Package 'SpatialBSS'

March 27, 2025

Type Package

Title Blind Source Separation for Multivariate Spatial Data

Version 0.16-0

Date 2025-03-26

Author Christoph Muehlmann [aut] (<https://orcid.org/0000-0001-7330-8434>),
Mika Sipil<e4> [aut] (<https://orcid.org/0000-0002-5912-840X>),
Claudia Cappello [aut] (<https://orcid.org/0000-0002-7905-5068>),
Sandra De Iaco [aut] (<https://orcid.org/0000-0003-1820-2068>),
Klaus Nordhausen [aut, cre] (<https://orcid.org/0000-0002-3758-8501>),
Sara Taskinen [aut] (<https://orcid.org/0000-0001-9470-7258>),
Joni Virta [aut] (<https://orcid.org/0000-0002-2150-2769>)

Maintainer Klaus Nordhausen <klausnordhausenR@gmail.com>

Description Blind source separation for multivariate spatial data based on simultaneous/joint diagonalization of (robust) local covariance matrices. This package is an implementation of the methods described in Bachoc, Genton, Nordhausen, Ruiz-Gazen and Virta (2020) <doi:10.1093/biomet/asz079>.

License GPL (>= 2)

Imports Rcpp (>= 1.0.2), JADE, sp, stats, SpatialNP, distances, robustbase

Suggests sf, knitr, rmarkdown, markdown, gstat

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

NeedsCompilation yes

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2025-03-27 13:20:05 UTC

Contents

`SpatialBSS-package` *Blind Source Separation for Multivariate Spatial Data*

Description

Blind source separation for multivariate spatial data based on simultaneous/joint diagonalization of local covariance matrices. This package is an implementation of the methods described in Nordhausen, Oja, Filzmoser and Reimann (2015) <doi:10.1007/s11004-014-9559-5>, Bachoc, Genton, Nordhausen, Ruiz-Gazen and Virta (2020) <doi:10.1093/biomet/asz079> and Muehlmann, Bachoc and Nordhausen (2022) <doi:10.1016/j.spasta.2021.100574> as well as some related methods.

Details

Package:	SpatialBSS
Type:	Package
Version:	0.16-0
Date:	2025-03-26
License:	GPL (>= 2)

This package provides functions to solve the Blind Source Separation problem for multivariate spatial data. These methods are designed to work with random fields that are observed on irregular locations. Moreover, the random field is assumed to show weak second order stationarity. The main functions of this package are:

- `sbss`: This function derives a set of local scatter matrices that are based on spatial kernel functions, where the spatial kernel functions can be chosen. Then this set of local covariance matrices as well as the sample covariance matrix are simultaneously/jointly diagonalized. Local covariance matrices as well as local difference matrices are implemented.
- `sbss_asymp`, `sbss_boot`: These functions test for white noise components in the estimated latent field estimated by the `sbss` function based on asymptotic results or bootstrap inference principles.
- `snss_sd`, `snss_jd`, and `snss_sjd`: These functions estimate the latent random field assuming a spatial non-stationary source separation model. This is done by splitting the domain into a number of sub-domains and diagonalizing the corresponding covariance and/or local covariance matrices for each sub-domain.
- `robsbss`: Uses robust estimates of local covariance matrices to solve the SBSS problem.

Joint diagonalization is computed with the `frjd` (fast real joint diagonalization) algorithm from the package `JADE`.

The random field can be either a pair of numeric matrices giving the coordinates and field values or an object of class `SpatialPointsDataFrame` or `sf`.

Author(s)

Christoph Muehlmann, Mika Sipila, Claudia Cappello, Sandra De Iaco, Klaus Nordhausen, Sara Taskinen, Joni Virta

Maintainer: Klaus Nordhausen <klausnordhausenR@gmail.com>

References

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 107, 627-646, doi:10.1093/biomet/asz079.

Muehlmann, C., Bachoc, F. and Nordhausen, K. (2022), *Blind Source Separation for Non-Stationary Random Fields*, Spatial Statistics, 47, 100574, doi:10.1016/j.spasta.2021.100574.

Muehlmann, C., Bachoc, F., Nordhausen, K. and Yi, M. (2024), *Test of the Latent Dimension of a Spatial Blind Source Separation Model*, Statistica Sinica, 34, 837-865, doi:10.5705/ss.202021.0326.

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2024), *Spatial Blind Source Separation in the Presence of a Drift*, Austrian Journal of Statistics, 53, 48-68, doi:10.17713/ajs.v53i2.1668.

Nordhausen, K., Oja, H., Filzmoser, P. and Reimann, C. (2015), *Blind Source Separation for Spatial Compositional Data*, Mathematical Geosciences 47, 753-770, doi:10.1007/s1100401495595.

Sipila, M., Muehlmann, C. Nordhausen, K. and Taskinen, S. (2024). *Robust second order stationary spatial blind source separation using generalized sign matrices*, Spatial Statistics, 59, 100803, doi:10.1016/j.spasta.2023.100803.

coef.sbss *Coef Method for an Object of Class 'sbss'*

Description

Extracts the estimated unmixing matrix of an object of class `'sbss'`.

Usage

```
## S3 method for class 'sbss'
coef(object, ...)
```

Arguments

`object`	object of class `'sbss'`. Usually result of `sbss`.
`...`	further arguments to be passed to or from methods.

Value

Returns the estimated unmixing matrix of an object of class `'sbss'` as a numeric matrix.

See Also

`sbss`

gen_glob_outl *Contamination with Global Outliers*

Description

Generates synthetic global outliers and contaminates a given p-variate random field

Usage

```
gen_glob_outl(x, alpha = 0.05, h = 10, random_sign = FALSE)
```

Arguments

`x`	a numeric matrix of dimension `c(n, p)` where the `p` columns correspond to the entries of the random field and the `n` rows are the observations.
`alpha`	a numerical value between 0 and 1 giving the proportion of observations to contaminate.
`h`	a numerical constant to determine how large the contaminated outliers are, see details.
`random_sign`	logical. If `TRUE`, the sign of each component of the outlier is randomly selected. Default is `FALSE`. See more in details.

Details

gen_glob_outl generates outliers for a given field by selecting randomly round(alpha * n) observations x_i to be the outliers and contaminating them by setting $x_i^{out} = (c^i)' x_i$, where the elements c_j^i of vector c^i are determined by the parameter random_sign. If random_sign = TRUE, c_j^i is either h or $-h$ with $P(c_j^i = h) = P(c_j^i = -h) = 0.5$. If random_sign = FALSE, $c_j^i = h$ for all $j = 1, ...p$, $i = 1, ..., n$. The parameter alpha determines the contamination rate α and the parameter h determines the size of the outliers.

Value

gen_glob_outl returns a data.frame containing the contaminated fields as p first columns. The column $p + 1$ contains a logical indicator whether the observation is outlier or not.

See Also

gen_loc_outl

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- cbind(field_1, field_2, field_3)
  # Generate 10 % global outliers to data, with size h=15.
  field_cont <- gen_glob_outl(field, alpha = 0.1, h = 15)

  # Generate 5 % global outliers to data, with size h = 10 and random sign.
  field_cont2 <- gen_glob_outl(field, alpha = 0.05, h = 10, random_sign = TRUE)
}
```

gen_loc_outl *Contamination with Local Outliers*

Description

Generates synthetic local outliers and contaminates a given p-variate random field by swapping observations based on the first principal component score.

Usage

```
gen_loc_outl(x, coords, alpha = 0.05,
             neighborhood_type = c("radius", "fixed_n"),
             radius = NULL,
             neighborhood_size = NULL,
             swap_order = c("regular", "reverse", "random"))
```

Arguments

x	a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations.
coords	a numeric matrix or data frame with dimension c(n,2) containing the coordinates of the observations.
alpha	a numeric value between 0 and 1 determining the proportion of the contaminated observations.
neighborhood_type	a string determining the type of neighborhood. If 'radius', each neighborhood contains all points within the radius determined by the parameter radius. If 'fixed_n', each neighborhood contains a constant number of closest points, where the constant is determined by the parameter neighborhood_size. Default is 'radius'.
radius	a positive numeric value defining the size of the radius when the neighborhood_type is 'radius'. If NULL the radius defaults as 0.01*n.
neighborhood_size	a positive integer defining the number of points in each neighborhood when the neighborhood_type is 'fixed_n'. If NULL the number of points defaults as ceiling(0.01*n).
swap_order	a string to determine which swap order is used. Either 'regular' (default), 'reverse' or 'random'. See details.

Details

gen_loc_outl generates local outliers by swapping the most extreme and the least extreme observations based on the first principal component score under the condition that at most one outliers lies in each neighborhood. For each location s_i, the neighborhood N_i is defined based on the parameter neighborhood_type. When neighborhood_type is 'radius', the neighborhood N_i contains all

locations s_j for which the Euclidean norm $||s_i - s_j|| < r$, where r is determined by the parameter `radius`. When `neighborhood_type` is `'fixed_n'`, the neighborhood N_i contains $m - 1$ nearest locations of s_i, where m is determined by the parameter `neighborhood_size`. For more details see Ernst & Haesbroeck, (2017).

After calculating the neighborhoods, the local outliers are generated following Ernst & Haesbroeck, (2017) and Harris et al. (2014) using the steps:

1. Sort the observations from highest to lowest by their principle component analysis (PCA) scores of the first component (PC-1).
2. Set k to be $\alpha N/2$ rounded to nearest integer and select the set of local outlier points S^{out} by finding k observations with the highest PC-1 values and k observations with the lowest PC-1 values under the condition that for all $s_i, s_j \in S_{out}$ it holds that $N_i \neq N_j$.
3. Form sets X_{large}, which contains k observations with the largest PC-1 values of outlier points S_{out} and X^{small}, which contains k observations with the smallest PC-1 values of outlier points S^{out}. Generate the local outliers by swapping $X^{small,i}$ with $X^{large,k+1-i}$, $i = 1, ..., k$. The parameter `swap_order` defines how the sets X^{large} and X^{small} are ordered.

If the parameter `swap_order` is `'regular'`, X^{small} and X^{large} are sorted by PC-1 score from smallest to largest. If the parameter `swap_order` is `'reverse'`, X^{small} is sorted from largest to smallest and X^{large} from smallest to largest. If the parameter `swap_order` is `'random'`, X^{small} and X^{large} are in random order.

Value

`gen_loc_outl` returns a `data.frame` containing the contaminated fields as p first columns. The column $p + 1$ contains a logical indicator whether the observation is an outlier or not.

Note

This function is a modified version of code originally provided by M. Ernst and G. Haesbroeck.

References

Ernst, M., & Haesbroeck, G. (2017). *Comparison of local outlier detection techniques in spatial multivariate data*. Data Mining and Knowledge Discovery, 31 , 371-399. doi:10.1007/s10618016-04710

Harris, P., Brunsdon, C., Charlton, M., Juggins, S., & Clarke, A. (2014). *Multivariate spatial outlier detection using robust geographically weighted methods*. Mathematical Geosciences, 46 , 1-31. doi:10.1007/s1100401394910

See Also

`gen_glob_outl`

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
```

```
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)

  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- cbind(field_1, field_2, field_3)

  # Generate 5 % local outliers to data using radius neighborhoods
  # and regular swap_order.
  field_cont <- gen_loc_outl(field, coords, alpha = 0.05,
                             neighborhood_type = "radius",
                             radius = 0.5, swap_order = "regular")

  # Generate 10 % local outliers to data using fixed_n neighborhoods
  # and reverse swap_order.
  field_cont2 <- gen_loc_outl(field, coords, alpha = 0.1,
                              neighborhood_type = "fixed_n",
                              neighborhood_size = 10, swap_order = "reverse")
}
```

`local_covariance_matrix` *Computation of Local Covariance Matrices*

Description

`local_covariance_matrix` computes local covariance matrices for a random field based on a given set of spatial kernel matrices.

