# Package ‘JADE’

March 2, 2018

**Type** Package

**Title** Blind Source Separation Methods Based on Joint Diagonalization and Some BSS Performance Criteria

**Version** 2.0-1

**Date** 2018-03-02

**Author** Klaus Nordhausen, Jean-Francois Cardoso, Jari Miettinen, Hannu Oja, Esa Ollila, Sara Taskinen

**Maintainer** Klaus Nordhausen <klaus.nordhausen@tuwien.ac.at>

**Imports** clue, graphics

**Suggests** ICS, ICSNP

**Description** Cardoso's JADE algorithm as well as his functions for joint diagonalization are ported to R. Also several other blind source separation (BSS) methods, like AMUSE and SOBI, and some criteria for performance evaluation of BSS algorithms, are given. The package is described in Miettinen, Nordhausen and Taskinen (2017) <doi:10.18637/jss.v076.i02>.

**License** GPL (>= 2)

**LazyData** true

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2018-03-02 15:49:31 UTC

---

**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>JADE-package</td>
<td>2</td>
</tr>
<tr>
<td>amari.error</td>
<td>3</td>
</tr>
<tr>
<td>AMUSE</td>
<td>4</td>
</tr>
<tr>
<td>bss.components</td>
<td>6</td>
</tr>
<tr>
<td>cjd</td>
<td>7</td>
</tr>
<tr>
<td>coef.bss</td>
<td>8</td>
</tr>
<tr>
<td>ComonGAP</td>
<td>9</td>
</tr>
<tr>
<td>CPPdata</td>
<td>10</td>
</tr>
</tbody>
</table>
Description

Cardoso’s JADE algorithm as well as his functions for joint diagonalization are ported to R. Also several other blind source separation (BSS) methods, like AMUSE and SOBI, and some criteria for performance evaluation of BSS algorithms, are given. The package is described in Miettinen, Nordhausen and Taskinen (2017) doi: 10.18637/jss.v076.i02.

Details

Package: JADE
Type: Package
Version: 2.0-1
Date: 2018-03-02
License: GPL (>= 2)

Author(s)

Klaus Nordhausen, Jean-Francois Cardoso, Jari Miettinen, Hannu Oja, Esa Ollila, Sara Taskinen
Maintainer: Klaus Nordhausen <klaus.nordhausen@tuwien.ac.at>
References


Description

Computes the Amari Error to evaluate the performance of an ICA algorithm.

Usage

amari.error(W.hat, A, standardize = FALSE)

Arguments

- **W.hat**: The estimated square unmixing matrix W.
- **A**: The true square mixing matrix A.
- **standardize**: Logical value if A and W.hat need to be standardized. Default is FALSE.

Details

The Amari Error can be used in simulation studies to evaluate the performance of an ICA algorithm. The Amari error is permutation invariant but not scale invariant. Therefore if different algorithms should be compared the matrices should be scaled in the same way. If `standardize` is TRUE, this will be done by the function by standardizing `W.hat` and the inverse of `A` in such a way, that every row has length 1, the largest absolute value of the row has a positive sign and the rows are ordered decreasingly according to their largest values.

Note that this function assumes the ICA model is $X = SA'$, as is assumed by JADE and ics. However fastICA and PearsonICA assume $X = SA$. Therefore matrices from those functions have to be transposed first.

The Amari Error is scaled in such a way, that it takes a value between 0 and 1. And 0 corresponds to an optimal separation.

Value

The value of the Amari Error.

Author(s)

Klaus Nordhausen
References


See Also

ComonGAP, SIR

Examples

```r
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)

W.hat <- JADE(X, 3)$W
amari.error(W.hat, A)
amari.error(W.hat, A, TRUE)
```

---

**AMUSE**

**AMUSE Method for Blind Source Separation**

Description

AMUSE method for the second order blind source separation problem. The function estimates the unmixing matrix in a second order stationary source separation model by jointly diagonalizing the covariance matrix and an autocovariance matrix at lag k.

