Package 'gips'

October 13, 2022

Type Package

Title Gaussian Model Invariant by Permutation Symmetry

Version 1.0.0

Description Find the permutation symmetry group such that the covariance matrix of the given data is invariant under it. Discovering such a permutation decreases the number of observations needed to fit a Gaussian model, which is of great use when it is smaller than the number of variables. Even if that is not the case, the covariance matrix found with 'gips' approximates the actual covariance with less statistical error. The methods implemented in this package are described in Graczyk et al. (2022) <doi:10.1214/22-AOS2174>.

License GPL (>= 3)

URL `https://github.com/PrzeChoj/gips`, `https://przechoj.github.io/gips/`

BugReports `https://github.com/PrzeChoj/gips/issues`

Depends R (>= 3.5.0)

Imports numbers, permutations, rlang, utils

Suggests DAAG, dplyr, ggplot2, graphics, HSAUR, knitr, MASS (>= 7.3-39), rmarkdown, spelling, stringi, testthat (>= 3.0.0), tibble, tidyr, withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.2.1

NeedsCompilation no

Author Przemysław Chojecki [aut, cre], Paweł Morgen [aut], Bartosz Kołodziejek [aut] (<`https://orcid.org/0000-0002-5220-9012`>)

Maintainer Przemysław Chojecki <`premysl.choj@gmail.com`>

Repository CRAN

Date/Publication 2022-10-13 17:42:33 UTC

R topics documented:

`as.character.gips_perm`

Transform `gips_perm` *object to character vector*

Description

Implementation of S3 method.

Usage

```
## S3 method for class 'gips_perm'
as.character(x, ...)
```

Arguments

`x`	An object of a `gips_perm` class.
`...`	Further arguments passed to `permutations::as.character.cycle()`.

Value

Returns an object of a `character` type.

Methods (by class)

- `as.character(gips_perm)`:

See Also

permutations::as.character.cycle()

Examples

```
g_perm <- gips_perm(permutations::as.cycle("(5,4)"), 5)
as.character(g_perm)
```

`calculate_gamma_function` *Calculate Gamma function*

Description

It calculates the value of the integral defined in Definition 11 from references. It is implementation of the Theorem 8 from references and is using the formula (19) from references.

Usage

```
calculate_gamma_function(perm, lambda)
```

Arguments

`perm`	An object of a `gips_perm` class.
`lambda`	A positive real number.

Value

Returns the value of the Gamma function of the colored cone (for definition of colored cone see **Basic definitions** section in `vignette("Theory", package = "gips")` or in its pkgdown page).

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

- `get_structure_constants()` - The function useful inside the `calculate_gamma_function()` function.
- `log_posteriori_of_gips()` - The function that uses the values of the gamma function calculable with `calculate_gamma_function()`.
- `vignette("Theory", package = "gips")` or its pkgdown page - A place to learn more about the math behind the `gips` package.

Examples

```
id_perm <- gips_perm(permutations::id, 2)
calculate_gamma_function(id_perm, 0.5001) # 10.7...
calculate_gamma_function(id_perm, 0.50000001) # 19.9...
calculate_gamma_function(id_perm, 0.500000000001) # 29.1...

try(calculate_gamma_function(id_perm, 0.5))
# Error, integral diverges; returns Inf and warning
```

---

compare_posteriories_of_perms

Compare the posteriori probabilities of 2 permutations

Description

Check which permutation is more likely and how much more likely.

Usage

```
compare_posteriories_of_perms(
  perm1,
  perm2 = "()",
  S = NULL,
  number_of_observations = NULL,
  delta = 3,
  D_matrix = NULL,
  was_mean_estimated = TRUE,
  print_output = TRUE
)

compare_log_posteriories_of_perms(
  perm1,
  perm2 = "()",
  S = NULL,
  number_of_observations = NULL,
  delta = 3,
  D_matrix = NULL,
  was_mean_estimated = TRUE,
  print_output = TRUE
)
```

Arguments

`perm1, perm2`	Permutations to compare. How many times `perm1` is more likely than `perm2`? Those can be provided as the `gips` object, the `gips_perm` object or anything that can be used as the `x` parameter in the `gips_perm()` function. They do not have to be of the same class.