Usage

```
local_covariance_matrix(x, kernel_list, lcov = c('lcov', 'ldiff', 'lcov_norm'),
                        center = TRUE)
```

Arguments

x	a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations.
kernel_list	a list with spatial kernel matrices of dimension c(n, n). This list is usually computed with the function spatial_kernel_matrix.
lcov	a string indicating which type of local covariance matrix to use. Either 'lcov' (default) or 'ldiff'.
center	logical. If TRUE the data x is centered prior computing the local covariance matrices. Default is TRUE.

Details

Two versions of local covariance matrices are implemented, the argument lcov determines which version is used:

- 'lcov':

$$LCov(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$

- 'ldiff':

$$LDiff(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - x(s_j))(x(s_i) - x(s_j))',$$

- 'lcov_norm':

$$LCov^*(f) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$

with

$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}).$$

Where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the p random field values at location s_i, $\bar{x}$ is the sample mean vector, and the kernel function $f(d)$ determines the locality. The choice 'lcov_norm' is useful when testing for the actual signal dimension of the latent field, see sbss_asymp and sbss_boot. The function local_covariance_matrix computes local covariance matrices for a given random field and given spatial kernel matrices, the type of computed local covariance matrices is determined by the argument 'lcov'. If the argument center equals FALSE then the centering in the above formula for $LCov(f)$ is not carried out. See also spatial_kernel_matrix for details.

Value

local_covariance_matrix returns a list of equal length as the argument kernel_list. Each list entry is a numeric matrix of dimension c(p, p) corresponding to a local covariance matrix. The list has the attribute 'lcov' which equals the function argument lcov.

References

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2024), *Spatial Blind Source Separation in the Presence of a Drift*, Austrian Journal of Statistics, 53, 48-68, doi:10.17713/ajs.v53i2.1668.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 107, 627-646, doi:10.1093/biomet/asz079.

See Also

spatial_kernel_matrix, sbss

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- as.matrix(cbind(field_1, field_2, field_3))

  # computing two ring kernel matrices and corresponding local covariance matrices
  kernel_params_ring <- c(0, 0.5, 0.5, 2)
  ring_kernel_list <-
    spatial_kernel_matrix(coords, 'ring', kernel_params_ring)
  loc_cov_ring <-
    local_covariance_matrix(x = field, kernel_list = ring_kernel_list)

  # computing two ring kernel matrices and corresponding local difference matrices
  kernel_params_ring <- c(0, 0.5, 0.5, 2)
  ring_kernel_list <-
    spatial_kernel_matrix(coords, 'ring', kernel_params_ring)
  loc_cov_ring <-
    local_covariance_matrix(x = field, kernel_list = ring_kernel_list, lcov = 'ldiff')

  # computing three ball kernel matrices and corresponding local covariance matrices
  kernel_params_ball <- c(0.5, 1, 2)
  ball_kernel_list <-
```

```
  spatial_kernel_matrix(coords, 'ball', kernel_params_ball)
loc_cov_ball <-
  local_covariance_matrix(x = field, kernel_list = ball_kernel_list)

# computing three gauss kernel matrices and corresponding local covariance matrices
kernel_params_gauss <- c(0.5, 1, 2)
gauss_kernel_list <-
  spatial_kernel_matrix(coords, 'gauss', kernel_params_gauss)
loc_cov_gauss <-
  local_covariance_matrix(x = field, kernel_list = gauss_kernel_list)
}
```

`local_gss_covariance_matrix`

Computation of Robust Local Covariance Matrices

Description

`local_gss_covariance_matrix` computes generalized local sign covariance matrices for a random field based on a given set of spatial kernel matrices.

Usage

```
local_gss_covariance_matrix(x, kernel_list, lcov = c('norm', 'winsor', 'qwinsor'),
                            center = TRUE)
```

Arguments

`x`	a numeric matrix of dimension `c(n, p)` where the `p` columns correspond to the entries of the random field and the `n` rows are the observations.
`kernel_list`	a list with spatial kernel matrices of dimension `c(n, n)`. This list is usually computed with the function `spatial_kernel_matrix`.
`lcov`	a string indicating which type of robust local covariance matrix to use. Either `'norm'` (default), `'winsor'` or `'qwinsor'`.
`center`	logical. If `TRUE` the data `x` is robustly centered prior computing the local covariance matrices. Default is `TRUE`. See also `white_data`.

Details

Generalized local sign matrices are determined by radial functions $w(l_i)$, where $l_i = ||x(s_i) - T(x)||$ and $T(x)$ is Hettmansperger Randles location estimator (Hettmansperger & Randles, 2002), and kernel functions $f(d_{i,j})$, where $d_{i,j} = ||s_i - s_j||$. Generalized local sign covariance (gLSCM) matrix is then calculated as

$$gLSCM(f, w) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j}) w(l_i) w(l_j) (x(s_i) - T(x))(x(s_j) - T(x))'$$

with

$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}).$$

Three radial functions $w(l_i)$ (Raymaekers & Rousseeuw, 2019) are implemented, the parameter `lcov` defines which is used:

- `'norm'`:

$$w(l_i) = 1/l_i$$

- `'winsor'`:

$$w(l_i) = Q/l_i$$

- `'qwinsor'`:

$$w(l_i) = Q^2/l_i^2.$$

The cutoff Q is defined as $Q = l_{(h)}$, where $l_{(h)}$ is hth order statistic of $\{l_1, ..., l_n\}$ and $h = (n + p + 1)/2$. If the argument `center` equals `FALSE` then the centering in the above formula for $gLSCM(f, w)$ is not carried out. See also `spatial_kernel_matrix` for details.

Value

`local_gss_covariance_matrix` returns a list with two entries:

`cov_sp_list`	List of equal length as the argument `kernel_list`. Each list entry is a numeric matrix of dimension `c(p, p)` corresponding to a robust local covariance matrix. The list has the attribute `'lcov'` which equals the function argument `lcov`.
`weights`	numeric vector of `length(n)` giving the weights for each observation for the robust local covariance estimation.

References

Hettmansperger, T. P., & Randles, R. H. (2002). *A practical affine equivariant multivariate median*. Biometrika, 89 , 851-860. doi:10.1093/biomet/89.4.851.

Raymaekers, J., & Rousseeuw, P. (2019). *A generalized spatial sign covariance matrix. Journal of Multivariate Analysis*, 171 , 94-111. doi:10.1016/j.jmva.2018.11.010.

Sipila, M., Muehlmann, C. Nordhausen, K. & Taskinen, S. (2024). *Robust second order stationary spatial blind source separation using generalized sign matrices*, Spatial Statistics, 59, 100803, doi:10.1016/j.spasta.2023.100803.

See Also

`spatial_kernel_matrix`, `robsbss`

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- cbind(field_1, field_2, field_3)

  # computing two ring kernel matrices and corresponding
  # robust local covariance matrices using 'norm' radial function:
  kernel_params_ring <- c(0, 0.5, 0.5, 2)
  ring_kernel_list <-
    spatial_kernel_matrix(coords, 'ring', kernel_params_ring)
  loc_cov_ring <-
    local_gss_covariance_matrix(x = field, kernel_list = ring_kernel_list,
                                lcov = 'norm')

  # computing three ball kernel matrices and corresponding
  # robust local covariance matrices using 'winsor' radial function:
  kernel_params_ball <- c(0.5, 1, 2)
  ball_kernel_list <-
    spatial_kernel_matrix(coords, 'ball', kernel_params_ball)
  loc_cov_ball <-
    local_gss_covariance_matrix(x = field, kernel_list = ball_kernel_list,
                                lcov = 'winsor')

  # computing three gauss kernel matrices and corresponding
  # robust local covariance matrices using 'qwinsor' radial function:
  kernel_params_gauss <- c(0.5, 1, 2)
  gauss_kernel_list <-
    spatial_kernel_matrix(coords, 'gauss', kernel_params_gauss)
  loc_cov_gauss <-
    local_gss_covariance_matrix(x = field, kernel_list = gauss_kernel_list,
                                lcov = 'qwinsor')
}
```

plot.sbss *Plot Method for an Object of Class 'sbss'*

Description

plot.sbss is an interface to the standard plot method for the class of the estimated source random field.

Usage

```
## S3 method for class 'sbss'
plot(x, which = 1:ncol(x$s), ...)
```

Arguments

x	object of class 'sbss'. Usually result of sbss.
which	a numeric vector indicating which components of the latent field should be plotted.
...	further arguments to the plot method of class(x$s), which is either spplot or plot.

Details

This method calls the corresponding plot method of class(x$s). Either spplot for class(x$s) is SpatialPointsDataFrame or plot.sf for class(x$s) is sf. If x$s is a matrix then it is internally cast to SpatialPointsDataFrame and spplot is used for plotting. Arguments to the corresponding plot functions can be given through ....