Usage

```r
AMUSE(x, ...)  
```

## Default S3 method:

```r
AMUSE(x, k = 1, ...)  
```

## S3 method for class 'ts'

```r
AMUSE(x, ...)  
```

Arguments

- `x` a numeric matrix or a multivariate time series object of class `ts`. Missing values are not allowed.
- `k` integer lag for the autocovariance matrix, must be larger than 0. Default is 1.
- `...` further arguments to be passed to or from methods.
Details

The lag $k$ has a huge effect on the performance and it should be chosen so that the eigenvalues of the autocovariance matrix are distinct. The function assumes always as many sources as there are time series.

Value

A list with class ’bss’ containing the following components:

- $w$: estimated unmixing matrix.
- $ev$: eigenvectors of the autocovariance matrix.
- $k$: lag of the autocovariance matrix used.
- $s$: estimated sources as time series objects standardized to have mean 0 and unit variances.

Author(s)

Klaus Nordhausen

References


See Also

ts

Examples

```r
# creating some toy data
A <- matrix(rnorm(9), 3, 3)
s1 <- arima.sim(list(ar=c(0.3, 0.6)), 1000)
s2 <- arima.sim(list(ma=c(-0.3, 0.3)), 1000)
s3 <- arima.sim(list(ar=c(-0.8, 0.1)), 1000)
S <- cbind(s1, s2, s3)
X <- S %>% t(A)
res1 <- AMUSE(X)
res1
coef(res1)
plot(res1)  # compare to plot.ts(S)
```
MD(coef(res1), A)

# input of a time series
X2 <- ts(X, start = c(1961, 1), frequency = 12)
plot(X2)
res2 <- AMUSE(X2, k = 2)
plot(res2)

---

**bss.components**

*Function to Extract Estimated Sources from an Object of Class bss*

**Description**

Extracts the sources estimated by a bss method.

**Usage**

```r
bss.components(object)
```

**Arguments**

- `object`: object of class bss

**Author(s)**

Klaus Nordhausen

**Examples**

```r
A <- matrix(rnorm(9), 3, 3)
s1 <- arima.sim(list(ar = c(0.3, 0.6)), 1000)
s2 <- arima.sim(list(ma = c(-0.3, 0.3)), 1000)
s3 <- arima.sim(list(ar = c(-0.8, 0.1)), 1000)
S <- cbind(s1, s2, s3)
X <- 5 * t(A)
res1 <- AMUSE(X)
head(bss.components(res1))
colMeans(bss.components(res1))
cov(bss.components(res1))
```
Joint Diagonalization of Complex Matrices

Description

This is an R version of Cardoso's joint_diag matlab function for joint diagonalization of k complex-valued square matrices.

Usage

`cjd(X, eps = 1e-06, maxiter = 100)`

Arguments

- **X**: A matrix of k stacked pxp complex matrices with dimension c(kp,p) or an array with dimension c(p,p,k).
- **eps**: Convergence tolerance.
- **maxiter**: Maximum number of iterations.

Value

- **V**: An orthogonal matrix.
- **D**: A stacked matrix with the diagonal matrices or an array with the diagonal matrices. The form of the output depends on the form of the input.

Author(s)

Jean-Francois Cardoso. Ported to R by Klaus Nordhausen.

References


See Also

`rjd`, `rjd.fortran`

Examples

```r
d1 <- diag(complex(real=runif(3,0,2), imaginary=runif(3)))
d2 <- diag(complex(real=runif(3,0,2), imaginary=runif(3)))
d3 <- diag(complex(real=runif(3,0,2), imaginary=runif(3)))
d4 <- diag(complex(real=runif(3,0,2), imaginary=runif(3)))

z <- matrix(runif(9), ncol = 3)
v <- egen(Z %% t(Z))$vectors
```
M1 <- t(V)%*%SD1%*%V
M2 <- t(V)%*%SD2%*%V
M3 <- t(V)%*%SD3%*%V
M4 <- t(V)%*%SD4%*%V
MS <- rbind(M1,M2,M3,M4)
Ms <- array(0,dim=c(3,3,4))
Ms[,1]<-M1
Ms[,3]<-M3
Ms[,2]<-M2
Ms[,4]<-M4
res.array <- cjd(Ms)
res.mat <- cjd(MS)
Re(res.array$V)
V
round(V%*%Re(res.array$V),2)
round(V%*%Re(res.mat$V),2)

coef.bss

Coefficients of a bss Object

Description

Extracts the estimated unmixing matrix from an object of class bss.