`S`, `number_of_observations`, `delta`, `D_matrix`, `was_mean_estimated`
: The same parameters as in the `gips()` function. If at least one of `perm1` or `perm2` is of a `gips` class, they overwritten with those from `gips` object.

`print_output`
: A boolean. When TRUE, the computed value will be printed with additional text and returned invisibly. When FALSE, the computed value will be returned visibly.

Value

`compare_posteriories_of_perms` returns the value of how many times the `perm1` is more likely than `perm2`.

`compare_log_posteriories_of_perms` returns the logarithm of how many times the `perm1` is more likely than `perm2`.

Functions

- `compare_log_posteriories_of_perms()`: More stable, logarithmic version of `compare_posteriories_of_perms`. The natural logarithm is used.

See Also

- `print.gips()` - The function that prints the posterior of the optimized `gips` object compared to the starting permutation.
- `summary.gips()` - The function that calculates the posterior of the optimized `gips` object compared to the starting permutation.

Examples

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

perm_size <- 6
mu <- runif(6, -10, 10) # Assume we don't know the mean
sigma_matrix <- 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
) # sigma_matrix is a matrix invariant under permutation (1,2,3,4,5,6)
number_of_observations <- 13
Z <- MASS::mvrnorm(number_of_observations, mu = mu, Sigma = sigma_matrix)
S <- cov(Z) # Assume we have to estimate the mean

g <- gips(S, number_of_observations)
g_map <- find_MAP(g, max_iter = 10, show_progress_bar = FALSE, optimizer = "Metropolis_Hastings")
```

```
compare_posteriories_of_perms(g_map, g, print_output = FALSE)
compare_log_posteriories_of_perms(g_map, g, print_output = FALSE)
```

---

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. More information can be found in the "**Possible algorithms to use as optimizers**" section below.

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 `gips` class

`max_iter` Number of iterations for an algorithm to perform. At least 2. For `optimizer="MH"` it has to be finite; for `optimizer="HC"` it can be infinite; for `optimizer="BF"` it is not used.

`optimizer` The optimizer for the search of the maximum posteriori.

- `"MH"` (the default for unoptimized `g`) - Metropolis-Hastings
- `"HC"` - Hill Climbing
- `"BF"` - Brute Force
- `"continue"` (the default for optimized `g`) - The same as the `g` was optimized by (see Examples).

For more details, see the "**Possible algorithms to use as optimizers**" section below.

`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 to save a list of all permutations that were visited during optimization. This can be useful, but need a lot more RAM.

`return_probabilities`
: A boolean. `TRUE` can only be provided when `save_all_perms` is `TRUE` and for:

  - `optimizer="MH"` - use Metropolis-Hastings results to estimate posterior probabilities
  - `optimizer="BF"` - use brute force results to calculate exact posterior probabilities

  This additional calculations are costly, so second progress bar is shown (when `show_progress_bar` is `TRUE`).

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

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` (for more information on what it means, see $C\sigma$ **and** `n0` section in `vignette("Theory", package = "gips")` or in its pkgdown page).

Value

Returns an optimized object of a `gips` class.

Possible algorithms to use as optimizers

For a more in-depth explanations, see `vignette("Optimizers", package = "gips")` or in its pkgdown page).

For every algorithm, there are some aliases available.

- `"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 so far as the MAP Estimator. This implements the *Second approach* from references, section 4.1.2. This algorithm used in this context is a special case of the **Simulated Annealing** the reader 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 start the optimization again on the output of the `find_MAP()`. Remember that there are `p*(p-1)/2` transpositions to be checked in every iteration. For bigger `p`, this may be costly.
- `"brute_force"`, `"BF"`, `"full"` - use the **Brute Force** algorithm that checks the whole permutation space of a given size. This algorithm will definitely find the actual Maximum A Posteriori Estimation but is very computationally expensive for bigger spaces.

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

- `gips()` - The constructor of a `gips` class. The `gips` object is used as the g parameter.
- `plot.gips()` - Practical plotting function for visualizing the optimization process.
- `summary.gips()` - The function that summarizes the output of optimization.
- `get_probabilities_from_gips()` - When `find_MAP(return_probabilities = TRUE)` was called, then those probabilities can be extracted with this function.
- `log_posteriori_of_gips()` - The function that the optimizers of `find_MAP()` tries to find the argmax of.
- `forget_perms()` - When the `gips` object was optimized with `find_MAP(save_all_perms = TRUE)`, it will be of considerable size in RAM. `forget_perms` can make such an object lighter in memory by forgetting the permutations that it was in.
- `vignette("Optimizers", package = "gips")` or its pkgdown page) - A place to learn more about the available optimizers.
- `vignette("Theory", package = "gips")` or its pkgdown page) - A place to learn more about the math behind the `gips` package.

Examples

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

perm_size <- 5
mu <- runif(perm_size, -10, 10) # Assume we don't know the mean
sigma_matrix <- matrix(
  data = c(
    1.0, 0.8, 0.6, 0.6, 0.8,
    0.8, 1.0, 0.8, 0.6, 0.6,
    0.6, 0.8, 1.0, 0.8, 0.6,
    0.6, 0.6, 0.8, 1.0, 0.8,
    0.8, 0.6, 0.6, 0.8, 1.0
  ),
  nrow = perm_size, byrow = TRUE
) # sigma_matrix is a matrix invariant under permutation (1,2,3,4,5)
number_of_observations <- 13
Z <- MASS::mvrnorm(number_of_observations, mu = mu, Sigma = sigma_matrix)
S <- cov(Z) # Assume we have to estimate the mean

g <- gips(S, number_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")
summary(g_map_BF)
```