See Also

sbss, spplot, plot.sf

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
```

```
                  nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                  model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- as.matrix(cbind(field_1, field_2, field_3))

  # compute ring kernel matrices
  kernel_parameters <- c(0, 1, 1, 2, 2, 3)
  ring_kernel_list <- spatial_kernel_matrix(coords, 'ring', kernel_parameters)

  # apply sbss SpatialPointsDataFrame object
  field_sp <- sp::SpatialPointsDataFrame(coords = coords, data = data.frame(field))
  res_sp <- sbss(field_sp, kernel_list = ring_kernel_list)

  # plot with SpatialPointsDataFrame object
  plot(res_sp)

  # plot with SpatialPointsDataFrame object
  # and additional arguments for spplot function
  plot(res_sp, colorkey = TRUE, as.table = TRUE, cex = 1)

  # apply sbss with sf object
  if (!requireNamespace('sf', quietly = TRUE)) {
    message('Please install the package sf to run the example code.')
  } else {
    field_sf <- sf::st_as_sf(data.frame(coords = coords, field),
                             coords = c(1,2))
    res_sf <- sbss(x = field_sf, kernel_list = ring_kernel_list)

    # plot with sf object
    plot(res_sf)

    # plot with sf object
    # and additional arguments for plot.sf function
    plot(res_sf, axes = TRUE, key.pos = 4)
  }
}
```

`predict.sbss` *Predict Method for an Object of Class 'sbss'*

Description

`predict.sbss` predicts the estimated source random field on a grid with Inverse Distance Weighting (IDW) and plots these predictions.

Usage

```
## S3 method for class 'sbss'
predict(object, p = 2, n_grid = 50, which = 1:ncol(object$s), ...)
```

Arguments

object	object of class 'sbss'. Usually result of sbss.
p	numeric. The positive power parameter for IDW. Default is 2.
n_grid	numeric. Each dimension of the spatial domain is divided by this integer to derive a grid for IDW predictions. Default is 50.
which	a numeric vector indicating which components of the latent field should be predicted.
...	further arguments to the plot method of class(x$s), which is either spplot or plot.

Details

IDW predictions are made on a grid. The side lengths of the rectangular shaped grid cells are derived by the differences of the rounded maximum and minimum values divided by the n_grid argument for each column of object$coords. Hence, the grid contains a total of n_grid ^ 2 points. The power parameter of the IDW predictions is given by p (default: 2).

The predictions are plotted with the corresponding plot method of class(x$s). Either spplot for class(x$s) is SpatialPointsDataFrame or plot.sf for class(x$s) is sf. If x$s is a matrix then it is internally cast to SpatialPointsDataFrame and spplot is used for plotting. Arguments to the corresponding plot functions can be given through ... as it is done by the method plot.sbss.

Value

The return is dependent on the class of the latent field in the 'sbss' object. If class(object$s) is a matrix then a list with the following entries is returned:

vals_pred_idw	a matrix of dimension c(n,p) (when which is default or less than p columns according to the selected components with the which argument) with the IDW predictions of the estimated source random field.
coords_pred_idw	a matrix of dimension c(n,2) with the grid coordinates for the IDW predictions.

If class(object$s) is SpatialPointsDataFrame or sf then the predicted values and their coordinates are returned as an object of the corresponding class.

The return is invisible.

See Also

sbss, plot.sbss, spplot, plot.sf

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- as.matrix(cbind(field_1, field_2, field_3))

  # apply sbss with three ring kernels
  kernel_borders <- c(0, 1, 1, 2, 2, 4)
  res_sbss <-  sbss(field, coords, 'ring', kernel_borders)

  # predict latent fields on grid with default settings
  predict(res_sbss)

  # predict latent fields on grid with custom plotting settings
  predict(res_sbss, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields on a 60x60 grid
  predict(res_sbss, n_grid = 60, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields with a higher IDW power parameter
  predict(res_sbss, p = 10, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields and save the predictions
  predict_list <- predict(res_sbss, p = 5, colorkey = TRUE, as.table = TRUE, cex = 1)
}
```

`print.sbss` *Print Method for an Object of Class 'sbss'*

Description

Prints the estimated unmixing matrix and the diagonalized local covariance matrices for an object of class `'sbss'`.

Usage

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

Arguments

x	object of class 'sbss'. Usually result of sbss.
...	additional arguments for the method print.listof.

See Also

sbss

---

robsbss *Robust Spatial Blind Source Separation*

Description

robsbss is a robust variant of sbss. It estimates the unmixing matrix assuming a spatial blind source separation model by jointly diagonalizing the Hettmansperger-Randles scatter matrix and one/many generalized local sign covariance matrices. These local generalized sign covariance matrices are determined by spatial kernel functions and radial functions. Three types of such kernel functions and three types of radial functions are supported.

Usage

```
robsbss(x, ...)

## Default S3 method:
robsbss(x, coords, kernel_type = c('ring', 'ball', 'gauss'),
     kernel_parameters, lcov = c('norm', 'winsor', 'qwinsor'),
     ordered = TRUE, kernel_list = NULL, ...)
## S3 method for class 'SpatialPointsDataFrame'
robsbss(x, ...)
## S3 method for class 'sf'
robsbss(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, an object of class SpatialPointsDataFrame or an object of class sf.
coords	a numeric matrix of dimension c(n,2) where each row represents the coordinates of a point in the spatial domain. Only needed if x is a matrix and the argument kernel_list is NULL.

`kernel_type` a string indicating which kernel function to use. Either `'ring'` (default), `'ball'` or `'gauss'`.

`kernel_parameters` a numeric vector that gives the parameters for the kernel function. At least length of one for `'ball'` and `'gauss'` or two for `'ring'` kernel, see details.

`lcov` a string indicating which radial function or type of robust local covariance matrix to use. Either `'norm'` (default), `'winsor'` or `'qwinsor'`. See also `local_gss_covariance_matrix` for details.

`ordered` logical. If `TRUE` the entries of the latent field are ordered by the sum of squared (pseudo-)eigenvalues of the diagonalized local covariance matrix/matrices. Default is `TRUE`.

`kernel_list` a list of spatial kernel matrices with dimension `c(n,n)`, see details. Usually computed by the function `spatial_kernel_matrix`.

`...` further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the local covariance matrices. See details and `frjd`.

Details

`robsbss` is a robust variant of `sbss` which uses Hettmansperger-Randles (HR) location and scatter estimates (Hettmansperger & Randles, 2002) for whitening (see `white_data` for details) and jointly diagonalizes HR scatter matrix and generalized local sign matrices to estimate the unmixing matrix. The generalized local sign matrices are determined by radial functions $w(l_i)$, where $l_i = ||x(s_i) - T(x)||$ and $T(x)$ is HR location estimator, and kernel functions $f(d_{i,j})$, where $d_{i,j} = ||s_i - s_j||$. Generalized local sign covariance (gLSCM) matrix is then calculated as

$$gLSCM(f,w) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j}) w(l_i) w(l_j) (x(s_i) - T(x))(x(s_j) - T(x))'$$

with

$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}).$$

Three radial functions (Raymaekers & Rousseeuw, 2019) $w(l_i)$ are implemented, the parameter `lcov` defines which is used:

- `'norm'`:
$$w(l_i) = 1/l_i$$
- `'winsor'`:
$$w(l_i) = Q/l_i$$
- `'qwinsor'`:
$$w(l_i) = Q^2/l_i^2.$$

The cutoff Q is defined as $Q = l_{(h)}$, where $l_{(h)}$ is hth order statistic of $\{l_1, ..., l_n\}$ and $h = (n+p+1)/2$. In addition, three kernel functions $f(d)$ are implemented, the parameter `kernel_type` defines which is used:

- `'ring'`: parameters are inner radius r_{in} and outer radius r_{out}, with $r_{in} < r_{out}$, and $r_{in}, r_{out} \geq 0$:
$$f(d; r_{in}, r_{out}) = I(r_{in} < d \leq r_{out})$$

- 'ball': parameter is the radius r, with $r \geq 0$:

$$f(d; r) = I(d \leq r)$$

- 'gauss': Gaussian function where 95% of the mass is inside the parameter r, with $r \geq 0$:

$$f(d; r) = exp(-0.5(\Phi^{-1}(0.95)d/r)^2).$$

The argument kernel_type determines the used kernel function as presented above, the argument kernel_parameters gives the corresponding parameters for the kernel function. Specifically, if kernel_type equals 'ball' or 'gauss' then kernel_parameters is a numeric vector where each entry corresponds to one parameter. Hence, length(kernel_parameters) local covariance matrices are used. Whereas, if kernel_type equals 'ring', then kernel_parameters must be a numeric vector of even length where subsequently the inner and outer radii must be given (informally: c(r_in1, r_out1, r_in2, r_out2, ...)). In that case length(kernel_parameters) / 2 local covariance matrices are used.

robsbss calls spatial_kernel_matrix internally to compute a list of c(n,n) kernel matrices based on the parameters given, where each entry of those matrices corresponds to $f(d_{i,j})$. Alternatively, such a list of kernel matrices can be given directly to the function robsbss via the kernel_list argument. This is useful when robsbss is called numerous times with the same coordinates/kernel functions as the computation of the kernel matrices is then done only once prior the actual robsbss calls. For details see also spatial_kernel_matrix.

If more than one generalized local sign covariance matrix is used robsbss jointly diagonalizes these matrices with the function frjd. ... provides arguments for frjd, useful arguments might be:

- eps: tolerance for convergence.
- maxiter: maximum number of iterations.

Value

robsbss returns a list of class 'sbss' with the following entries:

s	object of class(x) containing the estimated source random field.
coords	coordinates of the observations. Is NULL if x was a matrix and the argument kernel_list was not NULL at the robsbss call.
w	estimated unmixing matrix.
weights	numeric vector of length(n) giving the weights for each observation for the robust local covariance estimation.
w_inv	inverse of the estimated unmixing matrix.
pevals	(pseudo-)eigenvalues for each latent field entry.
d	matrix of stacked (jointly) diagonalized local covariance matrices with dimension c(length(kernel_parameters)*p,p) for 'ball' and 'gauss' kernel or c((length(kernel_parameters) / 2)*p,p) for 'ring' kernel.
diags	matrix of dimension c(length(kernel_parameters),p) where the rows contain the diagonal of the diagonalized local autocovariance matrices.
x_mu	robustly estimated columnmeans of x.
cov_inv_sqrt	square root of the inverse sample covariance matrix of x.

References

Hettmansperger, T. P., & Randles, R. H. (2002). *A practical affine equivariant multivariate median*. Biometrika, 89 , 851-860. doi:10.1093/biomet/89.4.851.

Raymaekers, J., & Rousseeuw, P. (2019). *A generalized spatial sign covariance matrix. Journal of Multivariate Analysis*, 171 , 94-111. doi:10.1016/j.jmva.2018.11.010.

Sipila, M., Muehlmann, C. Nordhausen, K. & Taskinen, S. (2024). *Robust second order stationary spatial blind source separation using generalized sign matrices*. Spatial Statistics, 59, 100803. doi:10.1016/j.spasta.2023.100803.

See Also

spatial_kernel_matrix, local_gss_covariance_matrix, sp, sf, frjd

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- cbind(field_1, field_2, field_3)
  # Generate 5 % local outliers to data
  field_cont <- gen_loc_outl(field, coords, radius = 2,
                             swap_order = "regular")[,1:3]
  X <- as.matrix(field_cont)

  # apply sbss with three ring kernels
  kernel_parameters <- c(0, 1, 1, 2, 2, 3)
  robsbss_result <-
    robsbss(X, coords, kernel_type = 'ring', kernel_parameters = kernel_parameters)

  # print object
  print(robsbss_result)

  # plot latent field
  plot(robsbss_result, colorkey = TRUE, as.table = TRUE, cex = 1)
```

```
  # predict latent fields on grid
  predict(robsbss_result, colorkey = TRUE, as.table = TRUE, cex = 1)

  # unmixing matrix
  w_unmix <- coef(robsbss_result)
}
```

---

sbss *Spatial Blind Source Separation*

Description

sbss estimates the unmixing matrix assuming a spatial blind source separation model by simultaneous/jointly diagonalizing the covariance matrix and one/many local covariance matrices. These local covariance matrices are determined by spatial kernel functions. Three types of such kernel functions are supported.