Usage

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

Arguments

object

object of class bss.

...

further arguments to be passed to or from methods.

Author(s)

Klaus Nordhausen

Examples

A<- matrix(rnorm(9),3,3)
s1 <- arima.sim(list(ar=c(0.3,0.6)),1000)
s2 <- arima.sim(list(ma=c(-0.3,0.3)),1000)
s3 <- arima.sim(list(ar=c(-0.8,0.1)),1000)
S <- cbind(s1,s2,s3)
X <- S %>% t(A)
res1<--AMUSE(X)
coef(res1)
coef(res1) %>% A # should be a matrix with one dominant element in each row and column
Description

Comon’s GAP criterion to evaluate the performance of an ICA algorithm.

Usage

ComonGAP(A, A.hat)

Arguments

A The true square mixing matrix.
A.hat The estimated square mixing matrix.

Details

Comon’s GAP criterion is permutation and scale invariant. It can take every positive value and 0 corresponds to an optimal separation. If A is however nearly singular the values of the criterion can be huge.

Note that this function assumes the ICA model is $X = SA'$, as is assumed by JADE and ics. However fastICA and PearsonICA assume $X = SA$. Therefore matrices from those functions have to be transposed first.

Value

The value of the Comon’s GAP.

Author(s)

Klaus Nordhausen

References


See Also

amari.error, SIR
Examples

```r
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)

A.hat <- JADE(X, 3)$A
ComonGAP(A, A.hat)
```

### Description

This data set is a toy example for the so called cocktail party problem. In this case three sounds are mixed together with one noise source using four microphones.

### Usage

```r
data("CPPdata")
```

### Format

A data frame with 50000 observations on the following 4 variables:

- **Mic1** the mixture recorded by the first microphone.
- **Mic2** the mixture recorded by the second microphone.
- **Mic3** the mixture recorded by the third microphone.
- **Mic4** the mixture recorded by the fourth microphone.

### Details

The three original source files were kindly provided by Ella Bingham and are also available online at the following locations: `research.ics.aalto.fi/ica/cocktail/source5.wav`, `research.ics.aalto.fi/ica/cocktail/source7.wav` and `research.ics.aalto.fi/ica/cocktail/source9.wav`.

Note that the original sound files are included in the package’s subfolder datafiles. In the example section we illustrate how the CPPdata was created. An example analysis of the data is given in Miettinen et al. (2017).

### Source

`http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi`

### References

Examples

```r
## Not run:
# the data was created as follows:
library(tuneR)
S1 <- readWave(system.file("datafiles/source5.wav", package = "JADE"))
S2 <- readWave(system.file("datafiles/source7.wav", package = "JADE"))
S3 <- readWave(system.file("datafiles/source9.wav", package = "JADE"))

set.seed(321)
NOISE <- noise("white", duration = 50000)
S <- cbind(S1@left, S2@left, S3@left, NOISE@left)
S <- scale(S, center = FALSE, scale = apply(S, 2, sd))
St <- ts(S, start = 0, frequency = 8000)
p <- 4
A <- matrix(runif(p^2, 0, 1), p, p)
A

X <- tcrossprod(St, A)
Xt <- as.ts(X)

colnames(X) <- c("Mic1", "Mic2", "Mic3", "Mic4")
CPPdata <- as.data.frame(X)

## End(Not run)
```

**dj**d

*Function for Joint Diagonalization of k Square Matrices in a Deflation Based Manner*

---

**Description**

This function jointly diagonalizes $k$ real-valued square matrices by searching an orthogonal matrix in a deflation based manner.