---

forget_perms — *Forget the permutations for* gips *object optimized with* save_all_perms = TRUE

Description

Slim the gips object by forgetting the visited permutations from find_MAP(save_all_perms = TRUE).

Usage

```
forget_perms(g)
```

Arguments

g	An object of class "gips"; a result of a find_MAP(save_all_perms = TRUE).

Details

For perm_size = 150 and max_iter = 150000 we checked it saves ~350 MB.

Value

Returns the same object g as given, but without the visited permutation list.

See Also

- find_MAP() - The forget_perms() is called on the output of find_MAP(save_all_perms = TRUE).

Examples

```
example_matrix <- matrix(rnorm(10 * 10), nrow = 10)
S <- t(example_matrix) %*% example_matrix
g <- gips(S, 13, was_mean_estimated = FALSE)
g_map <- find_MAP(g,
  max_iter = 10, optimizer = "Metropolis_Hastings",
  show_progress_bar = FALSE, save_all_perms = TRUE
)

object.size(g_map) # ~18 KB
g_map_slim <- forget_perms(g_map)
object.size(g_map_slim) # ~8 KB
```

`get_probabilities_from_gips`

Extract probabilities for `gips` *object optimized with* `return_probabilities = TRUE`

Description

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

Usage

```
get_probabilities_from_gips(g)
```

Arguments

g	An object of class "gips"; a result of a `find_MAP(return_probabilities = TRUE)`.

Value

Returns a numeric vector, calculated values of probabilities. Names contains permutations this probability represent. For `gips` object optimized with `find_MAP(return_probabilities = FALSE)`, returns a `NULL` object.

See Also

- `find_MAP()` - The `get_probabilities_from_gips()` is called on the output of `find_MAP(return_probabilities = TRUE, save_all_perms = TRUE)`.
- `vignette("Optimizers", package = "gips")` or its pkgdown page) - A place to learn more about the available optimizers.

Examples

```
g <- gips(matrix(c(1, 0.5, 0.5, 1.3), nrow = 2), 13, was_mean_estimated = FALSE)
g_map <- find_MAP(g,
  optimizer = "BF", show_progress_bar = FALSE,
  return_probabilities = TRUE, save_all_perms = TRUE
)

get_probabilities_from_gips(g_map)
```

`get_structure_constants` *Get Structure Constants*

Description

Finds constants that are necessary for internal calculations of integrals and eventually the posteriori probability in `log_posteriori_of_gips()`.

Usage

```
get_structure_constants(perm)
```

Arguments

`perm`	An object of a `gips_perm` class.

Details

Uses the Theorem 5 from references to calculate the constants.

Value

Returns a list of 5 items: `r`, `d`, `k`, `L`, `dim_omega` - vectors of constants from Theorem 1 from references and the beginning of section 3.1. from references.

