for `p <= 9`.
- `"Metropolis_Hastings"`, `"MH"` - use the **Metropolis-Hastings** algorithm; see Wikipedia. The algorithm will draw a random transposition in every iteration and consider changing the current state (permutation). When the `max_iter` is reached, the algorithm will return the best permutation calculated as the MAP Estimator. This algorithm used in this context is a special case of the **Simulated Annealing** the user may be more familiar with; see Wikipedia.
- `"hill_climbing"`, `"HC"` - use the **hill climbing** algorithm; see Wikipedia. The algorithm will check all transpositions in every iteration and go to the one with the biggest a posteriori value. The optimization ends when all *neighbors* will have a smaller a posteriori value. If the `max_iter` is reached before the end, then the warning is shown, and it is recommended to

continue the optimization on the output of the `find_MAP()` with `optimizer = "continue"`; see examples. Remember that `p*(p-1)/2` transpositions will be checked in every iteration. For bigger `p`, this may be costly.

See Also

- `gipsmult()` - The constructor of a `gipsmult` class. The `gipsmult` object is used as the `g` parameter of `find_MAP()`.
- `plot.gipsmult()` - Practical plotting function for visualizing the optimization process.
- `get_probabilities_from_gipsmult()` - When `find_MAP(return_probabilities = TRUE)` was called, probabilities can be extracted with this function.
- `log_posteriori_of_gipsmult()` - The function that the optimizers of `find_MAP()` tries to find the argmax of.

Examples

```
require("MASS") # for mvrnorm()

perm_size <- 6
mu1 <- runif(6, -10, 10)
mu2 <- runif(6, -10, 10) # Assume we don't know the means
sigma1 <- matrix(
  data = c(
    1.0, 0.8, 0.6, 0.4, 0.6, 0.8,
    0.8, 1.0, 0.8, 0.6, 0.4, 0.6,
    0.6, 0.8, 1.0, 0.8, 0.6, 0.4,
    0.4, 0.6, 0.8, 1.0, 0.8, 0.6,
    0.6, 0.4, 0.6, 0.8, 1.0, 0.8,
    0.8, 0.6, 0.4, 0.6, 0.8, 1.0
  ),
  nrow = perm_size, byrow = TRUE
)
sigma2 <- matrix(
  data = c(
    1.0, 0.5, 0.2, 0.0, 0.2, 0.5,
    0.5, 1.0, 0.5, 0.2, 0.0, 0.2,
    0.2, 0.5, 1.0, 0.5, 0.2, 0.0,
    0.0, 0.2, 0.5, 1.0, 0.5, 0.2,
    0.2, 0.0, 0.2, 0.5, 1.0, 0.5,
    0.5, 0.2, 0.0, 0.2, 0.5, 1.0
  ),
  nrow = perm_size, byrow = TRUE
)
# sigma1 and sigma2 are matrices invariant under permutation (1,2,3,4,5,6)
numbers_of_observations <- c(21, 37)
Z1 <- MASS::mvrnorm(numbers_of_observations[1], mu = mu1, Sigma = sigma1)
Z2 <- MASS::mvrnorm(numbers_of_observations[2], mu = mu2, Sigma = sigma2)
S1 <- cov(Z1)
S2 <- cov(Z2) # Assume we have to estimate the mean

g <- gipsmult(list(S1, S2), numbers_of_observations)
```

```
g_map <- find_MAP(g, max_iter = 5, show_progress_bar = FALSE, optimizer = "Metropolis_Hastings")
g_map

g_map2 <- find_MAP(g_map, max_iter = 5, show_progress_bar = FALSE, optimizer = "continue")

if (require("graphics")) {
  plot(g_map2, type = "both", logarithmic_x = TRUE)
}

g_map_BF <- find_MAP(g, show_progress_bar = FALSE, optimizer = "brute_force")
```

`get_probabilities_from_gipsmult` *Extract probabilities for* `gipsmult` *object optimized with* `return_probabilities = TRUE`

Description

After the `gipsmult` object was optimized with the `find_MAP(return_probabilities = TRUE)` function, then those calculated probabilities can be extracted with this function.