**Usage**

```
djd(X, G = "max", r = 2, eps = 1e-06, maxiter = 500)
```

**Arguments**

- **X**: an array containing the $k p$ times $p$ real valued matrices of dimension $c(p, p, k)$.
- **G**: criterion function used for the algorithm. Options are `max`, `pow` and `log`. See details.
- **r**: power value used if $G$="pow" or $G$="max". 0 is not meaningful for this value. See details.
- **eps**: convergence tolerance.
- **maxiter**: maximum number of iterations.
Details
Denote the square matrices as $A_i$, $i = 1, \ldots, k$. This algorithm searches then an orthogonal matrix $W$ so that $D_i = W^t A_i W$ is diagonal for all $i$. If the $A_i$ commute then there is an exact solution. If not, the function will perform an approximate joint diagonalization by maximizing $\sum G(w_j^t A_i w_j)$ where $w_j$ are the orthogonal vectors in $W$.

The function $G$ can be chosen to be of the form $G(x) = |x|^r$ or $G(x) = \log(x)$. If $G=\text{max}$ is chosen, the function $G$ is of the form $G(x) = |x|^r$, and the diagonalization criterion will be maximized globally at each stage by choosing an appropriate initial value from a set of random vectors. If $G=\text{pow}$ or $G=\text{log}$ are chosen, the initial values are the eigenvectors of $A_1$ which plays hence a special role.

Value
The matrix $W$

Author(s)
Klaus Nordhausen, Jari Miettinen

References


Examples
```
Z <- matrix(runif(9), ncol = 3)
U <- eigen(Z)$vectors
d1 <- diag(runif(3))
d2 <- diag(runif(3))
d3 <- diag(runif(3))
d4 <- diag(runif(3))

X.matrix <- array(0, dim=c(3, 3, 4))
X.matrix[,1] <- t(U) %*% d1 %*% U
X.matrix[,2] <- t(U) %*% d2 %*% U
X.matrix[,3] <- t(U) %*% d3 %*% U
X.matrix[,4] <- t(U) %*% d4 %*% U

W1 <- djd(X.matrix)
round(U %*% W1, 4) # should be a signed permutation
# matrix if W1 is correct.

W2 <- djd(X.matrix, r=1)
```
Joint Diagonalization of Real Positive-definite Matrices

Description

This is a slightly modified version of Flury's FG algorithm for the joint diagonalization of \( k \) positive-definite matrices. The underlying function is written in C.

Usage

\[
\text{FG}(X, \text{weight} = \text{NULL}, \text{init} = \text{NULL}, \text{maxiter} = 100, \text{eps} = 1e-06, \text{na.action} = \text{na.fail})
\]

Arguments

- \( X \) A matrix of \( k \) stacked \( p \times p \) matrices with dimension \( c(kp,p) \) or an array with dimension \( c(p,p,k) \).
- \( \text{weight} \) A vector of length \( k \) to give weight to the different matrices, if NULL, all matrices have equal weight.
- \( \text{init} \) Initial value for the orthogonal matrix to be estimated, if NULL, the identity matrix is used.
- \( \text{maxiter} \) Maximum number of iterations.
- \( \text{eps} \) Convergence tolerance.
- \( \text{na.action} \) A function which indicates what should happen when the data contain 'NA's. Default is to fail.

Value

A list with the components

- \( V \) An orthogonal matrix.
- \( D \) A stacked matrix with the diagonal matrices or an array with the diagonal matrices. The form of the output depends on the form of the input.
- \( \text{iter} \) The Fortran function returns also the number of iterations.

Author(s)

Jari Miettinen
References


See Also

rjd, rjd.fortran

---

Fobi

Function to perform FOBI for ICA

Description

The FOBI method for independent component analysis (ICA). We assume that all components have different kurtosis values.

Usage

Fobi(x, na.action = na.fail)

Arguments

x           a numeric matrix.
na.action   A function which indicates what should happen when the data contain 'NA's. Default is to fail.

Value

A list with class 'bss' containing the following components:

W           estimated unmixing matrix.
EV          eigenvectors of autocovariance matrix.
Xmu         the original mean of the data.
S           estimated sources as time series objected standardized to have mean 0 and unit variances.

Note

More general is the function ics in the ICS package.

