# Package: tensorBSS (via r-universe)

October 12, 2026

**Type** Package

**Title** Blind Source Separation Methods for Tensor-Valued Observations

**Version** 0.3.9

**Date** 2024-09-12

**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', 'tPP' and 'tTUCKER'.

**License** GPL (>= 2)

**Imports** Rcpp (>= 0.12.3), tensor, tsBSS, ICtest, ggplot2, abind

**LinkingTo** Rcpp, RcppArmadillo

**Depends** JADE, R (>= 2.10), fICA

**Suggests** stochvol

**NeedsCompilation** yes

**Author** Joni Virta [aut, cre] (<https://orcid.org/0000-0001-7330-8434>), Christoph L. Koesner [aut] (<https://orcid.org/0000-0002-2061-8022>), Bing Li [aut], Klaus Nordhausen [aut] (<https://orcid.org/0000-0002-3758-8501>), Hannu Oja [aut], Una Radojicic [aut] (<https://orcid.org/0000-0003-0329-0595>)

**Repository** https://cran.r-universe.dev

**Date/Publication** 2024-09-13 02:56:43 UTC

**RemoteUrl** https://github.com/cran/tensorBSS

**RemoteRef** HEAD

**RemoteSha** 3aec27e7199ccfb6ab0739b5f54317f3e42717f0

# Contents

tensorBSS-package . . . 2
ggtaugplot . . . 4
ggtladleplot . . . 6
k_tJADE . . . 8
mFlatten . . . 10
mModeAutoCovariance . . . 11
mModeCovariance . . . 12
plot.tbss . . . 13
print.taug . . . 15
print.tladle . . . 15
selectComponents . . . 16
tensorBoot . . . 17
tensorCentering . . . 18
tensorStandardize . . . 19
tensorTransform . . . 20
tensorTransform2 . . . 21
tensorVectorize . . . 22
tFOBI . . . 23
tgFOBI . . . 25
tgJADE . . . 27
tJADE . . . 29
tMD . . . 31
tNSS.JD . . . 32
tNSS.SD . . . 34
tNSS.TD.JD . . . 37
tPCA . . . 38
tPCAaug . . . 40
tPCAladle . . . 43
tPP . . . 45
tSIR . . . 47
tSOBI . . . 48
tTUCKER . . . 50
zip.test . . . 51
zip.train . . . 52
zip2image . . . 54

**Index** **55**

---

`tensorBSS-package` *Blind Source Separation Methods for Tensor-Valued Observations*

---

## 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', 'tPP' and 'tTUCKER'.

## Details

| | |
|---|---|
| Package: | tensorBSS |
| Type: | Package |
| Version: | 0.3.9 |
| Date: | 2024-09-12 |
| License: | GPL (>= 2) |

## Author(s)

Joni Virta, Christoph Koesner, Bing Li, Klaus Nordhausen, Hannu Oja and Una Radojicic

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

## References

*Virta, J., Taskinen, S. and Nordhausen, K. (2016), Applying fully tensorial ICA to fMRI data, Signal Processing in Medicine and Biology Symposium (SPMB), 2016 IEEE, doi:10.1109/SPMB.2016.7846858*

*Virta, J., Li, B., Nordhausen, K. and Oja, H., (2017), Independent component analysis for tensor-valued data, Journal of Multivariate Analysis, doi:10.1016/j.jmva.2017.09.008*

*Virta, J. and Nordhausen, K., (2017), Blind source separation of tensor-valued time series. Signal Processing 141, 204-216, doi:10.1016/j.sigpro.2017.06.008*

*Virta J., Nordhausen K. (2017): Blind source separation for nonstationary tensor-valued time series, 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), doi:10.1109/MLSP.2017.8168122*

*Virta J., Li B., Nordhausen K., Oja H. (2018): JADE for tensor-valued observations, Journal of Computational and Graphical Statistics, 27, 628 - 637, doi:10.1080/10618600.2017.1407324*

*Virta J., Lietzen N., Ilmonen P., Nordhausen K. (2021): Fast tensorial JADE, Scandinavian Journal of Statistics, 48, 164-187, doi:10.1111/sjos.12445*

*Radojicic, U., Lietzen, N., Nordhausen, K. and Virta, J. (2021): Dimension estimation in two-dimensional PCA. In S. Loncaric, T. Petkovic and D. Petrinovic (editors) "Proceedings of the 12 International Symposium on Image and Signal Processing and Analysis (ISPA 2021)", 16-22. doi:10.1109/ISPA52656.2021.9552114.*

*Radojicic, U., Lietzen, N., Nordhausen, K. and Virta, J. (2022): Order determination for tensor-valued observations using data augmentation. <arXiv:2207.10423>.*

*Koesner, C, Nordhausen, K. and Virta, J. (2019), Estimating the signal tensor dimension using tensorial PCA. Manuscript.*

`ggtaugplot` *Augmentation plot for each mode of an object of class taug using ggplot2*

## Description

The augmentation plot is a measure for deciding about the number of interesting components. Of interest for the augmentation plot, which is quite similar to the ladle plot, is the minimum. The function offers, however, also the possibility to plot other criterion values that combined make up the actual criterion.

## Usage

```
ggtaugplot(x, crit = "gn", type = "l", scales = "free", position = "horizontal",
  ylab = crit, xlab = "component", main = deparse(substitute(x)),  ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class taug. |
| `crit` | the criterion to be plotted, options are `"gn"`, `"fn"`, `"phin"` and `"lambda"`. |
| `type` | plotting type, either lines `l` or points `p`. |
| `position` | placement of augmentation plots for separate modes, options are `"horizontal"` and `"vertical"`. |
| `scales` | determines whether the x- and y-axis scales are shared or allowed to vary freely across the subplots. The options are: both axes are free (the default, `"free"`), x-axis is free (`"free_x"`), y-axis is free (`"free_y"`), neither is free (`"fixed"`). |
| `ylab` | default ylab value. |
| `xlab` | default xlab value. |
| `main` | default title. |
| `...` | other arguments for the plotting functions. |

## Details

The main criterion of the augmentation criterion is the scaled sum of the eigenvalues and the measure of variation of the eigenvectors up to the component of interest.

The sum is denoted `"gn"` and the individual parts are `"fn"` for the measure of the eigenvector variation and `"phin"` for the scaled eigenvalues. The last option `"lambda"` corresponds to the unscaled eigenvalues yielding then a screeplot.

The plot is drawn separately for each mode of the data.

## Author(s)

Klaus Nordhausen, Joni Virta, Una Radojicic

## References

*Radojicic, U., Lietzen, N., Nordhausen, K. and Virta, J. (2021), On order determinaton in 2D PCA. Manuscript.*

## See Also

tPCAaug

## Examples