Usage

```
get_probabilities_from_gipsmult(g)
```

Arguments

`g`	An object of class `gipsmult`. A result of a `find_MAP(return_probabilities = TRUE)`.

Value

Returns a numeric vector, calculated values of probabilities. Names contain permutations this probabilities represent. For `gipsmult` object optimized with `find_MAP(return_probabilities = FALSE)`, it returns a `NULL` object. It is sorted according to the probability.

See Also

- `find_MAP()` - The `get_probabilities_from_gipsmult()` is called on the output of `find_MAP(return_probabiliti = TRUE, save_all_perms = TRUE)`.

Examples

```
Ss <- list(
  matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE),
  matrix(c(2, 1, 3, 7), nrow = 2, byrow = TRUE)
)
noo <- c(10, 13)
```

```
g <- gipsmult(Ss, noo)
g_map <- find_MAP(g,
  optimizer = "BF", show_progress_bar = FALSE,
  return_probabilities = TRUE, save_all_perms = TRUE
)

get_probabilities_from_gipsmult(g_map)
```

`gipslda` *Linear Discriminant Analysis with gips Covariance Projection*

Description

Linear discriminant analysis (LDA) using covariance matrices projected via the *gips* framework to enforce permutation symmetry and improve numerical stability.

Usage

```
gipslda(x, ...)

## S3 method for class 'formula'
gipslda(formula, data, ..., subset, na.action)

## Default S3 method:
gipslda(x, grouping, prior = proportions,
  tol = 1e-4, weighted_avg = FALSE,
  MAP = TRUE, optimizer = NULL, max_iter = NULL, ...)

## S3 method for class 'data.frame'
gipslda(x, ...)

## S3 method for class 'matrix'
gipslda(x, grouping, ..., subset, na.action)
```

Arguments

`x`	(required if no formula is given as the principal argument) a matrix or data frame or Matrix containing the explanatory variables.
`...`	Arguments passed to or from other methods.
`formula`	A formula of the form `groups ~ x1 + x2 + ...`. The response is the grouping factor and the right-hand side specifies the (non-factor) discriminators.
`data`	An optional data frame, list or environment from which variables specified in `formula` are preferentially taken.
`grouping`	(required if no formula principal argument is given) a factor specifying the class for each observation.
`prior`	The prior probabilities of class membership. If unspecified, the class proportions for the training set are used.

tol	A tolerance to decide if a matrix is singular; variables whose variance is less than tol^2 are rejected.
subset	An index vector specifying the cases to be used in the training sample. (NOTE: must be named.)
na.action	A function specifying the action for NAs. #' @param weighted_avg Logical; if TRUE, uses a weighted average of class-specific covariance matrices instead of the pooled covariance.
MAP	Logical; whether to compute a Maximum A Posteriori gips projection of the covariance matrix.
optimizer	Character; optimization method used by gips (e.g. "BF" or "MH").
max_iter	Maximum number of iterations for the optimizer.
weighted_avg	Logical; Whether to compute scatter from all classes at once or to compute them within classes and compute the main one as average weighted by class proportions.

Details

This function is a minor modification of lda, replacing the classical sample covariance estimators by projected covariance matrices obtained using project_covs().

Unlike classical LDA, the within-class covariance matrix is first projected onto a permutation-invariant structure using the gips framework. This can stabilize covariance estimation in high dimensions or when symmetry assumptions are justified.

The choice of optimizer and MAP estimation affects both the covariance estimate and the resulting discriminant directions.

See Chojecki et al. (2025) for theoretical background.

Value

An object of class "gipslda" containing:

- prior: prior class probabilities
- counts: number of observations per class
- means: group means
- scaling: linear discriminant coefficients
- svd: singular values of the between-class scatter
- N: number of observations
- optimization_info: information about the gips optimization
- call: matched call

Note

This function is inspired by lda but is not a drop-in replacement. The covariance estimator, optimization procedure, and returned object differ substantially.

