Package ‘tensorBSS’

March 1, 2018

Type Package
Title Blind Source Separation Methods for Tensor-Valued Observations
Version 0.3.4
Date 2018-03-01
Author Joni Virta, Bing Li, Klaus Nordhausen, Hannu Oja
Maintainer Joni Virta <joni.virta@outlook.com>
Description Contains several utility functions for manipulating tensor-valued data (centering, multiplication from a single mode etc.) and the implementations of the following blind source separation methods for tensor-valued data: 'tPCA', 'tFOBI', 'tJADE', k-tJADE', 'tgFOBI', 'tgJADE', 'tSOBI', 'tNSS.SD', 'tNSS.JD', 'tNSS.TD.JD' and 'tPP'.
License GPL (>= 2)
Imports Rcpp (>= 0.12.3), tensor, tsBSS, ICtest
LinkingTo Rcpp, RcppArmadillo
Depends JADE
Suggests stochvol, ElemStatLearn
NeedsCompilation yes
Repository CRAN
Date/Publication 2018-03-01 16:32:51 UTC

R topics documented:

  tensorBSS-package ........................................ 2
  k_tJADE ................................................... 3
  mModeAutoCovariance ..................................... 5
  mModeCovariance .......................................... 6
  plot.tbss .................................................. 7
  selectComponents ......................................... 9
  tensorCentering ......................................... 10
  tensorStandardize ....................................... 11
  tensorTransform ......................................... 12
  tensorVectorize ......................................... 13
Description

Contains several utility functions for manipulating tensor-valued data (centering, multiplication from a single mode etc.) and the implementations of the following blind source separation methods for tensor-valued data: ‘tPCA’, ‘tFOBI’, ‘tJADE’, ‘k-tJADE’, ‘tgFOBI’, ‘tgJADE’, ‘tSOBI’, ‘tNSS.SD’, ‘tNSS.JD’, ‘tNSS.TD.JD’ and ‘tPP’.

Details

Package: tensorBSS
Type: Package
Version: 0.3.4
Date: 2018-03-01
License: GPL (>= 2)

Author(s)

Joni Virta, Bing Li, Klaus Nordhausen and Hannu Oja

Maintainer: Joni Virta <joni.virta@outlook.com>

References


---

k_tJADE  

**k-tJADE for Tensor-Valued Observations**

**Description**

Computes the faster “k”-version of tensorial JADE in an independent component model.

**Usage**

```r
k_tJADE(x, k = NULL, maxiter = 100, eps = 1e-06)
```

**Arguments**

- **x**: Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.
- **k**: A vector with one less element than dimensions in `x`. The elements of `k` give upper bounds for cumulant matrix indices we diagonalize in each mode. Lower values mean faster computation times. The default value `NULL` puts `k` equal to 1 in each mode (the fastest choice).
- **maxiter**: Maximum number of iterations. Passed on to `rjd`.
- **eps**: Convergence tolerance. Passed on to `rjd`.

**Details**

It is assumed that `S` is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ with mutually independent elements and measured on $N$ units. The tensor independent component model further assumes that the tensors $S$ are mixed from each mode $m$ by the mixing matrix $A_m$, $m = 1, \ldots, r$, yielding the observed data $X$. In R the sample of $X$ is saved as an array of dimensions $p_1, p_2, \ldots, p_r, N$.

`k_tJADE` recovers then based on `x` the underlying independent components $S$ by estimating the $r$ unmixing matrices $W_1, \ldots, W_r$ using fourth joint moments at the same time in a more efficient way than `tFOBI` but also in fewer numbers than `tJADE`. `k_tJADE` diagonalizes in each mode only those cumulant matrices $C_{ij}$ for which $|i - j| < k_m$.

If `x` is a matrix, that is, $r = 1$, the method reduces to JADE and the function calls `k_JADE`.
Value
A list with class 'tbss', inheriting from class 'bss', containing the following components:

- **S**: Array of the same size as x containing the independent components.
- **W**: List containing all the unmixing matrices
- **Xmu**: The data location.
- **k**: The used vector of k-values.
- **datatype**: Character string with value "iid". Relevant for plot.tbss.

Author(s)
Joni Virta

References


See Also
k_JADE, tJADE, JADE

Examples
```r
n <- 1000
S <- t(cbind(rexp(n)-1, runif(n, -sqrt(3), sqrt(3)), rt(n,5)*sqrt(0.6), (rchisq(n,1)-1)/sqrt(2), (rchisq(n,2)-2)/sqrt(4)))

dim(S) <- c(3, 2, n)

A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)

X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)

k_tjade <- k_tJADE(X)

MD(k_tjade$W[[1]], A1)
MD(k_tjade$W[[2]], A2)
tMD(k_tjade$W, list(A1, A2))
```
k_tjade <- k_tJADE(X, k = c(2, 1))

MD(k_tjade$W[[1]], A1)
MD(k_tjade$W[[2]], A2)
tMD(k_tjade$W, list(A1, A2))

mModeAutoCovariance

Description
Estimates the m-mode autocovariance matrix from an array of array-valued observations with the specified lag.

Usage
mModeAutoCovariance(x, m, lag, center = TRUE)

Arguments
x
Array of order higher than two with the last dimension corresponding to the sampling units.
m
The mode with respect to which the autocovariance matrix is to be computed.
lag
The lag with respect to which the autocovariance matrix is to be computed.
center
Logical, indicating whether the observations should be centered prior to computing the autocovariance matrix. Default is TRUE.

Details
The m-mode autocovariance matrix provides a higher order analogy for the ordinary autocovariance matrix of a random vector and is computed for a random tensor $X_t$ of size $p_1 \times p_2 \times \ldots \times p_r$ as $\text{Cov}_{m\tau}(X_t) = E(X_t^{(m)}X_{t+\tau}^{(m)\text{T}})/(p_1 \ldots p_{m-1}p_{m+1} \ldots p_r)$, where $X_t^{(m)}$ is the centered $m$-flattening of $X_t$ and $\tau$ is the desired lag. The algorithm computes the estimate of this based on the sample $x$.

Value
The m-mode autocovariance matrix of $x$ with respect to $\text{lag}$ having the size $p_m \times p_m$.