```
library(ICtest)


# matrix-variate example
n <- 200
sig <- 0.6

Z <- rbind(sqrt(0.7)*rt(n,df=5)*sqrt(3/5),
           sqrt(0.3)*runif(n,-sqrt(3),sqrt(3)),
           sqrt(0.3)*(rchisq(n,df=3)-3)/sqrt(6),
           sqrt(0.9)*(rexp(n)-1),
           sqrt(0.1)*rlogis(n,0,sqrt(3)/pi),
           sqrt(0.5)*(rbeta(n,2,2)-0.5)*sqrt(20)
)

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

U1 <- rorth(12)[,1:3]
U2 <- rorth(8)[,1:2]
U <- list(U1=U1, U2=U2)
Y <- tensorTransform2(Z,U,1:2)
EPS <- array(rnorm(12*8*n, mean=0, sd=sig), dim=c(12,8,n))
X <- Y + EPS


TEST <- tPCAaug(X)
TEST
ggtaugplot(TEST)

# higher order tensor example

Z2 <- rnorm(n*3*2*4*6)

dim(Z2) <- c(3,2,4,6,n)

U2.1 <- rorth(10)[ ,1:3]
U2.2 <- rorth(8)[ ,1:2]
U2.3 <- rorth(5)[ ,1:4]
U2.4 <- rorth(15)[ ,1:6]

U2 <- list(U1 = U2.1, U2 = U2.2, U3 = U2.3, U4 = U2.4)
Y2 <- tensorTransform2(Z2, U2, 1:4)
```

```
EPS2 <- array(rnorm(10*8*5*15*n, mean=0, sd=sig), dim=c(10, 8, 5, 15, n))
X2 <- Y2 + EPS2


TEST2 <- tPCAaug(X2)
ggtaugplot(TEST2, crit = "lambda", position = "vertical",
 scales = "free_x")
```

---

`ggtladleplot` *Ladle plot for each mode of an object of class tladle using ggplot2*

---

## Description

The ladle plot is a measure for deciding about the number of interesting components. Of interest for the ladle criterion is the minimum. The function here offers however also to plot other criterion values which are part of the actual ladle criterion.

## Usage

```
ggtladleplot(x, crit = "gn", type = "l", scales = "free",
  position = "horizontal", ylab = crit,
  xlab = "component", main = deparse(substitute(x)), ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class ladle. |
| `crit` | the criterion to be plotted, options are `"gn"`, `"fn"`, `"phin"` and `"lambda"`. |
| `type` | plotting type, either lines `l` or points `p`. |
| `position` | placement of augmentation plots for separate modes, options are `"horizontal"` and `"vertical"`. |
| `scales` | determines whether the x- and y-axis scales are shared or allowed to vary freely across the subplots. The options are: both axes are free (the default, `"free"`), x-axis is free (`"free_x"`), y-axis is free (`"free_y"`), neither is free (`"fixed"`). |
| `ylab` | default ylab value. |
| `xlab` | default xlab value. |
| `main` | default title. |
| `...` | other arguments for the plotting functions. |

## Details

The main criterion of the ladle is the scaled sum of the eigenvalues and the measure of variation of the eigenvectors up to the component of interest.

The sum is denoted `"gn"` and the individual parts are `"fn"` for the measure of the eigenvector variation and `"phin"` for the scaled eigenvalues. The last option `"lambda"` corresponds to the unscaled eigenvalues yielding then a screeplot.

The plot is drawn separately for each mode of the data.

## Author(s)

Klaus Nordhausen, Joni Virta

## References

*Koesner, C, Nordhausen, K. and Virta, J. (2019), Estimating the signal tensor dimension using tensorial PCA. Manuscript.*

*Luo, W. and Li, B. (2016), Combining Eigenvalues and Variation of Eigenvectors for Order Determination, Biometrika, 103. 875–887. <doi:10.1093/biomet/asw051>*

## See Also

tPCAladle

## Examples

```
library(ICtest)
n <- 500
sig <- 0.6

Z <- rbind(sqrt(0.7)*rt(n,df=5)*sqrt(3/5),
           sqrt(0.3)*runif(n,-sqrt(3),sqrt(3)),
           sqrt(0.3)*(rchisq(n,df=3)-3)/sqrt(6),
           sqrt(0.9)*(rexp(n)-1),
           sqrt(0.1)*rlogis(n,0,sqrt(3)/pi),
           sqrt(0.5)*(rbeta(n,2,2)-0.5)*sqrt(20)
)

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

U1 <- rorth(12)[,1:3]
U2 <- rorth(8)[,1:2]
U <- list(U1=U1, U2=U2)
Y <- tensorTransform2(Z,U,1:2)
EPS <- array(rnorm(12*8*n, mean=0, sd=sig), dim=c(12,8,n))
X <- Y + EPS


TEST <- tPCAladle(X, n.boot = 100)
TEST
ggtladleplot(TEST, crit = "gn")
ggtladleplot(TEST, crit = "fn")
ggtladleplot(TEST, crit = "phin")
ggtladleplot(TEST, crit = "lambda")
```

k_tJADE *k-tJADE for Tensor-Valued Observations*

## Description

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

## Usage

```
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

*Miettinen, J., Nordhausen, K., Oja, H. and Taskinen, S. (2013), Fast Equivariant JADE, In the Proceedings of* 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013), *6153–6157*, *doi:10.1109/ICASSP.2013.6638847*

*Virta J., Li B., Nordhausen K., Oja H. (2018): JADE for tensor-valued observations, Journal of Computational and Graphical Statistics, 27, 628-637, doi:10.1080/10618600.2017.1407324*

*Virta J., Lietzen N., Ilmonen P., Nordhausen K. (2021): Fast tensorial JADE, Scandinavian Journal of Statistics, 48, 164-187, doi:10.1111/sjos.12445*

## See Also

k_JADE, tJADE, JADE

## 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)


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))
```

mFlatten *Flattening an Array Along One Mode*

## Description

Reshapes a higher order array (tensor) into a matrix with a process known as m-mode flattening or matricization.