References

Chojecki, A., et al. (2025). *Learning Permutation Symmetry of a Gaussian Vector with gips in R*. Journal of Statistical Software, **112**(7), 1–38. doi:10.18637/jss.v112.i07

See Also

lda, gips

Examples

```
Iris <- data.frame(rbind(iris3[, , 1], iris3[, , 2], iris3[, , 3]),
  Sp = rep(c("s", "c", "v"), rep(50, 3))
)
train <- sample(1:150, 75)
z <- gipslda(Sp ~ ., Iris, prior = c(1, 1, 1) / 3, subset = train)
predict(z, Iris[-train, ])$class
(z1 <- update(z, . ~ . - Petal.W.))
```

`gipsmult` *The constructor of a* `gipsmult` *class.*

Description

Create a `gipsmult` object. This object will contain initial data and all other information needed to find the most likely invariant permutation. It will not perform optimization. One must call the `find_MAP()` function to do it. See the examples below.

Usage

```
gipsmult(
  Ss,
  numbers_of_observations,
  delta = 3,
  D_matrices = NULL,
  was_mean_estimated = TRUE,
  perm = ""
)

new_gipsmult(
  list_of_gips_perm,
  Ss,
  numbers_of_observations,
  delta,
  D_matrices,
  was_mean_estimated,
  optimization_info
)
```

Arguments

`Ss` A list of matrices; empirical covariance matrices. When `Z` is the observed data from single class:

- if one does not know the theoretical mean and has to estimate it with the observed mean, use `S = cov(Z)`, and leave parameter `was_mean_estimated = TRUE` as default;
- if one know the theoretical mean is 0, use `S = (t(Z) %*% Z) / number_of_observations`, and set parameter `was_mean_estimated = FALSE`.

`numbers_of_observations` Numbers of data points that `Ss` is based on.

`delta` A number, hyper-parameter of a Bayesian model. It has to be strictly bigger than 1. See the **Hyperparameters** section below.

`D_matrices` A list of symmetric, positive-definite matrices of the same size as matrices in `Ss`. Hyper-parameter of a Bayesian model. When `NULL`, the (hopefully) reasonable one is derived from the data. For more details, see the **Hyperparameters** section below.

`was_mean_estimated` A boolean.

- Set `TRUE` (default) when your `S` parameter is a result of a `stats::cov()` function.
- Set FALSE when your `S` parameter is a result of a `(t(Z) %*% Z) / number_of_observations` calculation.

`perm` An optional permutation to be the base for the `gipsmult` object. It can be of a `gips_perm` or a `permutation` class, or anything the function `permutations::permutation()` can handle. It can also be of a `gipsmult` class, but it will be interpreted as the underlying `gips_perm`.

`list_of_gips_perm` A list with a single element of a `gips_perm` class. The base object for the `gipsmult` object.

`optimization_info` For internal use only. `NULL` or the list with information about the optimization process.

Value

`gipsmult()` returns an object of a `gipsmult` class after the safety checks.

`new_gipsmult()` returns an object of a `gipsmult` class without the safety checks.

Functions

- `new_gipsmult()`: Constructor. It is only intended for low-level use.

Methods for a `gipsmult` class

- `plot.gipsmult()`
- `print.gipsmult()`

Hyperparameters

We encourage the user to try `D_matrix = d * I`, where `I` is an identity matrix of a size `p x p` and `d > 0` for some different `d`. When `d` is small compared to the data (e.g., `d = 0.1 * mean(diag(S))`), bigger structures will be found. When `d` is big compared to the data (e.g., `d = 100 * mean(diag(S))`), the posterior distribution does not depend on the data.

Taking `D_matrix = d * I` is equivalent to setting `S <- S / d`.

The default for `D_matrix` is `D_matrix = d * I`, where `d = mean(diag(S))`, which is equivalent to modifying `S` so that the mean value on the diagonal is 1.

In the Bayesian model, the prior distribution for the covariance matrix is a generalized case of Wishart distribution.

See Also

- `stats::cov()` – The `Ss` parameter, as a list of empirical covariance matrices, is most of the time a result of the `cov()` function. For more information, see Wikipedia - Estimation of covariance matrices.
- `find_MAP()` – The function that finds the Maximum A Posteriori (MAP) Estimator for a given `gipsmult` object.
- `gips::gips_perm()` – The constructor of a `gips_perm` class. The `gips_perm` object is used as the base object for the `gipsmult` object.