Author(s)
Joni Virta

References
See Also

mModeCovariance

Examples

n <- 1000
S <- t(cbind(as.vector(arima.sim(n = n, list(ar = 0.9))),
            as.vector(arima.sim(n = n, list(ar = -0.9))),
            as.vector(arima.sim(n = n, list(ma = c(0.5, -0.5)))),
            as.vector(arima.sim(n = n, list(ar = c(-0.5, -0.3)))),
            as.vector(arima.sim(n = n, list(ar = c(0.5, -0.3, 0.1, -0.1), ma=c(0.7, -0.3)))),
            as.vector(arima.sim(n = n, list(ar = c(-0.7, 0.1), ma = c(0.9, 0.3, 0.1, -0.1))))))
dim(S) <- c(3, 2, n)

mModeAutoCovariance(S, m = 1, lag = 1)
mModeAutoCovariance(S, m = 1, lag = 4)

mModeCovariance  The m-Mode Covariance Matrix

Description

Estimates the m-mode covariance matrix from an array of array-valued observations.

Usage

mModeCovariance(x, m, center = TRUE)

Arguments

x  Array of order higher than two with the last dimension corresponding to the sampling units.

m  The mode with respect to which the covariance matrix is to be computed.

center  Logical, indicating whether the observations should be centered prior to computing the covariance matrix. Default is TRUE.

Details

The m-mode covariance matrix provides a higher order analogy for the ordinary covariance matrix of a random vector and is computed for a random tensor \( X \) of size \( p_1 \times p_2 \times \ldots \times p_r \) as \( \text{Cov}_m(X) = E(X^{(m)}X^{(m)T})/(p_1 \ldots p_{m-1}p_{m+1} \ldots p_r) \), where \( X^{(m)} \) is the centered \( m \)-flattening of \( X \). The algorithm computes the estimate of this based on the sample \( x \).

Value

The \( m \)-mode covariance matrix of \( x \) having the size \( p_m \times p_m \).
plot.tbss

Author(s)

Joni Virta

References


See Also

mModeAutoCovariance

Examples

```r
## Generate sample data.
 n <- 100
x <- t(cbind(rnorm(n, mean = 0),
           rnorm(n, mean = 1),
           rnorm(n, mean = 2),
           rnorm(n, mean = 3),
           rnorm(n, mean = 4),
           rnorm(n, mean = 5)))
dim(x) <- c(3, 2, n)

# The m-mode covariance matrices of the first and second modes
mModeCovariance(x, 1)
mModeCovariance(x, 2)
```

plot.tbss

Plot an Object of the Class tbss

Description

Plots the most interesting components (in the sense of extreme kurtosis) obtained by a tensor blind source separation method.

Usage

```r
## S3 method for class 'tbss'
plot(x, first = 2, last = 2, datatype = NULL,
     main = "The components with most extreme kurtoses", ...)
```
Arguments

- **x**: Object of class `tbss`.
- **first**: Number of components with maximal kurtosis to be selected. See `selectComponents` for details.
- **last**: Number of components with minimal kurtosis to be selected. See `selectComponents` for details.
- **main**: The title of the plot.
- **datatype**: Parameter for choosing the type of plot, either `NULL`, "iid" or "ts". The default `NULL` means the value from the `tbss` object `x` is taken.
- **...**: Further arguments to be passed to the plotting functions, see details.

Details

The function `plot.tbss` first selects the most interesting components using `selectComponents` and then plots them either as a matrix of scatter plots using `pairs` (datatype = "iid") or as a time series plot using `plot.ts` (datatype = "ts"). Note that for `tsobi` this criterion might not necessarily be meaningful as the method is based on second moments only.

Author(s)

Joni Virta

Examples

```r
library(ElemStatLearn)
x <- zip.train

rows <- which(x[, 1] == 0 | x[, 1] == 1)
x0 <- x[rows, 2:257]
y0 <- x[rows, 1] + 1

x0 <- t(x0)
dim(x0) <- c(16, 16, 2199)

tfobi <- tfOBI(x0)
plot(tfobi, col=y0)

library("stochvol")
n <- 1000
S <- t(cbind(svsim(n, mu = -10, phi = 0.98, sigma = 0.2, nu = Inf)$y,
                 svsim(n, mu = -5, phi = -0.98, sigma = 0.2, nu = 10)$y,
                 svsim(n, mu = -10, phi = 0.70, sigma = 0.7, nu = Inf)$y,
                 svsim(n, mu = -5, phi = -0.70, sigma = 0.7, nu = 10)$y,
                 svsim(n, mu = -9, phi = 0.20, sigma = 0.01, nu = Inf)$y,
                 svsim(n, mu = -9, phi = -0.20, sigma = 0.01, nu = 10)$y))
dim(S) <- c(3, 2, n)
A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)
```
selectComponents

X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)

tgfobi <- tgfobi(X)
plot(tgfobi, 1, 1)

---

**selectComponents**

Select the Most Informative Components

**Description**

Takes an array of observations as an input and outputs a subset of the components having the most extreme kurtoses.

**Usage**

selectComponents(x, first = 2, last = 2)

**Arguments**

- **x**: Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.
- **first**: Number of components with maximal kurtosis to be selected. Can equal zero but the total number of components selected must be at least two.
- **last**: Number of components with minimal kurtosis to be selected. Can equal zero but the total number of components selected must be at least two.

**Details**

In independent component analysis (ICA) the components having the most extreme kurtoses are often thought to be also the most informative. With this viewpoint in mind the function `selectComponents` selects from `x` first components having the highest kurtosis and last components having the lowest kurtoses and outputs them as a standard data matrix for further analysis.

**Value**

Data matrix with rows corresponding to the observations and the columns corresponding to the first + last selected components in decreasing order with respect to kurtosis. The names of the components in the output matrix correspond to the indices of the components in the original array `x`.

**Author(s)**

Joni Virta
Examples

```r
library(ElemStatLearn)
x <- zip.train

rows <- which(x[, 1] == 0 | x[, 1] == 1)
x0 <- x[rows, 2:257]

x0 <- t(x0)
dim(x0) <- c(16, 16, 2199)

tfobi <- tfobi(x0)
comp <- selectComponents(tfobi$S)
head(comp)
```

tensorCentering  

Center an Array of Observations

Description

Centers an array of array-valued observations by subtracting the mean array from each observation.

Usage

```r
tensorCentering(x)
```

Arguments

- `x`  
  Array of order at least two with the last dimension corresponding to the sampling units.