Author(s)

Klaus Nordhausen
References


See Also

ics

Examples

# 3 source and 3 signals
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)
res <- FOBI(X)
MD(coef(res), A)

---

### JADE

#### JADE Algorithm for ICA

**Description**

This is an R version of Cardoso’s JADE ICA algorithm (for real data) ported from matlab. The ported version is 1.5. Some minor changes compared to the matlab function are explained in the details section. The matlab code can be found for example on the ICA central homepage.

The function uses frjd for the joint diagonalization.

**Usage**

```
JADE(X, n.comp = NULL, eps = 1e-06, maxiter = 100, na.action = na.fail)
```

**Arguments**

- `X`: Numeric data matrix or dataframe.
- `n.comp`: Number of components to extract.
- `eps`: Convergence tolerance.
- `maxiter`: Maximum number of iterations.
- `na.action`: A function which indicates what should happen when the data contain ’NA’s. Default is to fail.
Details

Some minor modifications were done when porting the function to R, and they are:

1. The model assumed here is \( X = S A' + \mu \). Therefore \( S \) and \( X \) have one row per observation. Note that this still differs from the model definition in R of FastICA and PearsonICA but agrees with ics.

2. The whitening covariance matrix is divided by \( n-1 \) and not \( n \) (\( n \) = number of observations).

3. The initial value for the joint diagonalisation is always I.

4. The original eps would be \( \frac{1}{100\sqrt{n}} \).

It is also worth mentioning that the estimated independent components \( S \) are scaled to unit variance and are ordered in such a way, that their fourth moments are in the decreasing order. The signs of the unmixing matrix \( W \) are fixed so that the sum of the elements on each row is positive.

For further details see also the documentation of the original matlab code ("MatlabjadeR.m") on the ICA central homepage (http://www.tsi.enst.fr/icacentral/).

Value

A list with class 'bss' containing the following components:

A The estimated mixing matrix.
W The estimated unmixing matrix.
S Dataframe with the estimated independent components.
xmu The location of the original data.

Author(s)

Jean-Francois Cardoso. Ported to R by Klaus Nordhausen

References


Examples

# 3 source and 3 signals

S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)
res <- JADE(X, 3)
**Description**

This algorithm generalizes the **jade** algorithm, as it provides **JADE** when k is set to the number of dimensions. Otherwise k can be considered as a way to reduce the number of cumulant matrices to be jointly diagonalized. Hence small values of k speed up the method considerably in high-dimensional cases. In general, k can be considered as maximum number of underlying identical sources.

The function uses **FOBI** as an initial estimate and **frjd** for the joint diagonalization.

**Usage**

```r
k_JADE(X, k = 1, eps = 1e-06, maxiter = 100, na.action = na.fail)
```

**Arguments**

- **X**: Numeric data matrix or dataframe.
- **k**: integer value between 1 and the number of columns of X. Default is 1.
- **eps**: Convergence tolerance.
- **maxiter**: Maximum number of iterations.
- **na.action**: A function which indicates what should happen when the data contain 'NA's. Default is to fail.

**Details**

The order of the estimated components is fixed so that their fourth moments are in the decreasing order.
Value

A list with class ‘bss’ containing the following components:

- \( A \) The estimated mixing matrix.
- \( \hat{W} \) The estimated unmixing matrix.
- \( S \) Matrix with the estimated independent components.
- \( xmu \) The location of the original data.

Note

The function uses F0BI as initial estimate and frjd for the joint diagonalization.

Author(s)

Jari Miettinen

References


See Also

jade, F0BI, frjd

Examples

# 3 source and 3 signals
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)
res_k1 <- k_JADE(X, 1)
res_k1$A
res_k1$W
res_k1$S[1:10,]

MD(coef(res_k1), A)
**Description**
Computes the Minimum Distance index MD to evaluate the performance of an ICA algorithm.

**Usage**
\[ \text{MD}(\hat{W}, A) \]

**Arguments**
- \( \hat{W} \) The estimated square unmixing matrix \( W \).
- \( A \) The true square mixing matrix \( A \).

**Details**
\[ \text{MD}(\hat{W}, A) = \frac{1}{\sqrt{p-1}} \inf_{P_D} ||PD\hat{W}A - I||, \]

where \( P \) is a permutation matrix and \( D \) a diagonal matrix with nonzero diagonal entries.