## Usage

```
mFlatten(x, m)
```

## Arguments

| | |
|---|---|
| x | an $(r+1)$-dimensional array with $r \geq 2$. The final mode is understood to correspond to the observations (i.e., its length is usually the sample size n). |
| m | an integer between 1 and $r$ signifying the mode along which the array should be flattened. Note that the flattening cannot be done w.r.t. the final $(r+1)$th mode. |

## Details

If the original tensor x has the size $p_1 \times \cdots \times p_r \times n$, then `mFlatten(x, m)` returns tensor of size $p_m \times p_1 \cdots p_{m-1} p_{m+1} \cdots p_r \times n$ obtained by gathering all $m$-mode vectors of x into a wide matrix (an $m$-mode vector of x is any vector of length $p_m$ obtained by varying the $m$th index and holding the other indices constant).

## Value

The $m$-mode flattened 3rd order tensor of size $p_m \times p_1 \cdots p_{m-1} p_{m+1} \cdots p_r \times n$.

## Author(s)

Joni Virta

## Examples

```
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)

dim(mFlatten(x, 1))
dim(mFlatten(x, 2))
```

`mModeAutoCovariance` *The m-Mode Autocovariance Matrix*

## 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, normalize = 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`. |
| `normalize` | Logical, indicating whether the resulting matrix is divided by `p_1 ... p_{m-1} p_{m+1} ... p_r` or not. 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 $Cov_{m\tau}(X_t) = E(X_t^{(m)} X_{t+\tau}^{(m)T})/(p_1 \ldots p_{m-1} p_{m+1} \ldots p_r)$, 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 `lag` having the size $p_m \times p_m$.

## Author(s)

Joni Virta

## References

*Virta, J. and Nordhausen, K., (2017), Blind source separation of tensor-valued time series, Signal Processing, 141, 204-216, doi:10.1016/j.sigpro.2017.06.008*

## 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, normalize = 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. |
| normalize | Logical, indicating whether the resulting matrix is divided by p_1 ... p_{m-1} p_{m+1} ... p_r or not. 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 $Cov_m(X) = E(X^{(m)}X^{(m)T})/(p_1 \ldots p_{m-1}p_{m+1} \ldots p_r)$, 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 $p_m \times p_m$.

## Author(s)

Joni Virta

## References

*Virta, J., Li, B., Nordhausen, K. and Oja, H., (2017), Independent component analysis for tensor-valued data, Journal of Multivariate Analysis, doi:10.1016/j.jmva.2017.09.008*

## See Also

mModeAutoCovariance

## Examples

```
## 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

```
## 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

```
data(zip.train)
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)

if(require("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)

  tgfobi <- tgFOBI(X)
  plot(tgfobi, 1, 1)
}
```

## print.taug *Printing an object of class taug*

### Description

Prints an object of class taug.

### Usage

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

### Arguments

| | |
|---|---|
| x | object of class taug. |
| ... | further arguments to be passed to or from methods. |

### Author(s)

Klaus Nordhausen

## print.tladle *Printing an object of class tladle*

### Description

Prints an object of class tladle.

### Usage

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

### Arguments

| | |
|---|---|
| x | object of class tladle. |
| ... | further arguments to be passed to or from methods. |

### Author(s)

Klaus Nordhausen

`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 correponding 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

```
data(zip.train)
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)
```

---

`tensorBoot` *Bootstrapping or Permuting a Data Tensor*

---

## Description

The function takes bootstrap samples or permutes its content along the last dimension of the tensor.

## Usage

```
tensorBoot(x, replace = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | Array of an order of at least two with the last dimension corresponding to the sampling units. |
| `replace` | Logical. Should sampling be performed with or without replacement. |

## Details

Assume an array of dimension $r + 1$, where the last dimension represents the $n$ sampling units and the first $r$ dimensions the data per unit. The function then returns an array of the same dimension as `x` where either $n$ bootstraps samples are selected or the units are permuted.

## Value

The bootstrapped or permuted samples in an array with the same dimension as `x`.

## Author(s)

Christoph Koesner

## Examples

```
x <- array(1:50, c(2, 5, 5))
x
tensorBoot(x)
tensorBoot(x, replace = FALSE)

x <- array(1:100, c(2, 5, 2, 5))
x
tensorBoot(x)
```

`tensorCentering` *Center an Array of Observations*

## Description

Centers an array of array-valued observations by substracting a location array (the mean array by default) from each observation.

## Usage

```
tensorCentering(x, location = NULL)
```

## Arguments

| | |
|---|---|
| `x` | Array of order at least two with the last dimension corresponding to the sampling units. |
| `location` | The location to be used in the centering. Either `NULL`, defaulting to the mean array, or a user-specified $p_1 \times p_2 \times \ldots \times p_r$-dimensional array. |

## Details

Centers a $p_1 \times p_2 \times \ldots \times p_r \times n$-dimensional array by substracting the $p_1 \times p_2 \times \ldots \times p_r$-dimensional `location` from each of the observed arrays.

## Value

Array of centered observations with the same dimensions as the input array. The used location is returned as attribute `"location"`.

## Author(s)

Joni Virta

## Examples