Details

Centers a $p_1 \times p_2 \times \ldots \times p_r \times n$-dimensional array by subtracting the $p_1 \times p_2 \times \ldots \times p_r$-dimensional array of element-wise means from each of the observed arrays.

Value

Array of centered observations with the same dimensions as the input array.

Author(s)

Joni Virta
## Examples

```r
## Generate sample data.
n <- 1000
x <- t(cbind(rnorm(n, mean = 0),
             rnorm(n, mean = 1),
             rnorm(n, mean = 2),
             rnorm(n, mean = 3),
             rnorm(n, mean = 4),
             rnorm(n, mean = 5)))
dim(x) <- c(3, 2, n)

## Centered data
xcen <- tensorCentering(x)

## Check the means of individual cells
apply(xcen, 1:2, mean)
```

### tensorStandardize

**Standardize an Observation Array**

#### Description

Standardizes an array of array-valued observations simultaneously from each mode. The method can be seen as a higher-order analogy for the regular multivariate standardization of random vectors.

#### Usage

`tensorStandardize(x)`

#### Arguments

- **x**: Array of an order higher than two with the last dimension corresponding to the sampling units.

#### Details

The algorithm first centers the $n$ observed tensors $X_i$ to have an element-wise mean of zero. Then it estimates the $m$th mode covariance matrix $\text{Cov}_m(X) = E(X^{(m)}X^{(m)T})/(p_1 \ldots p_m \ldots p_{m+1} \ldots p_r)$, where $X^{(m)}$ is the centered $m$-flattening of $X$, for each mode and transforms the observations with the inverse square roots of the covariance matrices from the corresponding modes.

#### Value

A list containing the following components:

- **x**: Array of the same size as `x` containing the standardized observations.
- **s**: List containing inverse square roots of the covariance matrices of different modes.
**Author(s)**

Joni Virta

**Examples**

```r
# Generate sample data.
n <- 100
x <- t(cbind(rnorm(n, mean = 0),
            rnorm(n, mean = 1),
            rnorm(n, mean = 2),
            rnorm(n, mean = 3),
            rnorm(n, mean = 4),
            rnorm(n, mean = 5)))
dim(x) <- c(3, 2, n)

# Standardize
z <- tensorStandardize(x)$x

# The m-mode covariance matrices of the standardized tensors
mModeCovariance(z, 1)
mModeCovariance(z, 2)
```

---

**tensortransform**

*Linear Transformation of Tensors from mth Mode*

**Description**

Applies a linear transformation to the mth mode of each individual tensor in an array of tensors

**Usage**

`tensortransform(x, A, m)`

**Arguments**

- **x**: Array of an order at least two with the last dimension corresponding to the sampling units.
- **A**: Matrix corresponding to the desired linear transformation with the number of columns equal to the size of the mth dimension of x.
- **m**: The mode from which the linear transform is to be applied.

**Details**

Applies the linear transformation given by the matrix $A$ of size $q_m \times p_m$ to the mth mode of each of the $n$ observed tensors $X_i$ in the given $p_1 \times p_2 \times \ldots \times p_r \times n$-dimensional array $x$. This is equivalent to separately applying the linear transformation given by $A$ to each m-mode vector of each $X_i$. 

---
Value

Array of size $p_1 \times p_2 \times \ldots \times p_{m-1} \times q_m \times p_{m+1} \times \ldots \times p_r \times n$

Author(s)

Joni Virta

Examples

```r
# Generate sample data.
n <- 10
x <- t(cbind(rnorm(n, mean = 0),
             rnorm(n, mean = 1),
             rnorm(n, mean = 2),
             rnorm(n, mean = 3),
             rnorm(n, mean = 4),
             rnorm(n, mean = 5)))
dim(x) <- c(3, 2, n)

# Transform from the second mode
A <- matrix(c(2, 1, 0, 3), 2, 2)
z <- tensorTransform(x, A, 2)

# Compare
z[, , 1]
x[, , 1] %*% t(A)
```

Description

Vectorizes an array of array-valued observations into a matrix so that each column of the matrix corresponds to a single observational unit.

Usage

tensorVectorize(x)

Arguments

x Array of an order at least two with the last dimension corresponding to the sampling units.

Details

Vectorizes a $p_1 \times p_2 \times \ldots \times p_r \times n$-dimensional array into a $p_1 p_2 \ldots p_r \times n$-dimensional matrix, each column of which then corresponds to a single observational unit. The vectorization is done so that the $r$th index goes through its cycle the fastest and the first index the slowest.
Value

Matrix whose columns contain the vectorized observed tensors.

Author(s)

Joni Virta

Examples

```r
# Generate sample data.
N <- 100
x <- t(cbind(rnorm(N, mean = 0),
            rnorm(N, mean = 1),
            rnorm(N, mean = 2),
            rnorm(N, mean = 3),
            rnorm(N, mean = 4),
            rnorm(N, mean = 5)))

dim(x) <- c(3, 2, N)

# Matrix of vectorized observations.
vecx <- tensorVectorize(x)

cov(t(vecx))
```

Description

Computes the tensorial FOBI in an independent component model.

Usage

tFOBI(x, norm = NULL)

Arguments

- **x**
  Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.

- **norm**
  A Boolean vector with number of entries equal to the number of modes in a single observation. The elements tell which modes use the "normed" version of tensorial FOBI. If NULL then all modes use the non-normed version.
Details

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ with mutually independent elements and measured on $N$ units. The tensor independent component model further assumes that the tensors $S$ are mixed from each mode $m$ by the mixing matrix $A_m$, $m = 1,\ldots,r$, yielding the observed data $X$. In R the sample of $X$ is saved as an array of dimensions $p_1,p_2,\ldots,p_r,N$.

$tFOBI$ recovers then based on $x$ the underlying independent components $S$ by estimating the $r$ unmixing matrices $W_1,\ldots,W_r$ using fourth joint moments.

The unmixing can in each mode be done in two ways, using a “non-normed” or “normed” method and this is controlled by the argument norm. The authors advocate the general use of non-normed version, see the reference below for their comparison.

If $x$ is a matrix, that is, $r = 1$, the method reduces to FOBI and the function calls $FOBI$.

For a generalization for tensor-valued time series see $tgFOBI$.

Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

- **S**  
  Array of the same size as $x$ containing the independent components.

- **W**  
  List containing all the unmixing matrices.

- **norm**  
  The vector indicating which modes used the “normed” version.

- **Xmu**  
  The data location.