Examples

```
perm_size <- 5
numbers_of_observations <- c(15, 18, 19)
Sigma <- diag(rep(1, perm_size))
n_matrices <- 3
df <- 20
Ss <- rWishart(n = n_matrices, df = df, Sigma = Sigma)
Ss <- lapply(1:n_matrices, function(x) Ss[, , x])
g <- gipsmult(Ss, numbers_of_observations)

g_map <- find_MAP(g, show_progress_bar = FALSE, optimizer = "brute_force")
g_map

print(g_map)

if (require("graphics")) {
  plot(g_map, type = "MLE", logarithmic_x = TRUE)
}
```

`gipsmultqda` *Quadratic Discriminant Analysis with multiple gips-projected covariances*

Description

Quadratic Discriminant Analysis (QDA) in which each class covariance matrix is projected using the *gipsmult* framework, allowing for structured permutation symmetry across multiple covariance matrices.

Usage

```
gipsmultqda(x, ...)

## S3 method for class 'formula'
gipsmultqda(formula, data, ..., subset, na.action)

## Default S3 method:
gipsmultqda(x, grouping, prior = proportions,
  nu = 5, MAP = TRUE, optimizer = NULL, max_iter = NULL, ...)

## S3 method for class 'data.frame'
gipsmultqda(x, ...)

## S3 method for class 'matrix'
gipsmultqda(x, grouping, ..., subset, na.action)
```

Arguments

x	(required if no formula is given as the principal argument) a matrix or data frame containing the explanatory variables.
...	Arguments passed to or from other methods.
formula	A formula of the form groups ~ x1 + x2 + ... The response is the grouping factor and the right-hand side specifies the (non-factor) discriminators.
data	An optional data frame, list or environment from which variables specified in formula are preferentially taken.
grouping	A factor specifying the class for each observation.
prior	Prior probabilities of class membership. Must sum to one.
nu	Degrees of freedom parameter used internally during covariance projection.
MAP	Logical; if TRUE, a maximum a posteriori covariance projection is used.
optimizer	Character string specifying the optimization method used for covariance projection. If NULL, a default choice is made based on the problem dimension.
max_iter	Maximum number of iterations for stochastic optimizers.
subset	An index vector specifying the cases to be used in the training sample. (NOTE: must be named.)
na.action	A function specifying the action to be taken if NAs are found.

Details

This function is a modification of qda in which the class-specific covariance matrices are jointly projected to improve numerical stability and exploit shared symmetry assumptions.

In contrast to classical QDA, which estimates each class covariance matrix independently, gipsmultqda performs a joint projection of all class covariance matrices using the *gipsmult* framework. This allows the incorporation of shared permutation symmetries and can improve classification performance in high-dimensional or small-sample regimes.

Several classification rules are available via predict.gipsmultqda, including plug-in, predictive, debiased, and leave-one-out cross-validation.

Value

An object of class "gipsmultqda" containing:

- prior: prior probabilities of the groups
- counts: number of observations per group
- means: group means
- scaling: array of group-specific scaling matrices derived from the projected covariance matrices
- ldet: log-determinants of the projected covariance matrices
- lev: class labels
- N: total number of observations
- optimization_info: information returned by the covariance projection optimizer
- call: the matched call

Note

This function is not a drop-in replacement for qda. The covariance estimation, returned object, and classification rules differ substantially.

The theoretical background and details of the covariance projection are documented in the gipsmult package.

See Also

qda, predict.gipsmultqda, gipsqda, gipslda

Examples

```
tr <- sample(1:50, 25)
train <- rbind(iris3[tr, , 1], iris3[tr, , 2], iris3[tr, , 3])
test <- rbind(iris3[-tr, , 1], iris3[-tr, , 2], iris3[-tr, , 3])
cl <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25)))
z <- gipsmultqda(train, cl)
predict(z, test)$class
```

gipsqda *Quadratic Discriminant Analysis with gips covariance projection*

Description

Quadratic discriminant analysis (QDA) using covariance matrices projected via the *gips* framework to enforce permutation symmetry and improve numerical stability.