Usage

```
sbss(x, ...)

## Default S3 method:
sbss(x, coords, kernel_type = c('ring', 'ball', 'gauss'),
     kernel_parameters, lcov = c('lcov', 'ldiff', 'lcov_norm'),
     angles = NULL, ordered = TRUE,
     kernel_list = NULL, rob_whitening = FALSE, ...)
## S3 method for class 'SpatialPointsDataFrame'
sbss(x, ...)
## S3 method for class 'sf'
sbss(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, an object of class SpatialPointsDataFrame or an object of class sf.
coords	a numeric matrix of dimension c(n,2) where each row represents the coordinates of a point in the spatial domain. Only needed if x is a matrix and the argument kernel_list is NULL.
kernel_type	a string indicating which kernel function to use. Either 'ring' (default), 'ball' or 'gauss'.
kernel_parameters	a numeric vector that gives the parameters for the kernel function. At least length of one for 'ball' and 'gauss' or two for 'ring' kernel, see details.

`lcov`	a string indicating which type of local covariance matrix to use. Either `'lcov'` (default), `'ldiff'` or `'lcov_norm'`. See `sbss_asymp` for details on the latter option.
`angles`	Optional argument specifying the anisotropic constraint. If `NULL` (default), the function computes isotropic kernels. Otherwise, `angles` must be a list of numeric vectors, where each vector has length 2: (α_1, α_2). The parameter α_1 must be in the range $0 \leq \alpha_1 \leq 2\pi$, and α_2 must satisfy $0 \leq \alpha_2 \leq \pi/2$. For details see `spatial_kernel_matrix`.
`ordered`	logical. If `TRUE` the entries of the latent field are ordered by the sum of squared (pseudo-)eigenvalues of the diagonalized local covariance matrix/matrices. Default is `TRUE`.
`kernel_list`	a list of spatial kernel matrices with dimension `c(n,n)`, see details. Usually computed by the function `spatial_kernel_matrix`.
`rob_whitening`	logical. If `TRUE` whitening is carried out with respect to the first spatial scatter matrix and not the sample covariance matrix, see details. Default is `FALSE`.
`...`	further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the local covariance matrices. See details and `frjd`.

Details

Three versions of local covariance matrices are implemented, the argument `lcov` determines which version is used:

- `'lcov'`:
$$LCov(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$
- `'ldiff'`:
$$LDiff(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - x(s_j))(x(s_i) - x(s_j))',$$
- `'lcov_norm'`:
$$LCov^*(f) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$
with
$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}).$$

Where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the `p` random field values at location s_i, $\bar{x}$ is the sample mean vector, and the kernel function $f(d)$ determines the locality. The choice `'lcov_norm'` is useful when testing for the actual signal dimension of the latent field, see `sbss_asymp` and `sbss_boot`. LDiff matrices are supposed to be more robust when the random field shows a smooth trend. The following kernel functions are implemented and chosen with the argument `kernel_type`:

- `'ring'`: parameters are inner radius r_{in} and outer radius r_{out}, with $r_{in} < r_{out}$, and $r_{in}, r_{out} \geq 0$:
$$f(d; r_{in}, r_{out}) = I(r_{in} < d \leq r_{out})$$

- 'ball': parameter is the radius r, with $r \geq 0$:

$$f(d; r) = I(d \leq r)$$

- 'gauss': Gaussian function where 95% of the mass is inside the parameter r, with $r \geq 0$:

$$f(d; r) = exp(-0.5(\Phi^{-1}(0.95)d/r)^2)$$

The argument kernel_type determines the used kernel function as presented above, the argument kernel_parameters gives the corresponding parameters for the kernel function. Specifically, if kernel_type equals 'ball' or 'gauss' then kernel_parameters is a numeric vector where each entry corresponds to one parameter. Hence, length(kernel_parameters) local covariance matrices are used. Whereas, if kernel_type equals 'ring', then kernel_parameters must be a numeric vector of even length where subsequently the inner and outer radii must be given (informally: c(r_in1, r_out1, r_in2, r_out2, ...)). In that case length(kernel_parameters) / 2 local covariance matrices are used. By default the spatial kernels are of isotropic nature - anisotropic variants can be obtained using the angles argument as detailed in spatial_kernel_matrix.

Internally, sbss calls spatial_kernel_matrix to compute a list of c(n,n) kernel matrices based on the parameters given, where each entry of those matrices corresponds to $f(d_{i,j})$. Alternatively, such a list of kernel matrices can be given directly to the function sbss via the kernel_list argument. This is useful when sbss is called numerous times with the same coordinates/kernel functions as the computation of the kernel matrices is then done only once prior the actual sbss calls. For details see also spatial_kernel_matrix.

rob_whitening determines which scatter is used for the whitening step. If TRUE, whitening is carried out with respect to the scatter matrix defined by the lcov argument, where the kernel function is given by the argument kernel_type and the parameters correspond to the first occuring in the argument kernel_parameters. Therefore, at least two different kernel parameters need to be given. Note that only $LDiff(f)$ matrices are positive definite, hence whitening with 'lcov' is likely to produce an error. If the argument is FALSE, whitening is carried out with respect to the usual sample covariance matrix. sbss internally calls white_data.

If more than one local covariance matrix is used sbss jointly diagonalizes these matrices with the function frjd. ... provides arguments for frjd, useful arguments might be:

- eps: tolerance for convergence.
- maxiter: maximum number of iterations.

Value

sbss returns a list of class 'sbss' with the following entries:

s	object of class(x) containing the estimated source random field.
coords	coordinates of the observations. Is NULL if x was a matrix and the argument kernel_list was not NULL at the sbss call.
w	estimated unmixing matrix.
w_inv	inverse of the estimated unmixing matrix.
pevals	(pseudo-)eigenvalues for each latent field entry.

d	matrix of stacked (jointly) diagonalized local covariance matrices with dimension c(length(kernel_parameters)*p,p) for 'ball' and 'gauss' kernel or c((length(kernel_parameters) / 2)*p,p) for 'ring' kernel.
diags	matrix of dimension c(length(kernel_parameters),p) where the rows contain the diagonal of the diagonalized local autocovariance matrices.
x_mu	columnmeans of x.
cov_inv_sqrt	square root of the inverse sample covariance matrix of x.

References

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2024), *Spatial Blind Source Separation in the Presence of a Drift*, Austrian Journal of Statistics, 53, 48-68, doi:10.17713/ajs.v53i2.1668.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 107, 627-646, doi:10.1093/biomet/asz079.

Nordhausen, K., Oja, H., Filzmoser, P., Reimann, C. (2015), *Blind Source Separation for Spatial Compositional Data*, Mathematical Geosciences 47, 753-770, doi:10.1007/s1100401495595.

Muehlmann, C., Bachoc, F., Nordhausen, K. and Yi, M. (2024), *Test of the Latent Dimension of a Spatial Blind Source Separation Model*, Statistica Sinica, 34, 837-865, doi:10.5705/ss.202021.0326.

See Also

spatial_kernel_matrix, local_covariance_matrix, sp, sf, frjd

Examples

```
# example on simulated data
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")

# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- as.matrix(cbind(field_1, field_2, field_3))
```

```
  # apply sbss with three ring kernels
  kernel_parameters <- c(0, 1, 1, 2, 2, 3)
  sbss_result <-
    sbss(field, coords, kernel_type = 'ring', kernel_parameters = kernel_parameters)

  # print object
  print(sbss_result)

  # plot latent field
  plot(sbss_result, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields on grid
  predict(sbss_result, colorkey = TRUE, as.table = TRUE, cex = 1)

  # unmixing matrix
  w_unmix <- coef(sbss_result)

  # apply the same sbss with a kernel list
  kernel_list <- spatial_kernel_matrix(coords, kernel_type = 'ring', kernel_parameters)
  sbss_result_k <- sbss(field, kernel_list = kernel_list)

  # apply sbss with three ring kernels and local difference matrices
  sbss_result_ldiff <-
    sbss(field, coords, kernel_type = 'ring',
         kernel_parameters = kernel_parameters, lcov = 'ldiff')

#example on real data
data(veneto_weather)

kernel_type <- 'ring'
kernel_parameters <-c(0, 40000, 40000, 65000, 65000, 80000)

angles_1 <- list(c(0, pi / 8), c(pi / 4, pi / 8), c(pi / 2, pi / 8), c(3 * pi / 4, pi / 8))

sbss_anis <- sbss(x = veneto_weather, kernel_type = kernel_type,
                  kernel_parameters = kernel_parameters,
                  angles = angles_1)
}
```

---

sbss_asymp *Asymptotic Test for the White Noise Dimension in a Spatial Blind Source Separation Model*

Description

sbss_asymp uses asymptotic theory for the spatial blind source separation (SBSS) methodology to test if the last $p - q$ entries of the latent random field are white noise assuming that the p-variate observed random field follows a SBSS model.

Usage

```
sbss_asymp(x, ...)

## Default S3 method:
sbss_asymp(x, coords, q, kernel_parameters,
           kernel_list = NULL, ...)
## S3 method for class 'SpatialPointsDataFrame'
sbss_asymp(x, ...)
## S3 method for class 'sf'
sbss_asymp(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, an object of class SpatialPointsDataFrame or an object of class sf.
coords	a numeric matrix of dimension c(n,2) where each row represents the coordinates of a point in the spatial domain. Only needed if x is a matrix and the argument kernel_list is NULL.
q	an integer between 0 and p - 1 specifying the number of hypothetical signal components (null hypothesis) in the latent random field.
kernel_parameters	a numeric vector that gives the parameters for the ring kernel function. At least length of two, see details.
kernel_list	a list of spatial kernel matrices with dimension c(n,n), see details. Usually computed by the function spatial_kernel_matrix.
...	further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the local covariance matrices. See details and frjd.

Details

This function uses the SBSS methodology in conjunction with local covariance matrices based on ring kernel functions to estimate the p-variate latent random field $s = x^{wh}w$, where x^{wh} is the whitened version of the data and w is the estimated unmixing matrix. The considered (adapted) local covariance matrices write as

$$LCov^* = 1/(nF_n^{1/2}) \sum_{i,j} I(r_i < d_{i,j} \leq r_o)(x(s_i) - \bar{x})(x(s_j) - \bar{x})'$$

with

$$F_n = 1/n \sum_{i,j} I(r_i < d_{i,j} \leq r_o).$$

Where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the p random field values at location s_i (which is the i-th row of the argument x and the location corresponds to the i-th row of the argument coords) and $\bar{x}$ is the sample mean vector. The function argument kernel_parameters determines the parameters of the used ring kernel functions or alternatively a list of kernel matrices can be given with the argument kernel_list, see sbss for details.