- **datatype**  
  Character string with value "iid". Relevant for $plot.tbss$.

Author(s)

Joni Virta

References


See Also

$FOBI$, $tgFOBI$

Examples

```r
n <- 1000
S <- t(cbind(rexp(n)-1,  
    rnorm(n),  
    runif(n, -sqrt(3), sqrt(3)),  
    rt(n,5)*sqrt(0.6),  
    (rchisq(n,1)-1)/sqrt(2),  
    (rchisq(n,2)-2)/sqrt(4)))

dim(S) <- c(3, 2, n)
```
### Description

Computes the tensorial gFOBI for time series where at each time point a tensor of order \( r \) is observed.

### Usage

\[
gFOBI(x, \text{lags} = 0:12, \text{maxiter} = 100, \text{eps} = 1e-06)
\]

### Arguments

- **x**: Numeric array of an order at least two. It is assumed that the last dimension corresponds to the time.
- **lags**: Vector of integers. Defines the lags used for the computations of the autocovariances.
- **maxiter**: Maximum number of iterations. Passed on to \( \text{rjd} \).
- **eps**: Convergence tolerance. Passed on to \( \text{rjd} \).
**Details**

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ measured at time points $1, \ldots, T$. The assumption is that the elements of $S$ are mutually independent, centered and weakly stationary time series and are mixed from each mode $m$ by the mixing matrix $A_m$, $m = 1, \ldots, r$, yielding the observed time series $X$. In R the sample of $X$ is saved as an array of dimensions $p_1, p_2, \ldots, p_r, T$.

tgFOBI recovers then based on $x$ the underlying independent time series $S$ by estimating the $r$ unmixing matrices $W_1, \ldots, W_r$ using the lagged fourth joint moments specified by `lags`. This reliance on higher order moments makes the method especially suited for stochastic volatility models.

If $x$ is a matrix, that is, $r = 1$, the method reduces to gFOBI and the function calls `gFOBI`.

If `lags = 0` the method reduces to `tfOBI`.

**Value**

A list with class 'tbss', inheriting from class 'bss', containing the following components:

- `S` Array of the same size as $x$ containing the estimated uncorrelated sources.
- `W` List containing all the unmixing matrices
- `xmu` The data location.
- `datatype` Character string with value "ts". Relevant for `plot.tbss`.

**Author(s)**

Joni Virta

**References**


**See Also**

`gFOBI`, `rjd`, `tfOBI`

**Examples**

```r
library("stochvol")
n <- 1000
S <- t(cbind(svasm(n, mu = -10, phi = 0.98, sigma = 0.2, nu = Inf)$y,
               svsm(n, mu = -5, phi = -0.98, sigma = 0.2, nu = 10)$y,
               svsm(n, mu = -10, phi = 0.70, sigma = 0.7, nu = Inf)$y,
               svsm(n, mu = -5, phi = -0.70, sigma = 0.7, nu = 10)$y,
               svsm(n, mu = -9, phi = 0.20, sigma = 0.01, nu = Inf)$y,
               svsm(n, mu = -9, phi = -0.20, sigma = 0.01, nu = 10)$y))
dim(S) <- c(3, 2, n)
A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)
X <- tensorTransform(S, A1, 1)
```
tgJADE

gJADE for Tensor-Valued Time Series

Description
Computes the tensorial gJADE for time series where at each time point a tensor of order \( r \) is observed.

Usage

tgJADE(x, lags = 0:12, maxiter = 100, eps = 1e-06)

Arguments

- **x**: Numeric array of an order at least two. It is assumed that the last dimension corresponds to the time.
- **lags**: Vector of integers. Defines the lags used for the computations of the autocovariances.
- **maxiter**: Maximum number of iterations. Passed on to \( \text{rjd} \).
- **eps**: Convergence tolerance. Passed on to \( \text{rjd} \).

Details
It is assumed that \( S \) is a tensor (array) of size \( p_1 \times p_2 \times \ldots \times p_r \) measured at time points \( 1, \ldots, T \). The assumption is that the elements of \( S \) are mutually independent, centered and weakly stationary time series and are mixed from each mode \( m = 1, \ldots, r \) by the mixing matrix \( A_m \), yielding the observed time series \( X \). In R the sample of \( X \) is saved as an array of dimensions \( p_1, p_2, \ldots, p_r, T \).

tgJADE recovers then based on \( x \) the underlying independent time series \( S \) by estimating the \( r \) unmixing matrices \( W_1, \ldots, W_r \) using the lagged fourth joint moments specified by \( \text{lags} \). This reliance on higher order moments makes the method especially suited for stochastic volatility models.

If \( x \) is a matrix, that is, \( r = 1 \), the method reduces to gJADE and the function calls \( \text{gJADE} \).

If \( \text{lags} = \emptyset \) the method reduces to \( \text{tJADE} \).
Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

- **S**
  Array of the same size as x containing the estimated uncorrelated sources.

- **W**
  List containing all the unmixing matrices

- **Xmu**
  The data location.

- **datatype**
  Character string with value "ts". Relevant for plot.tbss.

Author(s)

Joni Virta

References


See Also

gJADE, rjd, tJADE

Examples

```r
library("stochvol")
n <- 1000
S <- t(cbind(svsim(n, mu = -10, phi = 0.98, sigma = 0.2, nu = Inf)$y,
              svsim(n, mu = -5, phi = -0.98, sigma = 0.2, nu = 10)$y,
              svsim(n, mu = -10, phi = 0.70, sigma = 0.7, nu = Inf)$y,
              svsim(n, mu = -5, phi = -0.70, sigma = 0.7, nu = 10)$y,
              svsim(n, mu = -9, phi = 0.20, sigma = 0.01, nu = Inf)$y,
              svsim(n, mu = -9, phi = -0.20, sigma = 0.01, nu = 10)$y))
  dim(S) <- c(3, 2, n)

A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)

X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)

tgjade <- tgJADE(X)

MD(tgjade$W[[1]], A1)
MD(tgjade$W[[2]], A2)
tMD(tgjade$W, list(A1, A2))
```
Description

Computes the tensorial JADE in an independent component model.

Usage

tJADE(x, maxiter = 100, eps = 1e-06)

Arguments

x Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.
maxiter Maximum number of iterations. Passed on to rjd.
eps Convergence tolerance. Passed on to rjd.