```
## 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, location = NULL, scatter = NULL)
```

## Arguments

| | |
|---|---|
| `x` | Array of an order higher than two with the last dimension corresponding to the sampling units. |
| `location` | The location to be used in the standardizing. Either `NULL`, defaulting to the mean array, or a user-specified $p_1 \times p_2 \times \ldots \times p_r$-dimensional array. |
| `scatter` | The scatter matrices to be used in the standardizing. Either `NULL`, defaulting to the m-mode covariance matrices, or a user-specified list of length `r` of $p_1 \times p_1, \ldots, p_r \times p_r$-dimensional symmetric positive definite matrices. |

## Details

The algorithm first centers the $n$ observed tensors $X_i$ using `location` (either the sample mean, or a user-specified location). Then, if `scatter` = `NULL`, it estimates the $m$th mode covariance matrix $Cov_m(X) = E(X^{(m)}X^{(m)T})/(p_1 \ldots p_{m-1}p_{m+1} \ldots p_r)$, where $X^{(m)}$ is the centered $m$-flattening of $X$, for each mode, and transforms the observations with the inverse square roots of the covariance matrices from the corresponding modes. If, instead, the user has specified a non-`NULL` value for `scatter`, the inverse square roots of those matrices are used to transform the centered data.

## Value

A list containing the following components:

| | |
|---|---|
| `x` | Array of the same size as `x` containing the standardized observations. The used location and scatters are returned as attributes `"location"` and `"scatter"`. |
| `S` | List containing inverse square roots of the covariance matrices of different modes. |

## Author(s)

Joni Virta

## Examples

```
# 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 $m$th 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

```
# 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)
```

---

`tensorTransform2` *Linear Transformations of Tensors from Several Modes*

---

## Description

Applies a linear transformation to user selected modes of each individual tensor in an array of tensors. The function is a generalization of `tensorTransform` which only transforms one specific mode.

## Usage

```
tensorTransform2(x, A, mode, transpose = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Array of order r+1 >= 2 where the last dimension corresponds to the sampling units. |
| `A` | A list of r matrices to apply linearly to the corresponding mode. |
| `mode` | subsetting vector indicating which modes should be linearly transformed by multiplying them with the corresponding matrices from `A`. |
| `transpose` | logical. Should the matrices in `A` be transposed before the mode wise transformations or not. |

## Details

For the modes $i_1, \ldots, i_k$, specified via mode, the function applies the linear transformation given by the matrix $A^{i_j}$ of size $q_{i_j} \times p_{i_j}$ to the $i_j$th mode of each of the $n$ observed tensors $X_{i_j}$ in the given $p_1 \times p_2 \times \ldots \times p_r \times n$-dimensional array x.

## Value

Array with r+1 dimensions where the dimensions specfied via mode are transformed.

## Author(s)

Klaus Nordhausen

## See Also

tensorTransform

## Examples

```
n <- 5
x <- array(rnorm(5*6*7), dim = c(7, 6, 5))
A1 <- matrix(runif(14), ncol = 7)
A2 <- matrix(rexp(18), ncol = 6)
A  <- list(A1 = A1, A2 = A2)
At <- list(tA1 = t(A1), tA2 = t(A2))

x1 <- tensorTransform2(x, A, 1)
x2 <- tensorTransform2(x, A, -2)
x3 <- tensorTransform(x, A1, 1)
x1 == x2
x1 == x3
x4 <- tensorTransform2(x,At,-2, TRUE)
x1 == x4
x5 <- tensorTransform2(x, A, 1:2)
```

---

# tensorVectorize *Vectorize an Observation Tensor*

---

## 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.

Note that the output is a matrix of the size "number of variables" x "number of observations", that is, a transpose of the standard format for a data matrix.

### Value

Matrix whose columns contain the vectorized observed tensors.

### Author(s)

Joni Virta

### Examples

```
# 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)

# The covariance matrix of individual tensor elements
cov(t(vecx))
```

---

## tFOBI *FOBI for Tensor-Valued Observations*

---

### 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

*Virta, J., Li, B., Nordhausen, K. and Oja, H., (2017), Independent component analysis for tensor-valued data, Journal of Multivariate Analysis, doi:10.1016/j.jmva.2017.09.008*

## See Also

FOBI, tgFOBI

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

# Digit data example

data(zip.train)
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)
```

---

# tgFOBI *gFOBI for Tensor-Valued Time Series*

---

## Description

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

## Usage

```
tgFOBI(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 rjd. |
| eps | Convergence tolerance. Passed on to rjd. |

## Details

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ 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

*Virta, J. and Nordhausen, K., (2017), Blind source separation of tensor-valued time series. Signal Processing 141, 204-216, doi:10.1016/j.sigpro.2017.06.008*

## See Also

gFOBI, rjd, tFOBI

## Examples

```
if(require("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)

  tgfobi <- tgFOBI(X)

  MD(tgfobi$W[[1]], A1)
  MD(tgfobi$W[[2]], A2)
  tMD(tgfobi$W, list(A1, A2))
}
```

---

# 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 rjd. |
| eps | Convergence tolerance. Passed on to rjd. |

## Details

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ 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 $p_1, p_2, \ldots, p_r, T$.

`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 `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 `gJADE`.

If `lags = 0` the method reduces to `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

*Virta, J. and Nordhausen, K., (2017), Blind source separation of tensor-valued time series. Signal Processing 141, 204-216, doi:10.1016/j.sigpro.2017.06.008*

## See Also

`gJADE`, `rjd`, `tJADE`

## Examples

```
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))
```

---

tJADE *tJADE for Tensor-Valued Observations*

---

## 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

*Virta J., Li B., Nordhausen K., Oja H. (2018): JADE for tensor-valued observations, Journal of Computational and Graphical Statistics, Volume 27, p. 628 - 637, doi:10.1080/10618600.2017.1407324*

## See Also

JADE, tgJADE

## 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)

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

data(zip.train)
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)
```

---

tMD *Minimum Distance Index of a Kronecker Product*

---

## 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]] %x% ... %x% W.hat[[1]]` and `A[[r]] %x% ... %x% 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

*Ilmonen, P., Nordhausen, K., Oja, H. and Ollila, E. (2010), A New Performance Index for ICA: Properties, Computation and Asymptotic Analysis. In Vigneron, V., Zarzoso, V., Moreau, E., Gribonval, R. and Vincent, E. (editors) Latent Variable Analysis and Signal Separation, 229-236, Springer.*