The null hypothesis specified with the argument `q` states that the last $p - q$ components of the estimated latent field are white noise. The method orders the components of the latent field by the order of the decreasing sums of squares of the corresponding (pseudo-)eigenvalues of the local covariance matrices produced by the joint diagonalization algorithm (or the eigendecomposition if only one local covariance matrix is used). Under the null the lower right $(p - q) * (p - q)$ block matrices of the jointly diagonalized local covariance matrices equal zero matrices. Therefore, the sum of their squared norms m is used as test statistic.

This function conducts the hypothesis test using the asymptotic null distribution of m, a chi-squared distribution with $k(p - q)(p - q + 1)/2$ degrees of freedom (k is the number jointly diagonalized local covariance matrices).

If more than one local covariance matrix is used `sbss_asymp` jointly diagonalizes these matrices with the function `frjd`. `...` provides arguments for `frjd`, useful arguments might be:

- `eps`: tolerance for convergence.
- `maxiter`: maximum number of iterations.

Value

`sbss_asymp` returns a list of class `'sbss_test'` inheriting from the classes `'htest'` and `'sbss'` with the following entries:

`alternative`	a string containing the alternative hypothesis.
`method`	a string which indicates which test methods was used.
`data.name`	a string specifying the name of the used data.
`statistic`	the value of the test statistic.
`parameters`	degrees of freedom for the asymptotic chi-squared distribution of the test statistic under the null hypothesis.
`p.value`	the p-value of the test.
`s`	object of `class(x)` containing the estimated source random field.
`coords`	coordinates of the observations. Is `NULL` if `x` was a matrix and the argument `kernel_list` was not `NULL` at the `sbss_asymp` call.
`w`	estimated unmixing matrix.
`w_inv`	inverse of the estimated unmixing matrix.
`d`	matrix of stacked (jointly) diagonalized local covariance matrices with dimension `c((length(kernel_parameters) / 2)*p,p)`.
`x_mu`	columnmeans of `x`.
`cov_inv_sqrt`	square root of the inverse sample covariance matrix of `x`.

References

Muehlmann, C., Bachoc, F., Nordhausen, K. and Yi, M. (2024), *Test of the Latent Dimension of a Spatial Blind Source Separation Model*, Statistica Sinica, 34, 837-865, doi:10.5705/ss.202021.0326.

See Also

`sbss`, `spatial_kernel_matrix`, `local_covariance_matrix`, `sp`, `sf`, `frjd`

Examples

```
# simulate coordinates
n <- 1000
coords <- runif(n * 2) * 20
dim(coords) <- c(n, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- rnorm(n)
  field_4 <- rnorm(n)

  latent_field <- cbind(as.matrix(cbind(field_1, field_2)), field_3, field_4)
  mixing_matrix <- matrix(rnorm(16), 4, 4)
  observed_field <- latent_field %*% t(mixing_matrix)

  # apply the asymptotic test for a hypothetical latent white noise dimension of q
  # q can lie between 0 and 3 in this case
  # using one ring kernel function and the null hypothesis q = 1
  asymp_res_1 <-
    sbss_asymp(observed_field, coords, q = 1, kernel_parameters = c(0, 1))

  # using two ring kernel functions and the null hypothesis q = 3
  asymp_res_2 <-
    sbss_asymp(observed_field, coords, q = 3, kernel_parameters = c(0, 1, 1, 2))

  # the result is of class sbss_test which is inherited from htest and sbss
  # print object (print method for an object of class htest)
  print(asymp_res_1)
  print(asymp_res_2)

  # plot latent field (plot method for an object of class sbss)
  plot(asymp_res_1, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields on grid (predict method for an object of class sbss)
  predict(asymp_res_1, colorkey = TRUE, as.table = TRUE, cex = 1)

  # unmixing matrix (coef method for an object of class sbss)
  w_unmix <- coef(asymp_res_1)
}
```

sbss_boot *Different Bootstrap Tests for the White Noise Dimension in a Spatial Blind Source Separation Model*

Description

sbss_boot uses bootstrap tests for the spatial blind source separation (SBSS) methodology to test if the last $p - q$ entries of the latent random field are white noise assuming that the p-variate observed random field follows a SBSS model.

Usage

```
sbss_boot(x, ...)

## Default S3 method:
sbss_boot(x, coords, q, kernel_parameters,
          boot_method = c('permute', 'parametric'),
          n_boot = 200, kernel_list = NULL, ...)
## S3 method for class 'SpatialPointsDataFrame'
sbss_boot(x, ...)
## S3 method for class 'sf'
sbss_boot(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, an object of class SpatialPointsDataFrame or an object of class sf.
coords	a numeric matrix of dimension c(n,2) where each row represents the coordinates of a point in the spatial domain. Only needed if x is a matrix and the argument kernel_list is NULL.
q	an integer between 0 and p - 1 specifying the number of hypothetical signal components (null hypothesis) in the latent random field.
kernel_parameters	a numeric vector that gives the parameters for the ring kernel function. At least length of two, see details.
boot_method	a string indicating which bootstrap strategy is used, see details. Either 'permute' (default) or 'parametric'.
n_boot	positive integer specifying the number of bootstrap samples. Default is 200.
kernel_list	a list of spatial kernel matrices with dimension c(n,n), see details. Usually computed by the function spatial_kernel_matrix.
...	further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the local covariance matrices. See details and frjd.

Details

This function uses the SBSS methodology in conjunction with local covariance matrices based on ring kernel functions to estimate the p-variate latent random field $s = x^{wh}w$, where x^{wh} is the whitened version of the data and w is the estimated unmixing matrix. The considered (adapted) local covariance matrices write as

$$LCov^* = 1/(nF_n^{1/2}) \sum_{i,j} I(r_i < d_{i,j} \leq r_o)(x(s_i) - \bar{x})(x(s_j) - \bar{x})'$$

with

$$F_n = 1/n \sum_{i,j} I(r_i < d_{i,j} \leq r_o).$$

Where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the `p` random field values at location s_i (which is the i-th row of the argument `x` and the location corresponds to the i-th row of the argument `coords`) and $\bar{x}$ is the sample mean vector. The function argument `kernel_parameters` determines the parameters of the used ring kernel functions or alternatively a list of kernel matrices can be given with the argument `kernel_list`, see `sbss` for details.

The null hypothesis specified with the argument `q` states that the last $p - q$ components of the estimated latent field are white noise. The method orders the components of the latent field by the order of the decreasing sums of squares of the corresponding (pseudo-)eigenvalues of the local covariance matrices produced by the joint diagonalization algorithm (or the eigendecomposition if only one local covariance matrix is used). Under the null the lower right $(p - q) * (p - q)$ block matrices of the jointly diagonalized local covariance matrices equal zero matrices. Therefore, the sum of their squared norms m is used as test statistic for the bootstrap based inference methods described below.

1. Compute the test statistic m based on the original data x.
2. The estimated latent field s (its dimension is `c(n,p)`) is split into the signal part (first `q` columns) and the white noise part (last `p - q` columns).
3. Replace the noise part by a bootstrap sample drawn based on one of the two strategies described below.
4. Recombine the signal part and resampled noise part by concatenating the columns leading to s^{bs} and back-transform it by $x^{bs} = s^{bs}w^{-1}$.
5. Compute the test statistic m^{bs} based on x^{bs}.
6. Repeat Step 2 - 5 for a total amount of `n_boot` times (default is `200`) and the p-value of the bootstrap test is computed by

$$(sum(m > m^{bs}) + 1)/(n_{boot} + 1).$$

The argument `boot_method` (default is `"permute"`) specifies the used resample strategy. The two following strategies are implemented:

- `boot_method = "permute"`: This strategy is non-parametric. It draws each bootstrap sample from the vector of all $n(p - q)$ observed hypothetical white noise observations.
- `boot_method = "parametric"`: This is parametric. Each bootstrap sample is drawn independently and identically from the standard normal distribution.

If more than one local covariance matrix is used `sbss_boot` jointly diagonalizes these matrices with the function `frjd`. `...` provides arguments for `frjd`, useful arguments might be:

- `eps`: tolerance for convergence.
- `maxiter`: maximum number of iterations.

Value

`sbss_boot` returns a list of class `'sbss_test'` inheriting from the classes `'htest'` and `'sbss'` with the following entries:

`alternative`	a string containing the alternative hypothesis.
`method`	a string which indicates which test methods was used.
`data.name`	a string specifying the name of the used data.
`statistic`	the value of the test statistic.
`parameters`	a integer specifying the number of generated bootstrap samples (the value of the argument `n_boot`).
`p.value`	the p-value of the test.
`s`	object of `class(x)` containing the estimated source random field.
`coords`	coordinates of the observations. Is `NULL` if `x` was a matrix and the argument `kernel_list` was not `NULL` at the `sbss_boot` call.
`w`	estimated unmixing matrix.
`w_inv`	inverse of the estimated unmixing matrix.
`d`	matrix of stacked (jointly) diagonalized local covariance matrices with dimension `c((length(kernel_parameters) / 2)*p,p)`.
`x_mu`	columnmeans of `x`.
`cov_inv_sqrt`	square root of the inverse sample covariance matrix of `x`.

References

Muehlmann, C., Bachoc, F., Nordhausen, K. and Yi, M. (2024), *Test of the Latent Dimension of a Spatial Blind Source Separation Model*, Statistica Sinica, 34, 837-865, doi:10.5705/ss.202021.0326.

See Also

`sbss`, `spatial_kernel_matrix`, `local_covariance_matrix`, `sp`, `sf`, `frjd`

Examples

```
# simulate coordinates
n <- 1000
coords <- runif(n * 2) * 20
dim(coords) <- c(n, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
```

```
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- rnorm(n)
  field_4 <- rnorm(n)

  latent_field <- cbind(as.matrix(cbind(field_1, field_2)), field_3, field_4)
  mixing_matrix <- matrix(rnorm(16), 4, 4)
  observed_field <- latent_field %*% t(mixing_matrix)

  # apply the bootstrap tests for a hypothetical latent white noise dimension of q
  # q can lie between 0 and 3 in this case
  # using one ring kernel function with the permute strategy
  # and the null hypothesis q = 1
  boot_res_1 <-
    sbss_boot(observed_field, coords, q = 1, kernel_parameters = c(0, 1),
              boot_method = 'permute', n_boot = 100)

  # using two one ring kernel function with the parametric strategy
  # and the null hypothesis q = 3
  boot_res_2 <-
    sbss_boot(observed_field, coords, q = 3, kernel_parameters = c(0, 1, 1, 2),
              boot_method = 'parametric', n_boot = 100)

  # the result is of class sbss_test which is inherited from htest and sbss
  # print object (print method for an object of class htest)
  print(boot_res_1)
  print(boot_res_2)

  # plot latent field (plot method for an object of class sbss)
  plot(boot_res_1, colorkey = TRUE, as.table = TRUE, cex = 1)

  # predict latent fields on grid (predict method for an object of class sbss)
  predict(boot_res_1, colorkey = TRUE, as.table = TRUE, cex = 1)

  # unmixing matrix (coef method for an object of class sbss)
  w_unmix <- coef(boot_res_1)
}
```

---

snss_jd *Spatial Non-Stationary Source Separation Joint Diagonalization*

Description

snss_jd estimates the unmixing matrix assuming a spatial non-stationary source separation model implying non-constant covariance by jointly diagonalizing at least two covariance matrices computed for corresponding different sub-domains.

Usage

```
snss_jd(x, ...)