Details

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ with mutually independent elements and measured on $N$ units. The tensor independent component model further assumes that the tensors $S$ are mixed from each mode $m$ by the mixing matrix $A_m$, $m = 1, \ldots, r$, yielding the observed data $X$. In R the sample of $X$ is saved as an array of dimensions $p_1, p_2, \ldots, p_r, N$.

tJADE recovers then based on $x$ the underlying independent components $S$ by estimating the $r$ unmixing matrices $W_1, \ldots, W_r$ using fourth joint moments in a more efficient way than tFOBI. If $x$ is a matrix, that is, $r = 1$, the method reduces to JADE and the function calls JADE.

For a generalization for tensor-valued time series see tgJADE.

Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

S Array of the same size as $x$ containing the independent components.
W List containing all the unmixing matrices
Xmu The data location.
datatype Character string with value "iid". Relevant for plot.tbss.

Author(s)

Joni Virta

References

See Also

\textit{JADE, tgJADE}

Examples

\begin{verbatim}
  n <- 1000
  S <- t(cbind(rexp(n)-1,
              rnorm(n),
              runif(n, -sqrt(3), sqrt(3)),
              rt(n,5)*sqrt(0.6),
              (rchisq(n,1)-1)/sqrt(2),
              (rchisq(n,2)-2)/sqrt(4)))
  dim(S) <- c(3, 2, n)
  A1 <- matrix(rnorm(9), 3, 3)
  A2 <- matrix(rnorm(4), 2, 2)
  X <- tensorTransform(S, A1, 1)
  X <- tensorTransform(X, A2, 2)
  tjade <- tJADE(X)

  MD(tjade$W[[1]], A1)
  MD(tjade$W[[2]], A2)
  tMD(tjade$W, list(A1, A2))

  ## Not run:
  # Digit data example
  # Running will take a few minutes

  library(ElemStatLearn)
  x <- zip.train
  rows <- which(x[, 1] == 0 | x[, 1] == 1)
  x0 <- x[rows, 2:257]
  y0 <- x[rows, 1] + 1
  x0 <- t(x0)
  dim(x0) <- c(16, 16, 2199)
  tjade <- tJADE(x0)
  plot(tjade, col=y0)

  ## End(Not run)
\end{verbatim}
Description

A shortcut function for computing the minimum distance index of a tensorial ICA estimate on the Kronecker product “scale” (the vectorized space).

Usage

tMD(W.hat, A)

Arguments

W.hat  A list of r unmixing matrix estimates, W_1, W_2, ..., W_r.
A       A list of r mixing matrices, A_1, A_2, ..., A_r.

Details

The function computes the minimum distance index between W.hat[[r]] xx% ... xx% W.hat[[1]] and A[[r]] xx% ... xx% A[[1]]. The index is useful for comparing the performance of a tensor-valued ICA method to that of a method using first vectorization and then some vector-valued ICA method.

Value

The value of the MD index of the Kronecker product.

Author(s)

Joni Virta

References


See Also

MD

Examples

n <- 1000
S <- t(cbind(rexp(n)-1,
            rnorm(n),
            runif(n, -sqrt(3), sqrt(3)),
            rt(n,5)*sqrt(0.6),
            rchisq(n,1)-1)/sqrt(2),
            rchisq(n,2)-2)/sqrt(4)))
dim(S) <- c(3, 2, n)
A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)
X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)
tfobi <- tFOBI(X)
MD(tfobi$W[[2]] %x% tfobi$W[[1]], A2 %x% A1)
tMD(list(tfobi$W[[2]]), list(A2))

---

tNSS.JD

NSS-JD Method for Tensor-Valued Time Series

**Description**

Estimates the non-stationary sources of a tensor-valued time series using separation information contained in several time intervals.

**Usage**

```r
tNSS.JD(x, K = 12, n.cuts = NULL, eps = 1e-06, maxiter = 100, ...)
```

**Arguments**

- **x** Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.
- **K** The number of equisized intervals into which the time range is divided. If the parameter n.cuts is non-NULL it takes preference over this argument.
- **n.cuts** Either a interval cutoffs (the cutoffs are used to define the two intervals that are open below and closed above, e.g. \((a, b]\)) or NULL (the parameter K is used to define the the amount of intervals).
- **eps** Convergence tolerance for rjd.
- **maxiter** Maximum number of iterations for rjd.
- **...** Further arguments to be passed to or from methods.

**Details**

Assume that the observed tensor-valued time series comes from a tensorial BSS model where the sources have constant means over time but the component variances change in time. Then TNSSS-JD first standardizes the series from all modes and then estimates the non-stationary sources by dividing the time scale into K intervals and jointly diagonalizing the covariance matrices of the K intervals within each mode.
Value
A list with class 'tbss', inheriting from class 'bss', containing the following components:

S       Array of the same size as x containing the independent components.
W       List containing all the unmixing matrices.
K       The number of intervals.
n.cuts  The interval cutoffs.
Xmu     The data location.
datatype Character string with value "ts". Relevant for plot.tbss.

Author(s)
Joni Virta

References

See Also
NSS SD, NSS JD, NSS TD JD, tNSS SD, tNSS TD JD

Examples

# Create innovation series with block-wise changing variances
n1 <- 200
n2 <- 500
n3 <- 300
n <- n1 + n2 + n3
innov1 <- c(rnorm(n1, 0, 1), rnorm(n2, 0, 3), rnorm(n3, 0, 5))
innov2 <- c(rnorm(n1, 0, 1), rnorm(n2, 0, 5), rnorm(n3, 0, 3))
innov3 <- c(rnorm(n1, 0, 5), rnorm(n2, 0, 3), rnorm(n3, 0, 1))
innov4 <- c(rnorm(n1, 0, 5), rnorm(n2, 0, 1), rnorm(n3, 0, 3))

# Generate the observations
vecx <- cbind(as.vector(arima.sim(n = n, list(ar = 0.8, innov = innov1)),
                        as.vector(arima.sim(n = n, list(ar = c(0.5, 0.1)), innov = innov2)),
                        as.vector(arima.sim(n = n, list(ma = -0.7), innov = innov3)),
                        as.vector(arima.sim(n = n, list(ar = 0.5, ma = -0.5), innov = innov4)))

# Vector to tensor
tenx <- t(vecx)
dim(tenx) <- c(2, 2, n)

# Run TNSS- JD
res <- tNNS JD(tenx, K = 6)
res$W
tNSS.SD

res <- tNSS.JD(tenx, K = 12)
res$W

---

**tNSS.SD**

NSS-SD Method for Tensor-Valued Time Series

**Description**

Estimates the non-stationary sources of a tensor-valued time series using separation information contained in two time intervals.