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

- `calculate_gamma_function()`, `log_posteriori_of_gips()` - The functions that rely heavily on `get_structure_constants()`.

Examples

```
perm <- gips_perm(permutations::as.word(c(1, 2, 3, 5, 4)), 5)
get_structure_constants(perm)
```

gips — *The constructor of a* `gips` *class.*

Description

Create a `gips` object. This object will consist of data and all other information needed to find the most likely invariant permutation. The optimization itself will not be performed. One must call the `find_MAP()` function to do it. See examples below.

Usage

```
gips(
  S,
  number_of_observations,
  delta = 3,
  D_matrix = NULL,
  was_mean_estimated = TRUE,
  perm = ""
)

new_gips(
  list_of_gips_perm,
  S,
  number_of_observations,
  delta,
  D_matrix,
  was_mean_estimated,
  optimization_info
)

validate_gips(g)
```

Arguments

`S` A matrix; estimated covariance matrix. When `Z` is the observed data:

- 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`;

`number_of_observations`
A number of data points that `S` is based on.

`delta` A number, hyper-parameter of a Bayesian model. Has to be bigger than 2. See **Hyperparameters** section bellow.

`D_matrix`	A symmetric, positive-definite matrix of the same size as `S`. Hyper-parameter of a Bayesian model. When `NULL`, the identity matrix is taken. See **Hyperparameters** section bellow.
`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 `gips` object. Can be of a `gips_perm` or a `permutation` class, or anything the function `permutations::permutation()` can handle.
`list_of_gips_perm`	A list with a single element of a `gips_perm` class. The base object for the `gips` object.
`optimization_info`	For internal use only. `NULL` or the list with information about the optimization process.
`g`	Object to be checked whether it is proper object of a `gips` class.

Value

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

`new_gips()` returns an object of a `gips` class without the safety checks.

`validate_gips()` returns its argument unchanged. If the argument is not a proper element of a `gips` class, it produces an error.

Functions

- `new_gips()`: Constructor. Only intended for low-level use.
- `validate_gips()`: Validator. Only intended for low-level use.

Methods for a `gips` class

- `summary.gips()`
- `plot.gips()`
- `print.gips()`

Hyperparameters

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

For brief introduction, see **Bayesian model selection** section in `vignette("Theory", package = "gips")` or in its pkgdown page).

See Also

- `stats::cov()` - The `S` parameter is most of the time an estimated covariance matrix, so 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 `gips` object.
- `gips_perm()` - The constructor of a `gips_perm` class. The `gips_perm` object is used as the base object for the `gips` object. To be more precise, the base object for `gips` is a one-element list of a `gips_perm` object.

Examples

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

perm_size <- 5
mu <- runif(5, -10, 10) # Assume we don't know the mean
sigma_matrix <- matrix(
  data = c(
    1.0, 0.8, 0.6, 0.6, 0.8,
    0.8, 1.0, 0.8, 0.6, 0.6,
    0.6, 0.8, 1.0, 0.8, 0.6,
    0.6, 0.6, 0.8, 1.0, 0.8,
    0.8, 0.6, 0.6, 0.8, 1.0
  ),
  nrow = perm_size, byrow = TRUE
) # sigma_matrix is a matrix invariant under permutation (1,2,3,4,5)
number_of_observations <- 13
Z <- MASS::mvrnorm(number_of_observations, mu = mu, Sigma = sigma_matrix)
S <- cov(Z) # Assume we have to estimate the mean

g <- gips(S, number_of_observations)

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

summary(g_map)

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

`gips_perm` *Permutation object*

Description

Create permutation objects to be passed to other functions of the `gips` package.

Usage

```
gips_perm(x, size)

new_gips_perm(rearranged_cycles, size)

validate_gips_perm(g)
```

Arguments

x	An object created with a `permutations` package or any object that can be processed with the `permutations::permutation()` function.
size	An integer. Size of a permutation (AKA cardinality of a set, on which permutation is defined; see examples).
rearranged_cycles	A list of rearranged integer vectors. Each vector corresponds to a single cycle of a permutation.
g	Object to be checked whether it is a proper object of a `gips_perm` class.

Value

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

`new_gips_perm()` returns an object of a `gips_perm` class without the safety checks.