*Virta, J., Li, B., Nordhausen, K. and Oja, H., (2017), Independent component analysis for tensor-valued data, Journal of Multivariate Analysis, doi:10.1016/j.jmva.2017.09.008*

## 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

```
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 TNSS-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

*Virta J., Nordhausen K. (2017): Blind source separation for nonstationary tensor-valued time series, 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), doi:10.1109/MLSP.2017.8168122*

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

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

*Virta J., Nordhausen K. (2017): Blind source separation for nonstationary tensor-valued time series, 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), doi:10.1109/MLSP.2017.8168122*

## See Also

NSS.SD, NSS.JD, NSS.TD.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*t*(t - n - 1))))
}

var_4 <- function(n){
  t <- 1:n
  return(3.443 - 8*exp((n+1)^2/(4*t*(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)*(1 - t/(n + 1))^(3 - 1))
}

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 = c(-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
```

tNSS.TD.JD *TNSS-TD-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 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 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

*Virta J., Nordhausen K. (2017): Blind source separation for nonstationary tensor-valued time series, 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), doi:10.1109/MLSP.2017.8168122*

## See Also

NSS.SD, NSS.JD, NSS.TD.JD, tNSS.SD, 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)
res$W

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

---

# 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 $d_1, d_2, ...d_r$, 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

*Virta, J., Taskinen, S. and Nordhausen, K. (2016), Applying fully tensorial ICA to fMRI data, Signal Processing in Medicine and Biology Symposium (SPMB), 2016 IEEE,*
*doi:10.1109/SPMB.2016.7846858*

## Examples

```
# Digit data example

data(zip.train)
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)


tpca <- tPCA(x0, d = c(2, 2))
pairs(t(apply(tpca$S, 3, c)), col=y0)
```

---

tPCAaug *Order Determination for Tensorial PCA Using Augmentation*

---

## Description

In a tensorial PCA context the dimensions of a core tensor are estimated based on augmentation of additional noise components. Information from both eigenvectors and eigenvalues are then used to obtain the dimension estimates.

## Usage

```
tPCAaug(x, noise = "median", naug = 1, nrep = 1,
  sigma2 = NULL, alpha = NULL)
```

## Arguments

| | |
|---|---|
| x | array of an order at least three with the last dimension corresponding to the sampling units. |
| noise | specifies how to estimate the noise variance. Can be one of "median", "quantile", "last", "known". Default is "median". See details for further information. |
| naug | number of augmented variables in each mode. Default is 1. |
| nrep | number of repetitions for the augmentation. Default is 1. |
| sigma2 | if noise = "known" the value of the noise variance. |
| alpha | if noise = "quantile" this specifies the quantile to be used. |

## Details

For simplicity details are given for matrix valued observations.

Assume having a sample of $p_1 \times p_2$ matrix-valued observations which are realizations of the model $X = U_L Z U_R' + N$, where $U_L$ and $U_R$ are matrices with orthonormal columns, $Z$ is the random, zero mean $k_1 \times k_2$ core matrix with $k_1 \leq p_1$ and $k_2 \leq p_2$. $N$ is $p_1 \times p_2$ matrix-variate noise that follows a matrix variate spherical distribution with $E(N) = 0$ and $E(NN') = \sigma^2 I_{p_1}$ and is independent from $Z$. The goal is to estimate $k_1$ and $k_2$. For that purpose the eigenvalues and eigenvectors of the left and right covariances are used. To evaluate the variation in the eigenvectors, in each mode the matrix $X$ is augmented with naug normally distributed components appropriately

scaled by noise standard deviation. The procedure can be repeated nrep times to reduce random variation in the estimates.

The procedure needs an estimate of the noise variance and four options are available via the argument noise:

1. noise = "median": Assumes that at least half of components are noise and uses thus the median of the pooled and scaled eigenvalues as an estimate.
2. noise = "quantile": Assumes that at least 100 alpha % of the components are noise and uses the mean of the lower alpha quantile of the pooled and scaled eigenvalues from all modes as an estimate.
3. noise = "last": Uses the pooled information from all modes and then the smallest eigenvalue as estimate.
4. noise = "known": Assumes the error variance is known and needs to be provided via sigma2.

## Value

A list of class 'taug' inheriting from class 'tladle' and containing:

| | |
|---|---|
| U | list containing the modewise rotation matrices. |
| D | list containing the modewise eigenvalues. |
| S | array of the same size as x containing the principal components. |
| ResMode | a list with the modewise results which are lists containing:<br>**mode** label for the mode.<br>**k** the order estimated for that mode.<br>**fn** vector giving the measures of variation of the eigenvectors.<br>**phin** normalized eigenvalues.<br>**lambda** the unnormalized eigenvalues used to compute phin.<br>**gn** the main criterion augmented order estimator.<br>**comp** vector from 0 to the number of dimensions to be evaluated. |
| xmu | the data location |
| data.name | string with the name of the input data |
| method | string tPCA. |
| Sigma2 | estimate of standardized sigma2 from the model described above or the standardized provided value. Sigma2 is the estimate for the variance of individual entries of N. |
| AllSigHat2 | vector of noise variances used for each mode. |

## Author(s)

Klaus Nordhausen, Una Radojicic

## References

*Radojicic, U., Lietzen, N., Nordhausen, K. and Virta, J. (2021): Dimension Estimation in Two-Dimensional PCA. In S. Loncaric, T. Petkovic and D. Petrinovic (editors) "Proceedings of the 12 International Symposium on Image and Signal Processing and Analysis (ISPA 2021)", 16-22. doi:10.1109/ISPA52656.2021.9552114.*

*Radojicic, U., Lietzen, N., Nordhausen, K. and Virta, J. (2022): Order Determination for Tensor-valued Observations Using Data Augmentation. <arXiv:2207.10423>.*