**Usage**

tNSS.SD(x, n.cuts = NULL)

**Arguments**

- **x**
  Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.

- **n.cuts**
  Either a 3-vector of interval cutoffs (the cutoffs are used to define the two intervals that are open below and closed above, e.g. \((a, b]\)) or NULL (the time range is sliced into two parts of equal size).

**Details**

Assume that the observed tensor-valued time series comes from a tensorial BSS model where the sources have constant means over time but the component variances change in time. Then TNSS-SD estimates the non-stationary sources by dividing the time scale into two intervals and jointly diagonalizing the covariance matrices of the two intervals within each mode.

**Value**

A list with class 'tbss', inheriting from class 'bss', containing the following components:

- **S**
  Array of the same size as x containing the independent components.

- **W**
  List containing all the unmixing matrices.

- **EV**
  Eigenvalues obtained from the joint diagonalization.

- **n.cuts**
  The interval cutoffs.

- **xmu**
  The data location.

- **datatype**
  Character string with value "ts". Relevant for plot.tbss.

**Author(s)**

Joni Virta
References


See Also

nss.NSD, nss.NJD, nss.NTD.JD, tNSS.JD, tNSS.TD.JD

Examples

# Create innovation series with block-wise changing variances

# 9 smooth variance structures
var_1 <- function(n){
  t <- 1:n
  return(1 + cos((2*pi*t)/n)*sin((2*150*t)/(n*pi)))
}

var_2 <- function(n){
  t <- 1:n
  return(1 + sin((2*pi*t)/n)*cos((2*150*t)/(n*pi)))
}

var_3 <- function(n){
  t <- 1:n
  return(0.5 + 8*exp((n+1)^2/(4*n*(t - n - 1))))
}

var_4 <- function(n){
  t <- 1:n
  return(3.443 - 8*exp((n+1)^2/(4*n*(t - n - 1))))
}

var_5 <- function(n){
  t <- 1:n
  return(0.5 + 0.5*gamma(10)/(gamma(7)*gamma(3))*(t/(n + 1))^7*1 - t/(n + 1))^3
}

var_6 <- function(n){
  t <- 1:n
  res <- var_5(n)
  return(rev(res))
}

var_7 <- function(n){
  t <- 1:n
  return(0.2 + 2*t/(n + 1))
}

var_8 <- function(n){
  t <- 1:n

return(0.2+2*(n+1-t)/(n+1))
}

var_9 <- function(n){
  t <- 1:n
  return(1.5 + cos(4*pi*t/n))
}

# Innovation series
n <- 1000

innov1 <- c(rnorm(n, 0, sqrt(var_1(n))))
innov2 <- c(rnorm(n, 0, sqrt(var_2(n))))
innov3 <- c(rnorm(n, 0, sqrt(var_3(n))))
innov4 <- c(rnorm(n, 0, sqrt(var_4(n))))
innov5 <- c(rnorm(n, 0, sqrt(var_5(n))))
innov6 <- c(rnorm(n, 0, sqrt(var_6(n))))
innov7 <- c(rnorm(n, 0, sqrt(var_7(n))))
innov8 <- c(rnorm(n, 0, sqrt(var_8(n))))
innov9 <- c(rnorm(n, 0, sqrt(var_9(n))))

# Generate the observations
vecx <- cbind(as.vector(arima.sim(n = n, list(ar = 0.9), innov = innov1)),
              as.vector(arima.sim(n = n, list(ar = c(0, 0.2, 0.1, -0.1, 0.7)),
                                 innov = innov2)),
              as.vector(arima.sim(n = n, list(ar = c(0.5, 0.3, -0.2, 0.1)),
                                 innov = innov3)),
              as.vector(arima.sim(n = n, list(ma = -0.5), innov = innov4)),
              as.vector(arima.sim(n = n, list(ma = c(0.1, 0.1, 0.3, 0.5, 0.8)),
                                 innov = innov5)),
              as.vector(arima.sim(n = n, list(ma = c(0.5, -0.5, 0.5)), innov = innov6)),
              as.vector(arima.sim(n = n, list(ar = -0.5, -0.3), ma = c(-0.2, 0.1)),
                                 innov = innov7)),
              as.vector(arima.sim(n = n, list(ar = c(0, -0.1, -0.2, 0.5), ma = c(0, 0.1, 0.1, 0.6)),
                                 innov = innov8)),
              as.vector(arima.sim(n = n, list(ar = c(0.8), ma = c(0.7, 0.6, 0.5, 0.1)),
                                 innov = innov9)))

# Vector to tensor
tenx <- t(vecx)
dim(tenx) <- c(3, 3, n)

# Run TNSS-SD
res <- tNSS.SD(tenx)
res$W
Description

Estimates the non-stationary sources of a tensor-valued time series using separation information contained in several time intervals and lags.

Usage

tNSS.TD.JD(x, K = 12, lags = 0:12, n.cuts = NULL, eps = 1e-06, maxiter = 100, ...)

Arguments

x  Numeric array of an order at least two. It is assumed that the last dimension corresponds to the sampling units.
K  The number of equisized intervals into which the time range is divided. If the parameter n.cuts is non-NULL it takes preference over this argument.
lags The lag set for the autocovariance matrices.
n.cuts Either a interval cutoffs (the cutoffs are used to define the two intervals that are open below and closed above, e.g. (a, b]) or NULL (the parameter K is used to define the amount of intervals).
eps  Convergence tolerance for rjd.
maxiter Maximum number of iterations for rjd.
... Further arguments to be passed to or from methods.

Details

Assume that the observed tensor-valued time series comes from a tensorial BSS model where the sources have constant means over time but the component variances change in time. Then TNSS-TD-JD first standardizes the series from all modes and then estimates the non-stationary sources by dividing the time scale into K intervals and jointly diagonalizing the autocovariance matrices (specified by lags) of the K intervals within each mode.

Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

S  Array of the same size as x containing the independent components.
W  List containing all the unmixing matrices.
K  The number of intervals.
lags The lag set.
n.cuts The interval cutoffs.
xmu The data location.
datatype Character string with value "ts". Relevant for plot.tbss.