Usage

```
gipsqda(x, ...)

## S3 method for class 'formula'
gipsqda(formula, data, ..., subset, na.action)

## Default S3 method:
gipsqda(x, grouping, prior = proportions,
  nu = 5, MAP = TRUE, optimizer = NULL, max_iter = NULL, ...)

## S3 method for class 'data.frame'
gipsqda(x, ...)

## S3 method for class 'matrix'
gipsqda(x, grouping, ..., subset, na.action)
```

Arguments

x	(required if no formula is given as the principal argument) a matrix or data frame containing the explanatory variables.
...	Arguments passed to or from other methods.
formula	A formula of the form groups ~ x1 + x2 + ... The response is the grouping factor and the right-hand side specifies the (non-factor) discriminators.
data	An optional data frame, list or environment from which variables specified in formula are preferentially taken.
grouping	(required if no formula is given) a factor specifying the class for each observation.
prior	The prior probabilities of class membership. Must sum to one and have length equal to the number of groups.
nu	Degrees of freedom parameter used internally by covariance projection.
MAP	Logical; if TRUE, maximum a posteriori covariance projection is used.
optimizer	Character string specifying the optimization method used for covariance projection. If NULL, a default choice depending on the problem dimension is used.
max_iter	Maximum number of iterations for stochastic optimizers.
subset	An index vector specifying the cases to be used in the training sample. (NOTE: must be named.)
na.action	A function specifying the action to be taken if NAs are found.

Details

This function is a minor modification of qda, replacing the classical sample covariance estimators by projected covariance matrices obtained using `project_covs()`.

Quadratic discriminant analysis models each class with its own covariance matrix. In `gipsqda`, these covariance matrices are projected using the *gips* framework, which enforces permutation symmetry and mitigates singularity and overfitting in high-dimensional or small-sample settings.

Classification can be performed using plug-in, predictive, debiased, or leave-one-out cross-validation rules via `predict.gipsqda`.

Value

An object of class `"gipsqda"` containing the following components:

- `prior`: prior probabilities of the groups
- `counts`: number of observations in each group
- `means`: group means
- `scaling`: group-specific scaling matrices derived from the projected covariance matrices
- `ldet`: log-determinants of the projected covariance matrices
- `lev`: class labels
- `N`: total number of observations
- `optimization_info`: information returned by the covariance projection optimizer
- `call`: the matched call

Note

The function may be called with either a formula interface or with a matrix and grouping factor. Arguments `subset` and `na.action`, if used, must be named.

References

Chojecki, A., et al. (2025). *Learning Permutation Symmetry of a Gaussian Vector with gips in R*. Journal of Statistical Software, **112**(7), 1–38. doi:10.18637/jss.v112.i07

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Fourth edition. Springer.

See Also

qda, `predict.gipsqda`, gipslda, lda

Examples

```
tr <- sample(1:50, 25)
train <- rbind(iris3[tr, , 1], iris3[tr, , 2], iris3[tr, , 3])
test <- rbind(iris3[-tr, , 1], iris3[-tr, , 2], iris3[-tr, , 3])
cl <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25)))
z <- gipsqda(train, cl)
predict(z, test)$class
```

`log_posteriori_of_gipsmult` *A log of a posteriori that the covariance matrix is invariant under permutation*

Description

More precisely, it is the logarithm of an unnormalized posterior probability. It is the goal function for optimization algorithms in the `find_MAP()` function. The `perm_proposal` that maximizes this function is the Maximum A Posteriori (MAP) Estimator.

Usage

```
log_posteriori_of_gipsmult(g)
```

Arguments

`g`	An object of a `gipsmult` class.

Details

It is calculated using formulas (33) and (27) from references.

If `Inf` or `NaN` is reached, it produces a warning.

Value

Returns a value of the logarithm of an unnormalized A Posteriori.

References

Piotr Graczyk, Hideyuki Ishi, Bartosz Kołodziejek, Hélène Massam. "Model selection in the space of Gaussian models invariant by symmetry." The Annals of Statistics, 50(3) 1747-1774 June 2022. arXiv link; doi:10.1214/22AOS2174

See Also

- `find_MAP()` - The function that optimizes the `log_posteriori_of_gips` function.
- `gips::compare_posteriories_of_perms()` - Uses `log_posteriori_of_gips()` to compare a posteriori of two permutations.