## Default S3 method:
snss_jd(x, coords, n_block, ordered = TRUE, ...)
## S3 method for class 'list'
snss_jd(x, coords, ordered = TRUE, ...)
## S3 method for class 'SpatialPointsDataFrame'
snss_jd(x, ...)
## S3 method for class 'sf'
snss_jd(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, a list of length K defining the subdivision of the domain, an object of class sf or an object of class SpatialPointsDataFrame.
coords	a numeric matrix of dimension c(n,2) when x is a matrix where each row represents the sample location of a point in the spatial domain or a list of length K if x is a list which defines the subdivision of the domain. Not needed otherwise.
n_block	an integer defining the subdivision of the domain. See details.
ordered	logical. If TRUE the entries of the latent field are ordered by the sum of squared pseudo-eigenvalues of the diagonalized sub-domain covariance matrices. Default is TRUE.
...	further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the sub-domain covariance matrices. See details and frjd.

Details

This function assumes that the random field x is formed by

$$x(t) = As(t) + b,$$

where A is the deterministic $p \times p$ mixing matrix, b is the p-dimensional location vector, x is the observable p-variate random field given by the argument x, t are the spatial locations given by the argument coords and s is the latent p-variate random field assumed to consist of uncorrelated entries that have zero mean but non-constant variances. This function aims to recover s by

$$W(x(t) - \bar{x}),$$

where W is the $p \times p$ unmixing matrix and $\bar{x}$ is the sample mean. The function does this by splitting the given spatial domain into `n_block^2` equally sized rectangular sub-domains and jointly diagonalizing the corresponding covariance matrices for all sub-domains.

Alternatively the domain subdivision can be defined by providing lists of length `K` for the arguments `x` and `coords` where the first list entries correspond to the values and coordinates of the first sub-domain and the second entries to the values and coordinates of the second sub-domain, etc..

`snss_jd` jointly diagonalizes the covariance matrices for each sub-domain with the function `frjd`. `...` provides arguments for `frjd`, useful arguments might be:

- `eps`: tolerance for convergence.
- `maxiter`: maximum number of iterations.

Value

Similarly as `sbss` the function `snss_jd` returns a list of class `'snss'` and `'sbss'` with the following entries:

`s`	object of `class(x)` containing the estimated source random field.
`coords`	coordinates of the observations. Only given if `x` is a matrix or list.
`w`	estimated unmixing matrix.
`w_inv`	inverse of the estimated unmixing matrix.
`d`	matrix of stacked (jointly) diagonalized sub-domain covariance matrices with dimension `c(n_block^2*p,p)` or `c(K*p,p)` if `x` and `coords` are lists of length `K`.
`x_mu`	columnmeans of `x`.
`cov_inv_sqrt`	square root of the inverse sample covariance matrix of `x`.

References

Muehlmann, C., Bachoc, F. and Nordhausen, K. (2022), *Blind Source Separation for Non-Stationary Random Fields*, Spatial Statistics, 47, 100574, doi:10.1016/j.spasta.2021.100574.

See Also

`sbss`, `sp`, `sf`

Examples

```
# simulate coordinates
n <- 1000
coords <- runif(n * 2) * 20
dim(coords) <- c(n, 2)

# simulate random field
field_1 <- rnorm(n)
field_2 <- 2 * sin(pi / 20 * coords[, 1]) * rnorm(n)
field_3 <- rnorm(n) * (coords[, 1] < 10) + rnorm(n, 0, 3) * (coords[, 1] >= 10)
```

```
latent_field <- cbind(field_1, field_2, field_3)
mixing_matrix <- matrix(rnorm(9), 3, 3)
observed_field <- latent_field

observed_field_sp <- sp::SpatialPointsDataFrame(coords = coords,
                                                data = data.frame(observed_field))
sp::spplot(observed_field_sp, colorkey = TRUE, as.table = TRUE, cex = 1)

# apply snss_jd with 4 sub-domains
res_4 <- snss_jd(observed_field, coords, n_block = 2)
JADE::MD(W.hat = coef(res_4), A = mixing_matrix)

# apply snss_jd with 9 sub-domains
res_9 <- snss_jd(observed_field, coords, n_block = 3)
JADE::MD(W.hat = coef(res_9), A = mixing_matrix)
cor(res_9$s, latent_field)

# print object
print(res_4)

# plot latent field
plot(res_4, colorkey = TRUE, as.table = TRUE, cex = 1)

# predict latent fields on grid
predict(res_4, colorkey = TRUE, as.table = TRUE, cex = 1)

# unmixing matrix
w_unmix <- coef(res_4)

# apply snss_jd with SpatialPointsDataFrame object
res_4_sp <- snss_jd(observed_field_sp, n_block = 2)

# apply with list arguments
# first axis split by 5
# second axis split by 10
# results in 4 sub-domains
flag_x <- coords[, 1] < 5
flag_y <- coords[, 2] < 10
coords_list <- list(coords[flag_x & flag_y, ],
                    coords[!flag_x & flag_y, ],
                    coords[flag_x & !flag_y, ],
                    coords[!flag_x & !flag_y, ])
field_list <- list(observed_field[flag_x & flag_y, ],
                   observed_field[!flag_x & flag_y, ],
                   observed_field[flag_x & !flag_y, ],
                   observed_field[!flag_x & !flag_y, ])
plot(coords, col = 1)
points(coords_list[[2]], col = 2)
points(coords_list[[3]], col = 3)
points(coords_list[[4]], col = 4)

res_list <- snss_jd(x = field_list,
                    coords = coords_list)
```

```
plot(res_list, colorkey = TRUE, as.table = TRUE, cex = 1)
JADE::MD(W.hat = coef(res_list), A = mixing_matrix)
```

---

snss_sd *Spatial Non-Stationary Source Separation Simultaneous Diagonalization*

Description

snss_sd estimates the unmixing matrix assuming a spatial non-stationary source separation model implying non-constant covariance by simultaneously diagonalizing two covariance matrices computed for two corresponding different sub-domains.

Usage

```
snss_sd(x, ...)

## Default S3 method:
snss_sd(x, coords, direction = c('x', 'y'),
     ordered = TRUE, ...)
## S3 method for class 'list'
snss_sd(x, coords, ordered = TRUE, ...)
## S3 method for class 'SpatialPointsDataFrame'
snss_sd(x, ...)
## S3 method for class 'sf'
snss_sd(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, a list of length two defining the subdivision of the domain, an object of class sf or an object of class SpatialPointsDataFrame.
coords	a numeric matrix of dimension c(n,2) when x is a matrix where each row represents the sample location of a point in the spatial domain or a list of length two if x is a list which defines the subdivision of the domain. Not needed otherwise.
direction	a string indicating on which coordinate axis the domain is halved. Either 'x' (default) or 'y'.
ordered	logical. If TRUE the entries of the latent field are ordered according to the decreasingly ordered eigenvalues. Default is TRUE.
...	further arguments to be passed to or from methods.

Details

This function assumes that the random field x is formed by

$$x(t) = As(t) + b,$$

where A is the deterministic $p \times p$ mixing matrix, b is the p-dimensional location vector, x is the observable p-variate random field given by the argument `x`, t are the spatial locations given by the argument `coords` and s is the latent p-variate random field assumed to consist of uncorrelated entries that have zero mean but non-constant variances. This function aims to recover s by

$$W(x(t) - \bar{x}),$$

where W is the $p \times p$ unmixing matrix and $\bar{x}$ is the sample mean. The function does this by splitting the given spatial domain in half according to the first coordinate (argument `direction` equals `'x'`) or the second coodinate (argument `direction` equals `'y'`) and simultaneously diagonalizing the sample covariance matrices for each of the two sub-domains.

Alternatively the domain subdivison can be defined by providing lists of length two for the arguments `x` and `coords` where the first list entries correspond to the values and coordinates of the first sub-domain and the second entries to the values and coordinates of the second sub-domain.

Value

Similarly as `sbss` the function `snss_sd` returns a list of class `'snss'` and `'sbss'` with the following entries:

`s`	object of `class(x)` containing the estimated source random field.
`coords`	coordinates of the observations. Only given if `x` is a matrix or list.
`w`	estimated unmixing matrix.
`w_inv`	inverse of the estimated unmixing matrix.
`d`	diagonal matrix containing the eigenvalues of the eigendecomposition.
`x_mu`	columnmeans of `x`.
`cov_inv_sqrt`	square root of the inverse sample covariance matrix for the first sub-domain.

References

Muehlmann, C., Bachoc, F. and Nordhausen, K. (2022), *Blind Source Separation for Non-Stationary Random Fields*, Spatial Statistics, 47, 100574, doi:10.1016/j.spasta.2021.100574.

See Also

`sbss`, `sp`, `sf`

Examples

```
# simulate coordinates
n <- 1000
coords <- runif(n * 2) * 20
dim(coords) <- c(n, 2)
```

```
# simulate random field
field_1 <- rnorm(n)
field_2 <- 2 * sin(pi / 20 * coords[, 1]) * rnorm(n)
field_3 <- rnorm(n) * (coords[, 1] < 10) + rnorm(n, 0, 3) * (coords[, 1] >= 10)

latent_field <- cbind(field_1, field_2, field_3)
mixing_matrix <- matrix(rnorm(9), 3, 3)
observed_field <- latent_field %*% t(mixing_matrix)

observed_field_sp <- sp::SpatialPointsDataFrame(coords = coords,
                                                data = data.frame(observed_field))
sp::spplot(observed_field_sp, colorkey = TRUE, as.table = TRUE, cex = 1)

# apply snss_sd with split in x
res_x <- snss_sd(observed_field, coords, direction = 'x')
JADE::MD(W.hat = coef(res_x), A = mixing_matrix)

# apply snss_sd with split in y
# should be much worse as field shows only variation in x
res_y <- snss_sd(observed_field, coords, direction = 'y')
JADE::MD(W.hat = coef(res_y), A = mixing_matrix)

# print object
print(res_x)

# plot latent field
plot(res_x, colorkey = TRUE, as.table = TRUE, cex = 1)

# predict latent fields on grid
predict(res_x, colorkey = TRUE, as.table = TRUE, cex = 1)

# unmixing matrix
w_unmix <- coef(res_x)

# apply snss_sd with SpatialPointsDataFrame object
res_x_sp <- snss_sd(observed_field_sp, direction = 'x')

# apply with list arguments
# first axis split by 5
flag_coords <- coords[, 1] < 5
coords_list <- list(coords[flag_coords, ],
                    coords[!flag_coords, ])
field_list <- list(observed_field[flag_coords, ],
                   observed_field[!flag_coords, ])
plot(coords, col = flag_coords + 1)

res_list <- snss_sd(x = field_list,
                    coords = coords_list)
plot(res_list, colorkey = TRUE, as.table = TRUE, cex = 1)
JADE::MD(W.hat = coef(res_list), A = mixing_matrix)
```

snss_sjd *Spatial Non-Stationary Source Separation Spatial Joint Diagonalization*

Description

snss_sjd estimates the unmixing matrix assuming a spatial non-stationary source separation model implying non-constant (spatial) covariance by jointly diagonalizing several covariance and/or spatial covariance matrices computed for a subdivision of the spatial domain into at least two subdomains.

Usage

```
snss_sjd(x, ...)