Author(s)

Joni Virta
References


See Also

nss.NSD, nss.NJD, nss.TSD.JD, tNSS.NSD, tNSS.JD

Examples

# Create innovation series with block-wise changing variances
n1 <- 200
n2 <- 500
n3 <- 300
n <- n1 + n2 + n3
innov1 <- c(rnorm(n1, 0, 1), rnorm(n2, 0, 3), rnorm(n3, 0, 5))
innov2 <- c(rnorm(n1, 0, 1), rnorm(n2, 0, 5), rnorm(n3, 0, 3))
innov3 <- c(rnorm(n1, 0, 5), rnorm(n2, 0, 3), rnorm(n3, 0, 1))
innov4 <- c(rnorm(n1, 0, 5), rnorm(n2, 0, 1), rnorm(n3, 0, 3))

# Generate the observations
vecx <- cbind(as.vector(arima.sim(n = n, list(ar = 0.8), innov = innov1)),
              as.vector(arima.sim(n = n, list(ar = c(0.5, 0.1)), innov = innov2)),
              as.vector(arima.sim(n = n, list(ma = -0.7), innov = innov3)),
              as.vector(arima.sim(n = n, list(ar = 0.5, ma = -0.5), innov = innov4)))

# Vector to tensor
tenx <- t(vecx)
dim(tenx) <- c(2, 2, n)

# Run TNSS-TD-JD
res <- tNSS.TD.JD(tenx)
resW

res <- tNSS.TD.JD(tenx, K = 6, lags = 0:6)
resW

tPCA

PCA for Tensor-Valued Observations

Description

Computes the tensorial principal components.

Usage

tPCA(x, p = NULL, d = NULL)
Arguments

x Numeric array of an order at least three. It is assumed that the last dimension corresponds to the sampling units.

p A vector of the percentages of variation per each mode the principal components should explain.

d A vector of the exact number of components retained per each mode. At most one of this and the previous argument should be supplied.

Details

The observed tensors (array) \(X\) of size \(p_1 \times p_2 \times \ldots \times p_r\) measured on \(N\) units are projected from each mode on the eigenspaces of the \(m\)-mode covariance matrices of the corresponding modes. As in regular PCA, by retaining only some subsets of these projections (indices) with respective sizes \(d_1, d_2, \ldots, d_r\), a dimension reduction can be carried out, resulting into observations tensors of size \(d_1 \times d_2 \times \ldots \times d_r\). In R the sample of \(X\) is saved as an array of dimensions \(p_1, p_2, \ldots, p_r, N\).

Value

A list containing the following components:

S Array of the same size as \(x\) containing the principal components.

U List containing the rotation matrices

D List containing the amounts of variance explained by each index in each mode.

p.comp The percentages of variation per each mode that the principal components explain.

xmu The data location.

Author(s)

Joni Virta

References


Examples

# Digit data example

library(ElemStatLearn)
x <- zip.train

rows <- which(x[, 1] == 0 | x[, 1] == 1)
x0 <- x[rows, 2:257]
y0 <- x[rows, 1] + 1

x0 <- t(x0)
tPP

Projection pursuit for Tensor-Valued Observations

Description

Applies mode-wise projection pursuit to tensorial data with respect to the chosen measure of interestingness.

Usage

\texttt{tPP(x, nl = "pow3", eps = 1e-6, maxiter = 100)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} Numeric array of an order at least three. It is assumed that the last dimension corresponds to the sampling units.
  \item \texttt{nl} \hspace{1cm} The chosen measure of interestingness/objective function. Current choices include \texttt{pow3} (default) and \texttt{skew}, see the details below
  \item \texttt{eps} \hspace{1cm} The convergence tolerance of the iterative algorithm.
  \item \texttt{maxiter} \hspace{1cm} The maximum number of iterations.
\end{itemize}

Details

The observed tensors (arrays) $X$ of size $p_1 \times p_2 \times \ldots \times p_r$ measured on $N$ units are standardized from each mode and then projected mode-wise onto the directions that maximize the $L_2$-norm of the vector of the values $E[G(u_k^TX X^T u_k)] - E[G(c^2)]$, where $G$ is the chosen objective function and $c^2$ obeys the chi-squared distribution with $q$ degrees of freedom. Currently the function allows the choices $G(x) = x^2$ (\texttt{pow3}) and $G(x) = x\sqrt{x}$ (\texttt{skew}), which correspond roughly to the maximization of kurtosis and skewness, respectively. The algorithm is the multilinear extension of FastICA, where the names of the objective functions also come from.

Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

\begin{itemize}
  \item \texttt{S} \hspace{1cm} Array of the same size as \texttt{x} containing the estimated components.
  \item \texttt{W} \hspace{1cm} List containing all the unmixing matrices.
  \item \texttt{iter} \hspace{1cm} The numbers of iteration used per mode.
  \item \texttt{xmu} \hspace{1cm} The data location.
  \item \texttt{datatype} \hspace{1cm} Character string with value "iid". Relevant for \texttt{plot.tbss}.
\end{itemize}
Author(s)
Joni Virta

References

See Also
fICA, NGPP

Examples
n <- 1000
S <- t(cbind(rexp(n)-1,
          rnorm(n),
          runif(n, -sqrt(3), sqrt(3)),
          rt(n,5)*sqrt(0.6),
          (rchisq(n,1)-1)/sqrt(2),
          (rchisq(n,2)-2)/sqrt(4)))

dim(S) <- c(3, 2, n)
A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)
X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)
tpp <- tPP(X)
MD(tpp$W[[1]], A1)
MD(tpp$W[[2]], A2)
tMD(tpp$W, list(A1, A2))

---

tSIR

SIR for Tensor-Valued Observations

Description
Computes the tensorial SIR.

Usage
tSIR(x, y, h = 10, ...
Arguments

- **x**: Numeric array of an order at least three. It is assumed that the last dimension corresponds to the sampling units.
- **y**: A numeric or factor response vector.
- **h**: The number of slices. If `y` is a factor the number of factor levels is automatically used as the number of slices.
- **...**: Arguments passed on to `quantile`.

Details

Computes the mode-wise sliced inverse regression (SIR) estimators for a tensor-valued data set and a univariate response variable.

Value

A list with class 'tbss', inheriting from class 'bss', containing the following components:

- **S**: Array of the same size as `x` containing the predictors.
- **W**: List containing all the unmixing matrices.
- **Xmu**: The data location.
- **datatype**: Character string with value "iid". Relevant for `plot.tbss`.