`validate_gips_perm()` returns its argument unchanged. If the argument is not a proper element of a `gips_perm` class, it produces an error.

Functions

- `new_gips_perm()`: Constructor. Only intended for low-level use.
- `validate_gips_perm()`: Validator. Only intended for low-level use.

Methods for a `gips` class

- `as.character.gips_perm()`
- `print.gips_perm()`

See Also

- `permutations::permutation()` - The constructor for the `x` parameter.
- `gips()` - The constructor for the `gips` class uses the `gips_perm` object as the base object.

Examples

```
gperm <- gips_perm(permutations::as.word(c(1, 2, 3, 5, 4)), 5)
gperm <- gips_perm(permutations::as.cycle("(5,4)"), 5)
# note the necessity of `size` parameter
gperm <- gips_perm(permutations::as.cycle("(5,4)"), 7)
gperm <- gips_perm("(1,2)(5,4)", 7)
```

```
gperm


try(gperm <- gips_perm(permutations::as.cycle("(5,4)"), 3))
# Error, `size` equals 3 while the maximum element is 5.
```

log_posteriori_of_gips

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 `find_MAP()` function. The `perm_proposal` that maximizes this function is the Maximum A Posteriori (MAP) Estimator.

Usage

```
log_posteriori_of_gips(g)
```

Arguments

`g`	An object of a `gips_perm` 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

- `calculate_gamma_function()` - The function that calculates the value needed for `log_posteriori_of_gips()`.
- `find_MAP()` - The functions that tries to optimize the `log_posteriori_of_gips` function.
- `vignette("Theory", package = "gips")` or its pkgdown page - A place to learn more about the math behind the `gips` package.

Examples

```
# In the space with p = 2, there is only 2 permutations:
perm1 <- permutations::as.cycle(permutations::as.word(c(1, 2))) # (1)(2)
perm2 <- permutations::as.cycle(permutations::as.word(c(2, 1))) # (1,2)
S1 <- matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE)
g1 <- gips(S1, 100, perm = perm1)
g2 <- gips(S1, 100, perm = perm2)
log_posteriori_of_gips(g1) # -136.6, this is the MAP Estimator
log_posteriori_of_gips(g2) # -140.4

exp(log_posteriori_of_gips(g1) - log_posteriori_of_gips(g2)) # 41.3
# g1 is over 40 times more likely than g2.
# This is the expected outcome because S[1,1] significantly differs from S[2,2].

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

S2 <- matrix(c(1, 0.5, 0.5, 1.1), nrow = 2, byrow = TRUE)
g1 <- gips(S2, 100, perm = perm1)
g2 <- gips(S2, 100, perm = perm2)
log_posteriori_of_gips(g1) # -99.5
log_posteriori_of_gips(g2) # -96.9, this is the MAP Estimator

exp(log_posteriori_of_gips(g2) - log_posteriori_of_gips(g1)) # 12.7
# g2 is over 12 times more likely than g1.
# This is the expected outcome because S[1,1] is very close to S[2,2].
```

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

Description

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

Usage

```
## S3 method for class 'gips'
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 `gips` class.
`type`	A single character. One of `c("heatmap", "block_heatmap", "all", "best", "both")`. • "heatmap" - Plots a heatmap of the `S` matrix inside the `gips` object projected on the permutation in the `gips` object. • "block_heatmap" - Plots a heatmap of diagonally block representation of `S`. Non-block entries (equal to 0) are white for better clarity. For more information see **Block Decomposition - [1], Theorem 1** section in `vignette("Theory", package = "gips")` or in its pkgdown page). • "all" - Plots the line of a posteriori for all visited states. • "best" - Plots the line of the biggest a posteriori found over time. • "both" - Plots both lines from "all" and "best". The default value is `NA`, which will be changed to "heatmap" for non-optimized `gips` objects and to "both" for optimized ones. Using the default produces a warning. All other arguments are ignored for the `type = "heatmap"`.
`logarithmic_y, logarithmic_x`	A boolean. Sets the axis of the plot 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_gips()` are taken.
`xlim`	Limits of the x axis. When `NULL`, the whole optimization process is shown.
`...`	Additional arguments passed to `stats::heatmap()` or other various elements of the plot.

Value

Returns an invisible `NULL`.

See Also

- `find_MAP()` - Usually, the `plot.gips()` is called on the output of `find_MAP()`.
- `project_matrix()` - The function used with `type = "heatmap"`.
- `gips()` - The constructor of a `gips` class. The `gips` object is used as the `x` parameter.

Examples

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

perm_size <- 6
mu <- runif(6, -10, 10) # Assume we don't know the mean
sigma_matrix <- 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
) # sigma_matrix is a matrix invariant under permutation (1,2,3,4,5,6)
number_of_observations <- 13
Z <- MASS::mvrnorm(number_of_observations, mu = mu, Sigma = sigma_matrix)
S <- cov(Z) # Assume we have to estimate the mean

g <- gips(S, number_of_observations)
if (require("graphics")) {
  plot(g, type = "heatmap")
}

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 = "heatmap")
}
# Now, the output is (most likely) different because the permutation
  # `g_map[[1]]` is (most likely) not an identity permutation.
```

