Package 'SpatialBSS'

May 7, 2021

Type Package

Title Blind Source Separation for Multivariate Spatial Data

Version 0.10-0

Date 2021-05-04

Maintainer Christoph Muehlmann <christoph.muehlmann@tuwien.ac.at>

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 Bachoc, Genton, Nordhausen, Ruiz-Gazen and Virta (2020) <doi:10.1093/biomet/asz079>.

License GPL (>= 2)

Imports Rcpp (>= 1.0.2), JADE, sp

Suggests sf, RandomFields, knitr, rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Christoph Muehlmann [aut, cre] (<https://orcid.org/0000-0001-7330-8434>), Klaus Nordhausen [aut] (<https://orcid.org/0000-0002-3758-8501>), Joni Virta [aut] (<https://orcid.org/0000-0002-2150-2769>)

Repository CRAN

Date/Publication 2021-05-07 09:00:06 UTC

R topics documented:

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, Reimann (2015) <https://doi.org/10.1007/s11004-014-9559-5> and Bachoc, Genton, Nordhausen, Ruiz-Gazen and Virta (2020) <doi:10.1093/biomet/asz079>.

Details

Package:	SpatialBSS
Type:	Package
Version:	0.10-0
Date:	2021-05-04
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 function of this package is:

- `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.

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, Klaus Nordhausen, Joni Virta

Maintainer: Christoph Muehlmann <christoph.muehlmann@tuwien.ac.at>

References

Muehlmann, C., Filzmoser, P. and Nordhausen, K. (2021), *Spatial Blind Source Separation in the Presence of a Drift*, Submitted for publication.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 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.

`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`

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

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 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. (2021), *Spatial Blind Source Separation in the Presence of a Drift*, Submitted for publication.

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

See Also

spatial_kernel_matrix, sbss

Examples

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

# simulate random field
if (!requireNamespace('RandomFields', quietly = TRUE)) {
  stop('Please install the package RandomFields to run the example code.')
} else {
  RandomFields::RFoptions(spConform = FALSE)
  field_1 <- RandomFields::RFsimulate(model = RandomFields::RMexp(),
                                      x = coords)
  field_2 <- RandomFields::RFsimulate(model = RandomFields::RMspheric(),
                                      x = coords)
  field_3 <- RandomFields::RFsimulate(model = RandomFields::RMwhittle(nu = 2),
                                      x = coords)
  field <- 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)
```

`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)

# simulate random field
if (!requireNamespace('RandomFields', quietly = TRUE)) {
  stop('Please install the package RandomFields to run the example code.')
} else {
  RandomFields::RFoptions(spConform = FALSE)
  field_1 <- RandomFields::RFsimulate(model = RandomFields::RMexp(),
                                      x = coords)
  field_2 <- RandomFields::RFsimulate(model = RandomFields::RMspheric(),
                                      x = coords)
  field_3 <- RandomFields::RFsimulate(model = RandomFields::RMwhittle(nu = 2),
                                      x = coords)
  field <- 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)) {
  stop('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)

# simulate random field
if (!requireNamespace('RandomFields', quietly = TRUE)) {
  stop('Please install the package RandomFields to run the example code.')
} else {
  RandomFields::RFoptions(spConform = FALSE)
  field_1 <- RandomFields::RFsimulate(model = RandomFields::RMexp(),
                                      x = coords)
  field_2 <- RandomFields::RFsimulate(model = RandomFields::RMspheric(),
                                      x = coords)
```

```
  field_3 <- RandomFields::RFsimulate(model = RandomFields::RMwhittle(nu = 2),
                                      x = coords)
  field <- 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`

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'), 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) or 'ldiff'.
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

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

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. 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_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) 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_i1,r_o1,r_i2,r_o2,...)). In that case length(kernel_parameters) / 2 local covariance matrices are used.

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.
`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.
`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. (2021), *Spatial Blind Source Separation in the Presence of a Drift*, Submitted for publication.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 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.

See Also

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

Examples

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

# simulate random field
if (!requireNamespace('RandomFields', quietly = TRUE)) {
  stop('Please install the package RandomFields to run the example code.')
} else {
  RandomFields::RFoptions(spConform = FALSE)
```

```
  field_1 <- RandomFields::RFsimulate(model = RandomFields::RMexp(),
                                      x = coords)
  field_2 <- RandomFields::RFsimulate(model = RandomFields::RMspheric(),
                                      x = coords)
  field_3 <- RandomFields::RFsimulate(model = RandomFields::RMwhittle(nu = 2),
                                      x = coords)
  field <- 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')
```

`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)
```

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.

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

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.

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., Filzmoser, P. and Nordhausen, K. (2021), *Spatial Blind Source Separation in the Presence of a Drift*, Submitted for publication.

Bachoc, F., Genton, M. G, Nordhausen, K., Ruiz-Gazen, A. and Virta, J. (2020), *Spatial Blind Source Separation*, Biometrika, 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)
```

---

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, rob_whitening = FALSE, lcov = c('lcov', 'ldiff'),
           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.
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.
lcov	a string indicating which type of local covariance matrix is used for whitening. 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 with eigenvalue decomposition UDU' is defined as $M^{-1/2} = UD^{-1/2}U'$. white_data whitens the data by $M^{-1/2}(x - mu)$ where mu are the column means of x and the matrix M is as follows. If the argument rob_whitening is FALSE then M is the sample covariance matrix. If the argument rob_whitening is TRUE, then the argument lcov determines the matrix M 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':
$$LDiff(f) = 1/n \sum_{i,j} f(d_{i,j})(x(s_i) - x(s_j))(x(s_i) - x(s_j))'.$$

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. 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 favourable in the presence of spatially uncorrelated noise, and whitening with $LDiff(f)$ might be favourable 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 centerd data of x.
x_w	a numeric matrix of dimension c(n,p) containing the whitened data of x.
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.

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

See Also

sbss, spatial_kernel_matrix

Examples

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

# simulate random field
if (!requireNamespace('RandomFields', quietly = TRUE)) {
  stop('Please install the package RandomFields to run the example code.')
} else {
  RandomFields::RFoptions(spConform = FALSE)
  field_1 <- RandomFields::RFsimulate(model = RandomFields::RMexp(),
                                      x = coords)
  field_2 <- RandomFields::RFsimulate(model = RandomFields::RMspheric(),
                                      x = coords)
  field_3 <- RandomFields::RFsimulate(model = RandomFields::RMwhittle(nu = 2),
                                      x = coords)
  field <-  cbind(field_1, field_2, field_3)
}

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

# 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, rob_whitening = TRUE,
  lcov = 'ldiff', kernel_mat = ring_kernel_list[[1]])
```

Index