Examples

```
# In the space with p = 2, there is only 2 permutations:
perm1 <- permutations::as.cycle("(1)(2)")
perm2 <- permutations::as.cycle("(1,2)")
S1 <- matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE)
S2 <- matrix(c(2, 1, 3, 7), nrow = 2, byrow = TRUE)
g1 <- gipsmult(list(S1, S2), c(100, 100), perm = perm1)
```

```
g2 <- gipsmult(list(S1, S2), c(100, 100), perm = perm2)
log_posteriori_of_gipsmult(g1) # -354.4394, this is the MAP Estimator
log_posteriori_of_gipsmult(g2) # -380.0079

exp(log_posteriori_of_gipsmult(g1) - log_posteriori_of_gipsmult(g2)) # 127131902082
# g1 is 127131902082 times more likely than g2.
# This is the expected outcome because S1[1,1] and S2[1,1]
# differ significantly from S1[2,2] and S2[2,2] respectively.

# ============================================================================

S3 <- matrix(c(1, 0.5, 0.5, 1.1), nrow = 2, byrow = TRUE)
S4 <- matrix(c(2, 1, 3, 2.137), nrow = 2, byrow = TRUE)
g1 <- gipsmult(list(S3, S4), c(100, 100), perm = perm1)
g2 <- gipsmult(list(S3, S4), c(100, 100), perm = perm2)
log_posteriori_of_gipsmult(g1) # -148.6485
log_posteriori_of_gipsmult(g2) # -145.3019, this is the MAP Estimator

exp(log_posteriori_of_gipsmult(g2) - log_posteriori_of_gipsmult(g1)) # 28.406
# g2 is 28.406 times more likely than g1.
# This is the expected outcome because S1[1,1] and S2[1,1]
# are very close to S1[2,2] and S2[2,2] respectively.
```

`plot.gipsmult` *Plot optimized matrix or optimization* `gipsmult` *object*

Description

Plot heatmaps of the MAP covariance matrices estimator or the convergence of the optimization method. The plot depends on the `type` argument.

Usage

```
## S3 method for class 'gipsmult'
plot(
  x,
  type = NA,
  logarithmic_y = TRUE,
  logarithmic_x = FALSE,
  color = NULL,
  title_text = "Convergence plot",
  xlabel = NULL,
  ylabel = NULL,
  show_legend = TRUE,
  ylim = NULL,
  xlim = NULL,
  ...
)
```

Arguments

x	Object of a gipsmult class.
type	A character vector of length 1. One of c("heatmap", "MLE", "best", "all", "both", "n0", "block_heatmap"): • "heatmap", "MLE" - Plots heatmaps of the Maximum Likelihood Estimator of the covariance matrices given the permutation. That is, the Ss matrices inside the gipsmult object projected on the permutation in the gipsmult object. • "best" - Plots the line of the biggest a posteriori found over time. • "all" - Plots the line of a posteriori for all visited states. • "both" - Plots both lines from "all" and "best". • "n0" - Plots the line of n0s that were spotted during optimization (only for "MH" optimization). • "block_heatmap" - Plots heatmaps of diagonally block representation of Ss. Non-block entries (equal to 0) are white for better clarity. The default value is NA, which will be changed to "heatmap" for non-optimized gipsmult objects and to "both" for optimized ones. Using the default produces a warning. All other arguments are ignored for the type = "heatmap", type = "MLE", or type = "block_heatmap".
logarithmic_y, logarithmic_x	A boolean. Sets the axis of the plots in logarithmic scale.
color	Vector of colors to be used to plot lines.
title_text	Text to be in the title of the plot.
xlabel	Text to be on the bottom of the plot.
ylabel	Text to be on the left of the plot.
show_legend	A boolean. Whether or not to show a legend.
ylim	Limits of the y axis. When NULL, the minimum, and maximum of the log_posteriori_of_gipsmult() are taken.
xlim	Limits of the x axis. When NULL, the whole optimization process is shown.
...	Additional arguments passed to other various elements of the plot.

Value

When type is one of "best", "all", "both" or "n0", returns an invisible NULL. When type is one of "heatmap", "MLE" or "block_heatmap", returns an object of class ggplot.