## Default S3 method:
snss_sjd(x, coords, n_block, kernel_type = c('ring', 'ball', 'gauss'),
     kernel_parameters, with_cov = TRUE, lcov = c('lcov', 'ldiff', 'lcov_norm'),
     ordered = TRUE, ...)
## S3 method for class 'list'
snss_sjd(x, coords, kernel_type = c('ring', 'ball', 'gauss'),
     kernel_parameters, with_cov = TRUE, lcov = c('lcov', 'ldiff', 'lcov_norm'),
     ordered = TRUE, ...)
## S3 method for class 'SpatialPointsDataFrame'
snss_sjd(x, ...)
## S3 method for class 'sf'
snss_sjd(x, ...)
```

Arguments

x	either a numeric matrix of dimension c(n, p) where the p columns correspond to the entries of the random field and the n rows are the observations, a list of length K defining the subdivision of the domain, an object of class sf or an object of class SpatialPointsDataFrame.
coords	a numeric matrix of dimension c(n,2) when x is a matrix where each row represents the sample location of a point in the spatial domain or a list of length K if x is a list which defines the subdivision of the domain. Not needed otherwise.
n_block	either be an integer defining the subdivision of the domain, 'x' or 'y'. See details.
kernel_type	a string indicating which kernel function to use. Either 'ring' (default), 'ball' or 'gauss'.
kernel_parameters	a numeric vector that gives the parameters for the kernel function. At least length of one for 'ball' and 'gauss' or two for 'ring' kernel, see details.

with_cov	logical. If TRUE not only spatial covariance matrices but also the sample covariances matrices for each sub-domain are considered in the joint diagonalization procedure. Default is TRUE.
lcov	a string indicating which type of local covariance matrix to use. Either 'lcov' (default), 'ldiff' or 'lcov_norm'. See sbss_asymp for details on the latter option.
ordered	logical. If TRUE the entries of the latent field are ordered by the sum of squared pseudo-eigenvalues of the diagonalized sub-domain (local) covariance matrices. Default is TRUE.
...	further arguments for the fast real joint diagonalization algorithm that jointly diagonalizes the sub-domain covariance matrices. See details and frjd.

Details

This function assumes that the random field x is formed by

$$x(t) = As(t) + b,$$

where A is the deterministic $p \times p$ mixing matrix, b is the p-dimensional location vector, x is the observable p-variate random field given by the argument x, t are the spatial locations given by the argument coords and s is the latent p-variate random field assumed to consist of uncorrelated entries that have zero mean but non-constant (spatial) second order dependence. This function aims to recover s by

$$W(x(t) - \bar{x}),$$

where W is the $p \times p$ unmixing matrix and $\bar{x}$ is the sample mean. The function does this by splitting the given spatial domain into n_block^2 equally sized rectangular sub-domains and jointly diagonalizing the corresponding spatial covariance matrices for all sub-domains. If the argument with_cov equals TRUE (default) then additionally also the sample covariance matrices for each sub-domain are included in the joint diagonalization procedure.

The arguments kernel_type, kernel_parameters and lcov determine which spatial kernel functions and which type of local covariance matrices are used for each sub-domain. The usage is equal to the function sbss.

Alternatively the domain subdivision can be defined by providing lists of length K for the arguments x and coords where the first list entries correspond to the values and coordinates of the first sub-domain and the second entries to the values and coordinates of the second sub-domain, etc.. The argument n_block might be 'x' or 'y' indicating a split across the x or y coordinates similar as done by the function snss_sd.

snss_sjd jointly diagonalizes the covariance matrices for each sub-domain with the function frjd. ... provides arguments for frjd, useful arguments might be:

- eps: tolerance for convergence.
- maxiter: maximum number of iterations.

Value

Similarly as sbss the function snss_jd returns a list of class 'snss' and 'sbss' with the following entries:

s	object of class(x) containing the estimated source random field.
coords	coordinates of the observations. Only given if x is a matrix or list.
w	estimated unmixing matrix.
w_inv	inverse of the estimated unmixing matrix.
d	matrix of stacked (jointly) diagonalized sub-domain covariance and/or local covariance matrices.
x_mu	columnmeans of x.
cov_inv_sqrt	square root of the inverse sample covariance matrix of x.

References

Muehlmann, C., Bachoc, F. and Nordhausen, K. (2022), *Blind Source Separation for Non-Stationary Random Fields*, Spatial Statistics, 47, 100574, doi:10.1016/j.spasta.2021.100574.

See Also

sbss, sp, sf

Examples

```
# simulate coordinates
n <- 1000
coords <- runif(n * 2) * 20
dim(coords) <- c(n, 2)

# simulate random field
field_1 <- rnorm(n)
field_2 <- 2 * sin(pi / 20 * coords[, 1]) * rnorm(n)
field_3 <- rnorm(n) * (coords[, 1] < 10) + rnorm(n, 0, 3) * (coords[, 1] >= 10)

latent_field <- cbind(field_1, field_2, field_3)
mixing_matrix <- matrix(rnorm(9), 3, 3)
observed_field <- latent_field

observed_field_sp <- sp::SpatialPointsDataFrame(coords = coords,
                                                data = data.frame(observed_field))
sp::spplot(observed_field_sp, colorkey = TRUE, as.table = TRUE, cex = 1)

# apply snss_sjd with 4 sub-domains
# one ring kernel per sub-domain
# without covariances
res_4_ball <- snss_sjd(observed_field, coords, n_block = 2,
                  kernel_type = 'ball', kernel_parameters = c(0, 2),
                  with_cov = TRUE)
JADE::MD(W.hat = coef(res_4_ball), A = mixing_matrix)

# apply snss_sjd with split across y
# one ring kernel per sub-domain
# without covariances
# should not work as field does not show spatial dependence
```

```
res_4_ring <- snss_sjd(observed_field, coords, n_block = 'y',
                       kernel_type = 'ring', kernel_parameters = c(0, 2),
                       with_cov = FALSE)
JADE::MD(W.hat = coef(res_4_ring), A = mixing_matrix)

# print object
print(res_4_ball)

# plot latent field
plot(res_4_ball, colorkey = TRUE, as.table = TRUE, cex = 1)

# predict latent fields on grid
predict(res_4_ball, colorkey = TRUE, as.table = TRUE, cex = 1)

# unmixing matrix
w_unmix <- coef(res_4_ball)

# apply snss_jd with SpatialPointsDataFrame object
res_4_ball_sp <- snss_sjd(observed_field_sp, n_block = 2,
                          kernel_type = 'ball', kernel_parameters = c(0, 2),
                          with_cov = TRUE)

# apply with list arguments
# first axis split by 5
# second axis split by 10
# results in 4 sub-domains
flag_x <- coords[, 1] < 5
flag_y <- coords[, 2] < 10
coords_list <- list(coords[flag_x & flag_y, ],
                    coords[!flag_x & flag_y, ],
                    coords[flag_x & !flag_y, ],
                    coords[!flag_x & !flag_y, ])
field_list <- list(observed_field[flag_x & flag_y, ],
                   observed_field[!flag_x & flag_y, ],
                   observed_field[flag_x & !flag_y, ],
                   observed_field[!flag_x & !flag_y, ])
plot(coords, col = 1)
points(coords_list[[2]], col = 2)
points(coords_list[[3]], col = 3)
points(coords_list[[4]], col = 4)

res_list <- snss_sjd(x = field_list,
                     coords = coords_list,
                     kernel_type = 'ring', kernel_parameters = c(0, 2))
plot(res_list, colorkey = TRUE, as.table = TRUE, cex = 1)
JADE::MD(W.hat = coef(res_list), A = mixing_matrix)
```

---

spatial_kernel_matrix *Computation of Spatial Kernel Matrices*

Description

spatial_kernel_matrix computes spatial kernel matrices for a given kernel function with its parameters and a set of coordinates.

Usage