prepare_orthogonal_matrix

Prepare orthogonal matrix

Description

Calculate orthogonal matrix U_Gamma for decomposition in Theorem 1 from references.

Usage

```
prepare_orthogonal_matrix(perm, perm_size = NULL, basis = NULL)
```

Arguments

`perm`	An object of a `gips_perm` or a `permutations::cycle` class.
`perm_size`	Size of a permutation. Required if `perm` is of a `permutations::cycle` class.
`basis`	A matrix with basis vectors in COLUMNS. Identity by default.

Details

Given X - a matrix invariant under the permutation `perm`. Call Gamma the permutations cyclic group $< perm >= \{perm, perm^2, ...\}$.

Then, U_Gamma is such an orthogonal matrix, which block-diagonalizes X.

To be more precise, the matrix `t(U_Gamma) %*% X %*% U_Gamma` has a block-diagonal structure, which is ensured by Theorem 1 from references

Formula for U_Gamma can be found in Theorem 6 from references.

Value

A square matrix of size `perm_size` by `perm_size` with columns from vector elements $v_k^{(c)}$ according to Theorem 6 from references.

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

- `project_matrix()` - A function used in examples to show the properties of `prepare_orthogonal_matrix()`.
- **Block Decomposition - [1], Theorem 1** section of `vignette("Theory", package = "gips")` or its pkgdown page) - A place to learn more about the math behind the `gips` package and see more examples of `prepare_orthogonal_matrix()`.

Examples

```
gperm <- gips_perm("(1,2,3)(4,5)", 5)
U_Gamma <- prepare_orthogonal_matrix(gperm)

number_of_observations <- 10
X <- matrix(rnorm(5 * number_of_observations), number_of_observations, 5)
S <- cov(X)
X <- project_matrix(S, perm = gperm) # this matrix in invariant under gperm

block_decomposition <- t(U_Gamma) %*% X %*% U_Gamma
round(block_decomposition, 5) # the non-zeros only on diagonal and [1,2] and [2,1]
```

print.gips — *Printing* gips *object*

Description

Printing function for a gips class.

Usage

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

Arguments

x	An object of a gips 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 gips object); • the starting permutation (for optimized gips object).
log_value	A logical. Whether to print the value of a log_posteriori_of_gips(). 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 gips object out of the unoptimized one.
- compare_posteriories_of_perms() - The function that prints the compared posteriories between any two permutations, not only compared to the starting one or id.

Examples

```
S <- matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE)
g <- gips(S, 10)
print(g, digits = 4)
```

---

print.gips_perm — *Printing* gips_perm *object*

Description

Printing function for a gips_perm class.

Usage

```
## S3 method for class 'gips_perm'
print(x, ...)
```

Arguments

x	An object of a gips_perm class.
...	Further arguments passed to permutations::print.cycle().

Value

Returns its argument invisibly, after printing it.

Examples

```
g_perm <- gips_perm(permutations::as.cycle("(5,4)"), 5)
print(g_perm)
```

---

project_matrix — *Project matrix after optimization*

Description

After the MAP permutation was found with find_MAP(), use this permutation to approximate the covariance matrix with larger statistical confidence.