## See Also

tPCA, tPCAladle

## Examples

```
library(ICtest)


# matrix-variate example
n <- 200
sig <- 0.6

Z <- rbind(sqrt(0.7)*rt(n,df=5)*sqrt(3/5),
           sqrt(0.3)*runif(n,-sqrt(3),sqrt(3)),
           sqrt(0.3)*(rchisq(n,df=3)-3)/sqrt(6),
           sqrt(0.9)*(rexp(n)-1),
           sqrt(0.1)*rlogis(n,0,sqrt(3)/pi),
           sqrt(0.5)*(rbeta(n,2,2)-0.5)*sqrt(20)
)

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

U1 <- rorth(12)[,1:3]
U2 <- rorth(8)[,1:2]
U <- list(U1=U1, U2=U2)
Y <- tensorTransform2(Z,U,1:2)
EPS <- array(rnorm(12*8*n, mean=0, sd=sig), dim=c(12,8,n))
X <- Y + EPS


TEST <- tPCAaug(X)
# Dimension should be 3 and 2 and (close to) sigma2 0.36
TEST

# Noise variance in i-th mode is equal to Sigma2 multiplied by the product
# of number of colums of all modes except i-th one

TEST$Sigma2*c(8,12)
# This is returned as
TEST$AllSigHat2

# higher order tensor example
```

```
Z2 <- rnorm(n*3*2*4*10)

dim(Z2) <- c(3,2,4,10,n)

U2.1 <- rorth(12)[ ,1:3]
U2.2 <- rorth(8)[ ,1:2]
U2.3 <- rorth(5)[ ,1:4]
U2.4 <- rorth(20)[ ,1:10]

U2 <- list(U1 = U2.1, U2 = U2.2, U3 = U2.3, U4 = U2.4)
Y2 <- tensorTransform2(Z2, U2, 1:4)
EPS2 <- array(rnorm(12*8*5*20*n, mean=0, sd=sig), dim=c(12, 8, 5, 20, n))
X2 <- Y2 + EPS2


TEST2 <- tPCAaug(X2, noise = "quantile", alpha =0.3)
TEST2
```

---

`tPCAladle` *Ladle Estimate for tPCA*

---

## Description

For r-dimensional tensors, the Ladle estimate for tPCA assumes that for a given mode $m$, the last $p_m - k_m$ modewise eigenvalues are equal. Combining information from the eigenvalues and eigenvectors of the m-mode covariance matrix the ladle estimator yields estimates for $k_1, ..., k_r$.

## Usage

```
tPCAladle(x, n.boot = 200, ncomp = NULL)
```

## Arguments

| | |
|---|---|
| `x` | array of an order at least two with the last dimension corresponding to the sampling units. |
| `n.boot` | number of bootstrapping samples to be used. |
| `ncomp` | vector giving the number of components among which the ladle estimator is to be searched for each mode. The default follows the recommendation of Luo and Li 2016. |

## Details

The model here assumes that the eigenvalues of the m-mode covariance matrix are of the form $\lambda_{1,m} \geq ... \geq \lambda_{k_m,m} > \lambda_{k_m+1,m} = ... = \lambda_{p_m,m}$ and the goal is to estimate the value of $k_m$ for all modes. The ladle estimate for this purpose combines the values of the scaled eigenvalues and the variation of the eigenvectors based on bootstrapping. The idea there is that for distinct eigenvales

the variation of the eigenvectors is small and for equal eigenvalues the corresponding eigenvectors have large variation.

This measure is then computed assuming $k_m$=0,..., `ncomp[m]` and the ladle estimate for $k_m$ is the value where the measure takes its minimum.

## Value

A list of class 'tladle' containing:

`U` list containing the modewise rotation matrices.

`D` list containing the modewise eigenvalues.

`S` array of the same size as `x` containing the principal components.

`ResMode` a list with the modewise results which are lists containing:

**mode** label for the mode.

**k** the estimated value of k.

**fn** vector giving the measures of variation of the eigenvectors using the bootstrapped eigenvectors for the different number of components.

**phin** normalized eigenvalues.

**lambda** the unnormalized eigenvalues used to compute phin.

**gn** the main criterion for the ladle estimate - the sum of fn and phin. k is the value where gn takes its minimum.

**comp** vector from 0 to the number of dimensions to be evaluated.

`xmu` the data location

`data.name` string with the name of the input data

`method` string `tPCA`.

## Author(s)

Klaus Nordhausen

## References

*Koesner, C, Nordhausen, K. and Virta, J. (2019), Estimating the signal tensor dimension using tensorial PCA. Manuscript.*

*Luo, W. and Li, B. (2016), Combining Eigenvalues and Variation of Eigenvectors for Order Determination, Biometrika, 103, 875–887. <doi:10.1093/biomet/asw051>*

## See Also

`tPCA`, `ggtladleplot`

## Examples

```
library(ICtest)
n <- 200
sig <- 0.6

Z <- rbind(sqrt(0.7)*rt(n,df=5)*sqrt(3/5),
           sqrt(0.3)*runif(n,-sqrt(3),sqrt(3)),
           sqrt(0.3)*(rchisq(n,df=3)-3)/sqrt(6),
           sqrt(0.9)*(rexp(n)-1),
           sqrt(0.1)*rlogis(n,0,sqrt(3)/pi),
           sqrt(0.5)*(rbeta(n,2,2)-0.5)*sqrt(20)
)

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

U1 <- rorth(12)[,1:3]
U2 <- rorth(8)[,1:2]
U <- list(U1=U1, U2=U2)
Y <- tensorTransform2(Z,U,1:2)
EPS <- array(rnorm(12*8*n, mean=0, sd=sig), dim=c(12,8,n))
X <- Y + EPS


TEST <- tPCAladle(X)
TEST
ggtladleplot(TEST)
```

---

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

```
tPP(x, nl = "pow3", eps = 1e-6, maxiter = 100)
```

## Arguments

| | |
|---|---|
| x | Numeric array of an order at least three. It is assumed that the last dimension corresponds to the sampling units. |
| nl | The chosen measure of interestingness/objective function. Current choices include pow3 (default) and skew, see the details below |
| eps | The convergence tolerance of the iterative algortihm. |
| maxiter | The maximum number of iterations. |