Author(s)

Joni Virta, Klaus Nordhausen

Examples

```r
library(ElemStatLearn)
x <- zip.train

rows <- which(x[, 1] == 0 | x[, 1] == 3)
x0 <- x[rows, 2:257]
y0 <- as.factor(x[rows, 1])
x0 <- t(x0)
dim(x0) <- c(16, 16, length(y0))

res <- tSIR(x0, y0)
plot(res$S[1, 1, ], res$S[1, 2, ], col = y0)
```
SOBI for Tensor-Valued Time Series

Description

Computes the tensorial SOBI for time series where at each time point a tensor of order \( r \) is observed.

Usage

\[
\text{tSOBI}(x, \text{lags} = 1:12, \text{maxiter} = 100, \text{eps} = 1e-06)
\]

Arguments

- \( x \) Numeric array of an order at least two. It is assumed that the last dimension corresponds to the time.
- \( \text{lags} \) Vector of integers. Defines the lags used for the computations of the autocovariances.
- \( \text{maxiter} \) Maximum number of iterations. Passed on to \text{rjd}.
- \( \text{eps} \) Convergence tolerance. Passed on to \text{rjd}.

Details

It is assumed that \( S \) is a tensor (array) of size \( p_1 \times p_2 \times \ldots \times p_r \) measured at time points \( 1, \ldots, T \). The assumption is that the elements of \( S \) are uncorrelated, centered and weakly stationary time series and are mixed from each mode \( m \) by the mixing matrix \( A_m, m = 1, \ldots, r \), yielding the observed time series \( X \). In R the sample of \( X \) is saved as an \text{array} of dimensions \( p_1, p_2, \ldots, p_r, T \). tSOBI recovers then based on \( x \) the underlying uncorrelated time series \( S \) by estimating the \( r \) unmixing matrices \( W_1, \ldots, W_r \) using the lagged joint autocovariances specified by \( \text{lags} \).

If \( x \) is a matrix, that is, \( r = 1 \), the method reduces to SOBI and the function calls \text{SOBI}.

Value

A list with class ‘tbss’, inheriting from class ‘bss’, containing the following components:

- \( S \) Array of the same size as \( x \) containing the estimated uncorrelated sources.
- \( W \) List containing all the unmixing matrices
- \( \text{Xmu} \) The data location.
- \( \text{datatype} \) Character string with value "ts". Relevant for \text{plot.tbss}.

Author(s)

Joni Virta
References


See Also

SOBI, rjd

Examples

```r
n <- 1000
S <- t(cbind(as.vector(arima.sim(n = n, list(ar = 0.9))),
           as.vector(arima.sim(n = n, list(ar = -0.9))),
           as.vector(arima.sim(n = n, list(ma = c(0.5, -0.5)))),
           as.vector(arima.sim(n = n, list(ar = c(-0.5, -0.3)))),
           as.vector(arima.sim(n = n, list(ar = c(0.5, -0.3, 0.1, -0.1), ma=c(0.7, -0.3)))),
           as.vector(arima.sim(n = n, list(ar = c(-0.7, 0.1), ma = c(0.9, 0.3, 0.1, -0.1))))))
dim(S) <- c(3, 2, n)

A1 <- matrix(rnorm(9), 3, 3)
A2 <- matrix(rnorm(4), 2, 2)

X <- tensorTransform(S, A1, 1)
X <- tensorTransform(X, A2, 2)

tsobi <- tSOBI(X)

MD(tsobi$W[[1]], A1)
MD(tsobi$W[[2]], A2)
tMD(tsobi$W, list(A1, A2))
```
Index

*Topic **array**
  k_tJADE, 3
  mModeAutoCovariance, 5
  mModeCovariance, 6
  tensorBSS-package, 2
  tensorCentering, 10
  tensorStandardize, 11
  tensorTransform, 12
  tensorVectorize, 13
  tFOBI, 14
  tgFOBI, 16
  tgJADE, 18
  tJADE, 20
  tMD, 21
  tNSS JD, 23
  tNSS SD, 25
  tNSS TD JD, 27
  tPCA, 29
  tPP, 31
  tSIR, 32
  tSObI, 34

*Topic **methods**
  plot.tbss, 7

*Topic **multivariate**
  k_tJADE, 3
  tensorBSS-package, 2
  tFOBI, 14
  tgFOBI, 16
  tgJADE, 18
  tJADE, 20
  tMD, 21
  tNSS JD, 23
  tNSS SD, 25
  tNSS TD JD, 27
  tPCA, 29
  tPP, 31
  tSIR, 32
  tSObI, 34

*Topic **package**
  tensorBSS-package, 2

*Topic **ts**
  tensorBSS-package, 2
  tgFOBI, 16
  tgJADE, 18
  tSObI, 34

*Topic **utilities**
  mModeAutoCovariance, 5
  mModeCovariance, 6
  selectComponents, 9
  tensorCentering, 10
  tensorStandardize, 11
  tensorTransform, 12
  tensorVectorize, 13

array, 3, 15, 17, 18, 20, 30, 34
fICA, 32
FOBI, 15
gFOBI, 17
gJADE, 18, 19
JADE, 4, 20, 21
k_JADE, 3, 4
k_tJADE, 3
MD, 22
mModeAutoCovariance, 5, 7
mModeCovariance, 6, 6
NGPP, 32
NSS JD, 24, 26, 29
NSS SD, 24, 26, 29
NSS TD JD, 24, 26, 29
pairs, 8
plot.tbss, 4, 7, 15, 17, 19, 20, 24, 25, 28, 31, 33, 34
plot.ts, 8
INDEX

quantile, 33
rjd, 3, 16–20, 23, 28, 34, 35
selectComponents, 8, 9
SOBI, 34, 35

tensorBSS (tensorBSS-package), 2
tensorBSS-package, 2
tensorCentering, 10
tensorStandardize, 11
tensorTransform, 12
tensorVectorize, 13
tFOBI, 3, 14, 17, 20
tgFOBI, 15, 16
tgJADE, 18, 20, 21
tJADE, 3, 4, 18, 19, 20
tMD, 21
tNSS.JD, 23
tNSS.SD, 25
tNSS.TD.JD, 27
tPCA, 29
tPP, 31
tsSIR, 32
tsSOBI, 8, 34