Usage

```
project_matrix(S, perm, precomputed_equal_indices = NULL)
```

Arguments

`S`	A square matrix to be projected. The covariance estimator. (See the same parameter in `gips()` function).
`perm`	A permutation. Generator of a permutation group. Either of a `gips_perm` or a `permutations::cycle` class.
`precomputed_equal_indices`	This parameter is for internal use only.

Details

Project matrix on the space of symmetrical matrices invariant by a cyclic group generated by `perm`. This implements the formal Definition 3 from references.

When `S` is the sample covariance matrix (output of `cov()` function, see examples), then `S` is the **unbiased estimator** of the covariance matrix. However, the **maximum likelihood estimator** of the covariance matrix is `S*(n-1)/(n)`, unless n < p, when the **maximum likelihood estimator does not exist**. For more information, see Wikipedia - Estimation of covariance matrices.

The maximum likelihood estimator differs when one knows the covariance matrix is **invariant under some permutation**. This estimator will not only be symmetric but also have some values repeated (see examples and Corollary 12 from references).

The estimator will be invariant under the given permutation. Also, it will **need fewer observations** for the maximum likelihood estimator to exist (see **Project Matrix - Equation (6)** section in `vignette("Theory", package = "gips")` or in its pkgdown page). For some permutations, even $n = 2$ could be enough. The minimal number of observations needed are named `n0` and can be calculated by `summary.gips()`.

For more details, see the **Project Matrix - Equation (6)** section in `vignette("Theory", package = "gips")` or in its pkgdown page.

Value

Returns the matrix `S` projected on the space of symmetrical matrices invariant by a cyclic group generated by `perm`. See Details for more.

See Also

- Wikipedia - Estimation of covariance matrices
- **Project Matrix - Equation (6)** section of `vignette("Theory", package = "gips")` or its pkgdown page - A place to learn more about the math behind the `gips` package and see more examples of `project_matrix()`.
- `find_MAP()` - The function that finds the Maximum A Posteriori (MAP) Estimator for a given `gips` object. After the MAP Estimator is found, the matrix `S` can be projected on this permutation, creating the MAP Estimator of the covariance matrix (see examples).
- `gips_perm()` - Constructor for the `perm` parameter.
- `plot.gips()` - For `plot(g, type = 'heatmap')`, the `project_matrix()` is called (see examples).
- `summary.gips()` - Can calculate the `n0`, the minimal number of observations, so that the projected matrix will be the MAP estimator of the covariance matrix.

Examples

```
p <- 6
gperm <- gips_perm(permutations::as.word(c(4, 3, 2, 1, 5)), p) # permutation (1,4)(2,3)(5)(6)

number_of_observations <- 10
X <- matrix(rnorm(p * number_of_observations), number_of_observations, p)
S <- cov(X)
projected_S <- project_matrix(S, perm = gperm)
projected_S
# The value in [1,1] is the same as in [4,4]; also, [2,2] and [3,3];
  # also [1,2] and [4,3]; also, [1,5] and [4,5]; and so on

# Plot the projected matrix:
g <- gips(S, number_of_observations, perm = gperm)
plot(g, type = "heatmap")

# Find the MAP Estimator
g_MAP <- find_MAP(g, max_iter = 10, show_progress_bar = FALSE, optimizer = "Metropolis_Hastings")
S_MAP <- project_matrix(S, perm = g_MAP[[1]])
S_MAP
plot(g_MAP, type = "heatmap")
```

---

summary.gips *Summarizing the gips object*

Description

summary method for class "gips".