## 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 $E[G(u_k^T X X^T u_k)] - E[G(c^2)]$, where $G$ is the chosen objective function and $c^2$ obeys the chi-squared distribution with $q$ degress of freedom. Currently the function allows the choices $G(x) = x^2$ (`pow3`) and $G(x) = x\sqrt{x}$ (`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:

| | |
|---|---|
| `S` | Array of the same size as x containing the estimated components. |
| `W` | List containing all the unmixing matrices. |
| `iter` | The numbers of iteration used per mode. |
| `Xmu` | The data location. |
| `datatype` | Character string with value "iid". Relevant for `plot.tbss`. |

## Author(s)

Joni Virta

## References

*Nordhausen, K. and Virta, J. (2018), Tensorial projection pursuit, Manuscript in preparation.*

*Hyvarinen, A. (1999) Fast and robust fixed-point algorithms for independent component analysis, IEEE transactions on Neural Networks 10.3: 626-634.*

## 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

```
data(zip.train)
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)
```

---

`tSOBI` *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

```
tSOBI(x, lags = 1:12, 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 `rjd`. |
| `eps` | Convergence tolerance. Passed on to `rjd`. |

## Details

It is assumed that $S$ is a tensor (array) of size $p_1 \times p_2 \times \ldots \times p_r$ 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 `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 `lags`.

If `x` is a matrix, that is, $r = 1$, the method reduces to SOBI and the function calls `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 |
| Xmu | The data location. |
| datatype | Character string with value "ts". Relevant for plot.tbss. |

## Author(s)

Joni Virta

## References

*Virta, J. and Nordhausen, K., (2017), Blind source separation of tensor-valued time series. Signal Processing 141, 204-216, doi:10.1016/j.sigpro.2017.06.008*

## See Also

SOBI, rjd

## 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)

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))
```

tTUCKER *Tucker (2) Transformation for a Tensor*

## Description

This is a Tucker (2) transformation of a data tensor where the sampling dimension is uncompressed. The transfromation is known also under many different names like multilinear principal components analysis or generalized low rank approximation of matrices if the tensorial data is matrixvalued.

## Usage

```
tTUCKER(x, ranks, maxiter = 1000, eps = 1e-06)
```

## Arguments

| | |
|---|---|
| x | array with $r + 1$ dimensions where the last dimension corresponds to the sampling units. |
| ranks | vector of length r giving the dimensions of the compressed core tensor. |
| maxiter | maximum number of iterations for the algorithm. |
| eps | convergence tolerance. |

## Details

As initial solution tPCA is used and iterated using an alternating least squares (ALS) approach, known also as higher order orthogonal iteration (HOOI).

## Value

A list containing the following components:

| | |
|---|---|
| S | array of the compressed tensor. |
| U | list containing the rotation matrices. |
| Xmu | the data location. |
| norm2xc | squared norm of the original data tensor after centering. |
| norm2rxc | squared norm of the reconstructed (centered) data tensor. |
| norm2ratio | the ratio norm2rxc/norm2xc. |
| mEV | list containing the eigenvalues from the m-mode covariance matrix when all but the relevant mode have be compressed. |
| tPCA | The output from tPCA which was used as initial value. |

## Author(s)

Klaus Nordhausen

## References

*Lu, H., Plataniotis, K. and Venetsanopoulos, A. (2008), MPCA: Multilinear principal component analysis of tensor objects, IEEE Transactions on Neural Networks, 19, 18-39. doi:10.1109/TNN.2007.901277*

*Lietzen, N., Nordhausen, K. and Virta, J. (2019), Statistical analysis of second-order tensor decompositions, manuscript.*

## See Also

tPCA

## Examples

```
data(zip.train)
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)

tucker <-  tTUCKER(x0, ranks = c(2, 2), eps=1e-03)
pairs(t(apply(tucker$S, 3, c)), col=y0)

# To approximate the original data one uses then
x0r <- tensorTransform2(tucker$S, tucker$U)
```

---

# zip.test *Handwritten Digit Recognition Data*

---

## Description

This .RD-file and the corresponding data set are originally from the R-package ElemStatLearn which has now been removed from CRAN.

This example is a character recognition task: classification of handwritten numerals. This problem captured the attention of the machine learning and neural network community for many years, and has remained a benchmark problem in the field.

## Usage

```
data(zip.test)
```

## Format

The format is: num [1:2007, 1:257] 9 6 3 6 6 0 0 0 6 9 ...

## Details

Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16 x 16 grayscale images (Le Cun et al., 1990).

The data are in two gzipped files, and each line consists of the digit id (0-9) followed by the 256 grayscale values.

There are 7291 training observations and 2007 test observations, distributed as follows: 0 1 2 3 4 5 6 7 8 9 Total Train 1194 1005 731 658 652 556 664 645 542 644 7291 Test 359 264 198 166 200 160 170 147 166 177 2007

or as proportions: 0 1 2 3 4 5 6 7 8 9 Train 0.16 0.14 0.1 0.09 0.09 0.08 0.09 0.09 0.07 0.09 Test 0.18 0.13 0.1 0.08 0.10 0.08 0.08 0.07 0.08 0.09

The test set is notoriously "difficult", and a 2.5 excellent. These data were kindly made available by the neural network group at AT&T research labs (thanks to Yann Le Cunn).

## References

*Kjetil B Halvorsen (package maintainer) (2019), R-package ElemStatLearn: Data Sets, Functions and Examples from the Book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman*

---

# zip.train *Handwritten Digit Recognition Data*

---

## Description

This .RD-file and the corresponding data set are originally from the R-package ElemStatLearn which has now been removed from CRAN.

This example is a character recognition task: classification of handwritten numerals. This problem captured the attention of the machine learning and neural network community for many years, and has remained a benchmark problem in the field.

## Usage

```
data(zip.train)
```

## Format

The format is: num [1:7291, 1:257] 6 5 4 7 3 6 3 1 0 1 ...

## Details

Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16 x 16 grayscale images (Le Cun et al., 1990).

The data are in two gzipped files, and each line consists of the digit id (0-9) followed by the 256 grayscale values.