See Also

- find_MAP() - Usually, the plot.gipsmult() is called on the output of find_MAP().
- gipsmult() - The constructor of a gipsmult class. The gipsmult object is used as the x parameter.

Examples

```
require("MASS") # for mvrnorm()

perm_size <- 6
mu1 <- runif(6, -10, 10)
mu2 <- runif(6, -10, 10) # Assume we don't know the means
sigma1 <- matrix(
  data = c(
    1.0, 0.8, 0.6, 0.4, 0.6, 0.8,
    0.8, 1.0, 0.8, 0.6, 0.4, 0.6,
    0.6, 0.8, 1.0, 0.8, 0.6, 0.4,
    0.4, 0.6, 0.8, 1.0, 0.8, 0.6,
    0.6, 0.4, 0.6, 0.8, 1.0, 0.8,
    0.8, 0.6, 0.4, 0.6, 0.8, 1.0
  ),
  nrow = perm_size, byrow = TRUE
)
sigma2 <- matrix(
  data = c(
    1.0, 0.5, 0.2, 0.0, 0.2, 0.5,
    0.5, 1.0, 0.5, 0.2, 0.0, 0.2,
    0.2, 0.5, 1.0, 0.5, 0.2, 0.0,
    0.0, 0.2, 0.5, 1.0, 0.5, 0.2,
    0.2, 0.0, 0.2, 0.5, 1.0, 0.5,
    0.5, 0.2, 0.0, 0.2, 0.5, 1.0
  ),
  nrow = perm_size, byrow = TRUE
)
# sigma1 and sigma2 are matrices invariant under permutation (1,2,3,4,5,6)
numbers_of_observations <- c(21, 37)
Z1 <- MASS::mvrnorm(numbers_of_observations[1], mu = mu1, Sigma = sigma1)
Z2 <- MASS::mvrnorm(numbers_of_observations[2], mu = mu2, Sigma = sigma2)
S1 <- cov(Z1)
S2 <- cov(Z2) # Assume we have to estimate the mean

g <- gipsmult(list(S1, S2), numbers_of_observations)
if (require("graphics")) {
  plot(g, type = "MLE")
}

g_map <- find_MAP(g, max_iter = 30, show_progress_bar = FALSE, optimizer = "hill_climbing")
if (require("graphics")) {
  plot(g_map, type = "both", logarithmic_x = TRUE)
}

if (require("graphics")) {
  plot(g_map, type = "MLE")
}
# Now, the output is (most likely) different because the permutation
# `g_map[[1]]` is (most likely) not an identity permutation.

g_map_MH <- find_MAP(g, max_iter = 30, show_progress_bar = FALSE, optimizer = "MH")
```

```
if (require("graphics")) {
  plot(g_map_MH, type = "n0")
}
```

`print.gipsmult` *Printing* `gipsmult` *object*

Description

Printing function for a `gipsmult` class.

Usage

```
## S3 method for class 'gipsmult'
print(
  x,
  digits = 3,
  compare_to_original = TRUE,
  log_value = FALSE,
  oneline = FALSE,
  ...
)
```

Arguments

`x`	An object of a `gipsmult` class.
`digits`	The number of digits after the comma for a posteriori to be presented. It can be negative. By default, `Inf`. It is passed to `base::round()`.
`compare_to_original`	A logical. Whether to print how many times more likely is the current permutation compared to: • the identity permutation `()` (for unoptimized `gipsmult` object); • the starting permutation (for optimized `gipsmult` object).
`log_value`	A logical. Whether to print the logarithmic value. Default to `FALSE`.
`oneline`	A logical. Whether to print in one or multiple lines. Default to `FALSE`.
`...`	The additional arguments passed to `base::cat()`.

Value

Returns an invisible `NULL`.

See Also

- `find_MAP()` - The function that makes an optimized `gipsmult` object out of the unoptimized one.

Examples

```
Ss <- list(
  matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE),
  matrix(c(2, 1, 3, 7), nrow = 2, byrow = TRUE)
)
noo <- c(10, 13)
g <- gipsmult(Ss, noo, perm = "(12)")
print(g, digits = 4, oneline = TRUE)
```

Index