```
spatial_kernel_matrix(coords, kernel_type = c('ring', 'ball', 'gauss'),
                      kernel_parameters, angles = NULL)
```

Arguments

coords	a numeric matrix of dimension c(n,2) where each row represents the coordinates of a point in the spatial domain.
kernel_type	a character string indicating which kernel function to use. Either 'ring' (default), 'ball' or 'gauss'.
kernel_parameters	a numeric vector that gives the parameters for the kernel function. At least length of one for 'ball' and 'gauss' or two for 'ring' kernel, see details.
angles	Optional argument specifying the anisotropic constraint. If NULL (default), the function computes isotropic kernels. Otherwise, angles must be a list of numeric vectors, where each vector has length 2: (α_1, α_2). The parameter α_1 must be in the range $0 \leq \alpha_1 \leq 2\pi$, and α_2 must satisfy $0 \leq \alpha_2 \leq \pi/2$, see details.

Details

Two versions of local covariance matrices can be defined:

- 'lcov':
$$LCov(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$
- 'ldiff':
$$LDiff(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - x(s_j))(x(s_i) - x(s_j))',$$
- 'lcov_norm':
$$LCov^*(f) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$

  with
$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}).$$

Where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the p random field values at location s_i, $\bar{x}$ is the sample mean vector, and the kernel function $f(d)$ determines the locality. The function spatial_kernel_matrix computes a list of c(n,n) matrices where each entry of these matrices correspond to the spatial kernel function evaluated at the distance between two points, mathematically the entry ij of each kernel matrix is $f(d_{i,j})$. The following kernel functions are implemented and chosen with the argument kernel_type:

- `'ring'`: parameters are inner radius r_i and outer radius r_o, with $r_i < r_o$, and $r_i, r_o \geq 0$:

$$f(d; r_i, r_o) = I(r_i < d \leq r_o)$$

- `'ball'`: parameter is the radius r, with $r \geq 0$:

$$f(d; r) = I(d \leq r)$$

- `'gauss'`: Gaussian function where 95% of the mass is inside the parameter r, with $r \geq 0$:

$$f(d; r) = exp(-0.5(\Phi^{-1}(0.95)d/r)^2)$$

The argument `kernel_type` determines the used kernel function as presented above, the argument `kernel_parameters` gives the corresponding parameters for the kernel function. Specifically, if `kernel_type` equals `'ball'` or `'gauss'` then `kernel_parameters` is a numeric vector where each entry corresponds to one parameter. Hence, `length(kernel_parameters)` spatial kernel matrices of type `kernel_type` are computed. Whereas, if `kernel_type` equals `'ring'`, then `kernel_parameters` must be a numeric vector of even length where subsequently the inner and outer radii must be given (informally: `c(r_i1, r_o1, r_i2, r_o2, ...)`). In that case `length(kernel_parameters) / 2` spatial kernel matrices of type `'ring'` are computed.

If the optional argument `angles` is provided, the computed spatial kernel matrices incorporate anisotropic constraints. The `angles` argument must be a list of numeric vectors, where each vector consists of two values: α_1 (the main direction) and α_2 (the angular tolerance). The directional constraint ensures that only spatial relationships within the specified angle range are considered:

- $0 \leq \alpha_1 \leq 2\pi$ defines the main direction in radians.
- $0 \leq \alpha_2 \leq \pi/2$ specifies the angular tolerance.

When `angles` is not `NULL`, the function applies the kernel function only to points satisfying $d_{ij} \angle e_{\alpha_1} \in [0, \alpha_2]$. If `angles` is `NULL`, the kernels are computed isotropically.

The output of this function can be used with the function `sbss` to avoid unnecessary computation of kernel matrices when `sbss` is called multiple times with the same coordinate/kernel function setting. Additionally, the output can be used with the function `local_covariance_matrix` to actually compute local covariance matrices as defined above based on a given set of spatial kernel matrices.

Value

`spatial_kernel_matrix` returns a list with length of `length(kernel_parameters)` (for `'ball'` and `'gauss'` kernel functions) or `length(kernel_parameters) / 2` (for `'ring'` kernel function) containing numeric matrices of dimension `c(n,n)` corresponding to the spatial kernel matrices.

References

Muehlmann, C., Cappello, C., De Iaco, S. and Nordhausen, K. (2025), *Anisotropic Local Covariance Matrices for Spatial Blind Source Separation*. Manuscript.

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2024), *Spatial Blind Source Separation in the Presence of a Drift*, Austrian Journal of Statistics, 53, 48-68, doi:10.17713/ajs.v53i2.1668.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 107, 627-646, doi:10.1093/biomet/asz079.

See Also

sbss, local_covariance_matrix

Examples

```
# simulate a set of coordinates
coords <- rnorm(100 * 2)
dim(coords) <- c(100, 2)

# computing two ring kernel matrices
kernel_params_ring <- c(0, 0.5, 0.5, 2)
ring_kernel_list <-
  spatial_kernel_matrix(coords, 'ring', kernel_params_ring)

# computing three ball kernel matrices
kernel_params_ball <- c(0.5, 1, 2)
ball_kernel_list <-
  spatial_kernel_matrix(coords, 'ball', kernel_params_ball)

# computing three gauss kernel matrices
kernel_params_gauss <- c(0.5, 1, 2)
gauss_kernel_list <-
  spatial_kernel_matrix(coords, 'gauss', kernel_params_gauss)

# anisotropic examlple
# computing two ring kernel matrices each with two angles
kernel_params_ring <- c(0, 0.5, 0.5, 2)
angles_params <- list(c(pi / 4, pi / 4), c(3 * pi / 4, pi / 4))
ring_aniso_kernel_list <-
  spatial_kernel_matrix(coords, 'ring', kernel_params_ring, angles_params)
```

---

veneto_weather *Weekly Aggregated Climate and Meteorological Data in Veneto, Italy*

Description

The dataset contains weekly averages of various meteorological variables for week 28 of 2021, collected from 75 monitoring stations across the Veneto region in Italy.

Usage

```
data(veneto_weather)
```

Format

An sf (simple features) object with 75 observations and 7 variables:

ET0 Evapotranspiration levels (mm) (aggregated weekly).

tmax Maximum temperature (°C) (weekly max).

tmin Minimum temperature (°C) (weekly min).

hmax Maximum humidity (%) (weekly max).

hmin Minimum humidity (%) (weekly min).

log_prec Log-transformed precipitation values (aggregated weekly).

geometry Simple feature column containing point locations of monitoring stations (sfc_POINT). The coordinates are in meters (Gauss Boaga - EPSG: 3003)

Details

The evapotranspiration levels were estimated by ARPA Veneto according to the Hargreaves model.

Source

The raw data can be downloaded from the Environmental Protection Agency of Veneto Region (ARPA Veneto) website.

Examples

```
data(veneto_weather)
plot(veneto_weather["tmax"])
```

---

white_data *Different Approaches of Data Whitening*

Description

`white_data` whites the data with respect to the sample covariance matrix, or different spatial scatter matrices.

Usage

```
white_data(x, whitening = c("standard", "rob", "hr"),
           lcov = c('lcov', 'ldiff', 'lcov_norm'),
           kernel_mat = numeric(0))
```

Arguments

`x`	a numeric matrix of dimension `c(n, p)` where the `p` columns correspond to the entries of the random field and the `n` rows are the observations.
`whitening`	a string indicating the whitening method. If `'standard'` then the whitening is carried out with respect to sample covariance matrix, if `'rob'` then the first spatial scatter matrix is used instead of sample the covariance matrix and if `'hr'` then the Hettmansperger-Randles location and scatter estimates are used for whitening. See details for more. Default is `'standard'`.
`lcov`	a string indicating which type of local covariance matrix is used for whitening, when the whitening method `'rob'` is used. Either `'lcov'` (default) or `'ldiff'`.
`kernel_mat`	a spatial kernel matrix with dimension `c(n,n)`, see details. Usually computed by the function `spatial_kernel_matrix`.

Details

The inverse square root of a positive definite matrix $M(x)$ with eigenvalue decomposition UDU' is defined as $M(x)^{-1/2} = UD^{-1/2}U'$. `white_data` whitens the data by $M(x)^{-1/2}(x - T(x))$ where $T(x)$ is a location functional of x and the matrix $M(x)$ is a scatter functional. If the argument `whitening` is `'standard'`, $M(x)$ is the sample covariance matrix and $T(x)$ is a vector of column means of x. If the argument `whitening` is `'hr'`, the Hettmansperger-Randles location and scatter estimates (Hettmansperger & Randles, 2002) are used as location functional $T(x)$ and scatter functional $M(x)$. The Hettmansperger-Randles location and scatter estimates are robust variants of sample mean and covariance matrices, that are used for whitening in `robsbss`. If the argument `whitening` is `'rob'`, the argument `lcov` determines the scatter functional $M(x)$ to be one of the following local scatter matrices:

- `'lcov'`:
$$LCov(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$
- `'ldiff'`:
$$LDiff(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - x(s_j))(x(s_i) - x(s_j))',$$
- `'lcov_norm'`:
$$LCov^*(f) = 1/(nF_{f,n}^{1/2}) \sum_{i,j} f(d_{i,j})(x(s_i) - \bar{x})(x(s_j) - \bar{x})',$$

  with
$$F_{f,n} = 1/n \sum_{i,j} f^2(d_{i,j}),$$

where $d_{i,j} \geq 0$ correspond to the pairwise distances between coordinates, $x(s_i)$ are the `p` random field values at location s_i, $\bar{x}$ is the sample mean vector, and the kernel function $f(d)$ determines the locality. The choice `'lcov_norm'` is useful when testing for the actual signal dimension of the latent field, see `sbss_asymp` and `sbss_boot`. See also `sbss` for details.

Note that $LCov(f)$ are usually not positive definite, therefore in that case the matrix cannot be inverted and an error is produced. Whitening with $LCov(f)$ matrices might be favorable in the presence of spatially uncorrelated noise, and whitening with $LDiff(f)$ might be favorable when a non-constant smooth drift is present in the data.

The argument `kernel_mat` is a matrix of dimension `c(n,n)` where each entry corresponds to the spatial kernel function evaluated at the distance between two sample locations, mathematically the entry ij of each kernel matrix is $f(d_{i,j})$. This matrix is usually computed with the function `spatial_kernel_matrix`.

Value

`white_data` returns a list with the following entries:

`mu`	a numeric vector of length `ncol(x)` containing the column means of the data matrix `x`.
`x_0`	a numeric matrix of dimension `c(n, p)` containing the columns centered data of `x`.

x_w	a numeric matrix of dimension c(n, p) containing the whitened data of x.
s	a numeric matrix of dimension c(p, p) which is the scatter matrix M.
s_inv_sqrt	a numeric matrix of dimension c(p, p) which equals the inverse square root of the scatter matrix M used for whitening.
s_sqrt	a numeric matrix of dimension c(p, p) which equals the square root of the scatter matrix M.

References

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2021), *Spatial Blind Source Separation in the Presence of a Drift*, Submitted for publication. Preprint available at https://arxiv.org/abs/2108.13813.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 107, 627-646, doi:10.1093/biomet/asz079.

Hettmansperger, T. P., & Randles, R. H. (2002). *A practical affine equivariant multivariate median*. Biometrika, 89 , 851-860. doi:10.1093/biomet/89.4.851.

See Also

sbss, spatial_kernel_matrix

Examples

```
# simulate coordinates
coords <- runif(1000 * 2) * 20
dim(coords) <- c(1000, 2)
coords_df <- as.data.frame(coords)
names(coords_df) <- c("x", "y")
# simulate random field
if (!requireNamespace('gstat', quietly = TRUE)) {
  message('Please install the package gstat to run the example code.')
} else {
  library(gstat)
  model_1 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Exp'), nmax = 20)
  model_2 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, kappa = 2, model = 'Mat'),
                   nmax = 20)
  model_3 <- gstat(formula = z ~ 1, locations = ~ x + y, dummy = TRUE, beta = 0,
                   model = vgm(psill = 0.025, range = 1, model = 'Gau'), nmax = 20)
  field_1 <- predict(model_1, newdata = coords_df, nsim = 1)$sim1
  field_2 <- predict(model_2, newdata = coords_df, nsim = 1)$sim1
  field_3 <- predict(model_3, newdata = coords_df, nsim = 1)$sim1
  field <- cbind(field_1, field_2, field_3)
  X <- as.matrix(field)

  # white the data with the usual sample covariance
  x_w_1 <- white_data(X)

  # white the data with a ldiff matrix and ring kernel
```

```
  kernel_params_ring <- c(0, 1)
  ring_kernel_list <-
    spatial_kernel_matrix(coords, 'ring', kernel_params_ring)
  x_w_2 <- white_data(field, whitening = 'rob',
    lcov = 'ldiff', kernel_mat = ring_kernel_list[[1]])

  # Generate 5 % of global outliers to data
  field_cont <- gen_glob_outl(field)[,1:3]
  X <- as.matrix(field_cont)
  # white the data using Hettmansperger-Randles location and scatter estimates
  x_w_3 <- white_data(X, whitening = 'hr')
}
```

Index