There are 7291 training observations and 2007 test observations, distributed as follows: 0 1 2 3 4 5 6 7 8 9 Total Train 1194 1005 731 658 652 556 664 645 542 644 7291 Test 359 264 198 166 200 160 170 147 166 177 2007

or as proportions: 0 1 2 3 4 5 6 7 8 9 Train 0.16 0.14 0.1 0.09 0.09 0.08 0.09 0.09 0.07 0.09 Test 0.18 0.13 0.1 0.08 0.10 0.08 0.08 0.07 0.08 0.09

The test set is notoriously "difficult", and a 2.5 excellent. These data were kindly made available by the neural network group at AT&T research labs (thanks to Yann Le Cunn).

## References

*Kjetil B Halvorsen (package maintainer) (2019), R-package ElemStatLearn: Data Sets, Functions and Examples from the Book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman*

## Examples

```
data(zip.train
)
findRows <- function(zip, n) {
 # Find  n (random) rows with zip representing 0,1,2,...,9
 res <- vector(length=10, mode="list")
 names(res) <- 0:9
 ind <- zip[,1]
 for (j in 0:9) {
    res[[j+1]] <- sample( which(ind==j), n ) }
 return(res) }

# Making a plot like that on page 4:

digits <- vector(length=10, mode="list")
names(digits) <- 0:9
rows <- findRows(zip.train, 6)
for (j in 0:9) {
    digits[[j+1]] <- do.call("cbind", lapply(as.list(rows[[j+1]]),
                        function(x) zip2image(zip.train, x)) )
}
im <- do.call("rbind", digits)
image(im, col=gray(256:0/256), zlim=c(0,1), xlab="", ylab="" )
```

`zip2image` *function to convert row of zip file to format used by image()*

## Description

This .RD-file and the corresponding function are originally from the R-package ElemStatLearn which has now been removed from CRAN.

This is a utility function converting zip.train/zip.test data to format useful for plotting with the function `image`.

## Usage

```
zip2image(zip, line)
```

## Arguments

| | |
|---|---|
| `zip` | `zip.train` or `zip.test`. |
| `line` | row of matrix to take |

## Value

16 x 16 matrix suitable as argument for `image`.

## Author(s)

Kjetil Halvorsen

## References

*Kjetil B Halvorsen (package maintainer) (2019), R-package ElemStatLearn: Data Sets, Functions and Examples from the Book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman*

## Examples

```
## See example section of help file for zip.train
```

# Index


∗ **array**
k_tJADE, 8
mFlatten, 10
mModeAutoCovariance, 11
mModeCovariance, 12
print.taug, 15
print.tladle, 15
tensorBoot, 17
tensorBSS-package, 2
tensorCentering, 18
tensorStandardize, 19
tensorTransform, 20
tensorTransform2, 21
tensorVectorize, 22
tFOBI, 23
tgFOBI, 25
tgJADE, 27
tJADE, 29
tMD, 31
tNSS.JD, 32
tNSS.SD, 34
tNSS.TD.JD, 37
tPCA, 38
tPCAaug, 40
tPCAladle, 43
tPP, 45
tSIR, 47
tSOBI, 48
tTUCKER, 50
∗ **datasets**
zip.test, 51
zip.train, 52
∗ **dplot**
zip2image, 54
∗ **hplot**
ggtaugplot, 4
ggtladleplot, 6
∗ **methods**
plot.tbss, 13
∗ **multivariate**
k_tJADE, 8
print.taug, 15
print.tladle, 15
tensorBSS-package, 2
tFOBI, 23
tgFOBI, 25
tgJADE, 27
tJADE, 29
tMD, 31
tNSS.JD, 32
tNSS.SD, 34
tNSS.TD.JD, 37
tPCA, 38
tPCAaug, 40
tPCAladle, 43
tPP, 45
tSIR, 47
tSOBI, 48
tTUCKER, 50
∗ **package**
tensorBSS-package, 2
∗ **ts**
tensorBSS-package, 2
tgFOBI, 25
tgJADE, 27
tSOBI, 48
∗ **utilities**
mFlatten, 10
mModeAutoCovariance, 11
mModeCovariance, 12
selectComponents, 16
tensorBoot, 17
tensorCentering, 18
tensorStandardize, 19
tensorTransform, 20
tensorTransform2, 21
tensorVectorize, 22

array, *8*, *24*, *26*, *28*, *29*, *39*, *48*

fICA, *46*
FOBI, *24*

gFOBI, *26*
ggtaugplot, 4
ggtladleplot, 6, *44*
gJADE, *28*

image, *54*

JADE, *9*, *29, 30*

k_JADE, *8, 9*
k_tJADE, 8

MD, *31*
mFlatten, 10
mModeAutoCovariance, 11, *13*
mModeCovariance, *11*, 12

NGPP, *46*
NSS.JD, *33*, *35*, *38*
NSS.SD, *33*, *35*, *38*
NSS.TD.JD, *33*, *35*, *38*

pairs, *14*
plot.tbss, *8*, 13, *24*, *26*, *28, 29*, *33, 34*, *37*, *46, 47*, *49*
plot.ts, *14*
print.taug, 15
print.tladle, 15

quantile, *47*

rjd, *8*, *26–29*, *32*, *37*, *48, 49*

selectComponents, *13, 14*, 16
SOBI, *48, 49*

tensorBoot, 17
tensorBSS *(*tensorBSS-package*)*, 2
tensorBSS-package, 2
tensorCentering, 18
tensorStandardize, 19
tensorTransform, 20, *21, 22*
tensorTransform2, 21
tensorVectorize, 22
tFOBI, *8*, 23, *26*, *29*
tgFOBI, *24*, 25
tgJADE, 27, *29, 30*
tJADE, *8, 9*, *28*, 29
tMD, 31
tNSS.JD, 32
tNSS.SD, 34
tNSS.TD.JD, 37
tPCA, 38, *42*, *44*, *50, 51*
tPCAaug, *5*, 40
tPCAladle, *7*, *42*, 43
tPP, 45
tSIR, 47
tSOBI, *14*, 48
tTUCKER, 50

zip.test, 51, *54*
zip.train, 52, *54*
zip2image, 54