Usage

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

## S3 method for class 'summary.gips'
print(x, ...)
```

Arguments

object	An object of class "gips"; is usually a result of a find_MAP().
...	Further arguments passed to or from other methods.
x	An object of class "summary.gips" to be printed

Value

The function summary.gips computes and returns a list of summary statistics of the given gips object. Those are:

- For unoptimized gips object:
  1. optimized - FALSE
  2. start_permutation - the permutation this gips represents
  3. start_permutation_log_posteriori - the log of the a posteriori value the start permutation has
  4. times_more_likely_than_id - how many more likely the start_permutation is over the identity permutation, (). It can be a number less than 1, which means the identity permutation, (), is more likely. Keep in mind this number can be really big and can be overflowed to Inf
  5. n0 - the minimal number of observations needed for existence of the maximum likelihood estimator (corresponding to a MAP) of the covariance matrix (see $C\sigma$ **and** n0 section in vignette("Theory", package = "gips") or in its pkgdown page).
  6. S_matrix - the underlying matrix; this is used to calculate the posteriori value
  7. number_of_observations - the number of observations that were observed for the S_matrix to be calculated; this is used to calculate the posteriori value
  8. was_mean_estimated - given by the user while creating the gips object:
     - TRUE means the S parameter was output of stats::cov() function
     - FALSE means the S parameter was calculated with S = t(X) %*% X / number_of_observations
  9. delta, D_matrix - the parameters of the Bayesian method
- For optimized gips object:
  1. optimized - TRUE
  2. found_permutation - the permutation this gips represents; the visited permutation with the biggest a posteriori value
  3. found_permutation_log_posteriori - the log of the a posteriori value the found permutation have
  4. start_permutation - the original permutation this gips represented before optimization; the first visited permutation
  5. start_permutation_log_posteriori - the log of the a posteriori value the start permutation has
  6. times_more_likely_than_start - how many more likely the found_permutation is over the start_permutation. It cannot be a number less than 1. Keep in mind this number can be really big and can be overflowed to Inf
  7. n0 - the minimal number of observations needed for existence of the maximum likelihood estimator (corresponding to a MAP) of the covariance matrix (see $C\sigma$ **and** n0 section in vignette("Theory", package = "gips") or in its pkgdown page).
  8. S_matrix - the underlying matrix; this is used to calculate the posteriori value
  9. number_of_observations - the number of observations that were observed for the S_matrix to be calculated; this is used to calculate the posteriori value
  10. was_mean_estimated - given by the user while creating the gips object:
      - TRUE means the S parameter was output of stats::cov() function
      - FALSE means the S parameter was calculated with S = t(X) %*% X / number_of_observations
  11. delta, D_matrix - the parameters of the Bayesian method
  12. optimization_algorithm_used - all used optimization algorithms in order (one could start optimization with "MH", and then do an "HC")

13. `did_converge` - a boolean, did the last used algorithm converge
14. `number_of_log_posteriori_calls` - how many times was the `log_posteriori_of_gips()` function called during the optimization
15. `whole_optimization_time` - how long was the optimization process; the sum of all optimization times (when there were multiple)
16. `log_posteriori_calls_after_best` - how many times was the `log_posteriori_of_gips()` function called after the `found_permutation`; in other words, how long ago could the optimization be stopped and have the same result; if this value is small, consider running `find_MAP()` one more time with `optimizer = "continue"`. For `optimizer = "BF"`, it is `NULL`
17. `acceptance_rate` - only interesting for `optimizer = "MH"`; how often was the algorithm accepting the change of permutation in an iteration

`print.summary.gips` returns an invisible `NULL`.

Methods (by generic)

- `print(summary.gips)`: Printing method for class "summary.gips". Prints every interesting information in a pleasant, human readable form

See Also

- `find_MAP()` - Usually, the `summary.gips()` is called on the output of `find_MAP()`.
- `log_posteriori_of_gips()` - The function that calculates the likelihood of a permutation.
- `project_matrix()` - The function that can project the known matrix of the found permutations space.

Examples

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

perm_size <- 6
mu <- runif(6, -10, 10) # Assume we don't know the mean
sigma_matrix <- 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
) # sigma_matrix is a matrix invariant under permutation (1,2,3,4,5,6)
number_of_observations <- 13
Z <- MASS::mvrnorm(number_of_observations, mu = mu, Sigma = sigma_matrix)
S <- cov(Z) # Assume we have to estimate the mean

g <- gips(S, number_of_observations)
```

```
g_map <- find_MAP(g, max_iter = 10, show_progress_bar = FALSE, optimizer = "Metropolis_Hastings")
unclass(summary(g_map))

g_map2 <- find_MAP(g, max_iter = 10, show_progress_bar = FALSE, optimizer = "hill_climbing")
summary(g_map2)
# ================================================================================
S <- matrix(c(1, 0.5, 0.5, 2), nrow = 2, byrow = TRUE)
g <- gips(S, 10)
print(summary(g))
```

Index