The step that minimizes the index of the set over all permutation matrix can be expressed as a linear sum assignment problem (LSAP) for which we use as solver the Hungarian method implemented as `solve_lasp` in the `clue` package.

Note that this function assumes the ICA model is \( X = SA' \), as is assumed by `JADE` and `ics`. However `fastICA` and `PearsonICA` assume \( X = SA \). Therefore matrices from those functions have to be transposed first.

The MD index is scaled in such a way, that it takes a value between 0 and 1. And 0 corresponds to an optimal separation.

**Value**
The value of the MD index.

**Author(s)**
Klaus Nordhausen

**References**


See Also
ComonGAP, SIR, amari.error, solve_LSAP

Examples
```r
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)

W.hat <- JADE(X, 3)$W
MD(W.hat, A)
```

**multscatter**

Function to Compute Several Scatter Matrices for the Same Data

Description

The function can be used to compute several scatter matrices for the same data.

Usage

`multscatter(scatterlist, X, toshape = TRUE)`

Arguments

- `scatterlist`: a vector with the names of the scatter matrices to be computed. Note that each of these functions should only return a matrix of size p times p.
- `X`: the n times p data matrix for which the scatter should be computed.
- `toshape`: logical, whether scatter matrices should be converted to shape matrices. If TRUE, all matrices will have determinant 1.

Details

It is important that the functions do not need any additional input and that they return only the p times p scatter matrix. Hence it might be sometimes necessary to write wrappers for some of the functions. See examples.

Value

An array of dimension c(p,p,k) where k is the number of scatter matrices.
**NSS.JD**

**Author(s)**

Klaus Nordhausen

**Examples**

```r
# example requires the packages ICS and ICSNP
library(ICSNP)
X <- cbind(rexp(1000), rt(1000,6), runif(1000))

my.tm1 <- function(X,df=1) tM(X,)$W
my.tm2 <- function(X,df=2) tM(X,)$W

multscatter(c("cov","cov4","HP1.shape","my.tm1", "my.tm2"), X)
multscatter(c("cov","cov4","HP1.shape","my.tm1", "my.tm2"), X, toshape=FALSE)
```

---

**NSS. JD**

**NSS.JD Method for Nonstationary Blind Source Separation**

**Description**

The NSS.JD method for nonstationary blind source separation. The method first whitens the complete data and then divides it into K time intervals. Then `frjd` is used to jointly diagonalize the covariance matrices computed for the individual time intervals to find the sources.

**Usage**

```r
NSS.JD(X, ...)
```

### Default S3 method:

```r
NSS.JD(X, K=12, Tau=0, n.cuts=NULL, eps = 1e-06, maxiter = 100, ...)
```

### S3 method for class 'ts'

```r
NSS.JD(X, ...)
```

**Arguments**

- **X**: a numeric matrix or a multivariate time series object of class `ts`. Missing values are not allowed.
- **K**: number of intervals to be used.
- **Tau**: By default 0 which means covariance are computed of each time interval, if Tau is an integer > 0 then rather autocovariance matrices at lag Tau are used for the joint diagonaliation.
- **n.cuts**: if NULL, then the time series is divided into K equally long intervals. To specify intervals n.cuts should be given in the form c(1,n.cut.1,...,n.cut.k, nrow(X)) to specify where to split the time series.
- **eps**: maximum number of iterations for `frjd`.
- **maxiter**: convergence tolerance for `frjd`.
- **...**: further arguments to be passed to or from methods.
Details
The model assumes that the mean of the p-variate time series is constant but the variances change over time.

Value
A list with class 'bss' containing the following components:

- \( w \) estimated unmixing matrix.
- \( k \) the lag used for the autocovariance matrix.
- \( n.\text{cut} \) specifying the intervals where data is split
- \( K \) the number of intervals used
- \( S \) estimated sources as time series objected standardized to have mean 0 and that the variance of the sources are 1.

Author(s)
Klaus Nordhausen

References

See Also
ts, NSS.SD, NSS.TD.JD

Examples
```r
n <- 1000
s1 <- rnorm(n)
s2 <- 2*sin(pi/200*1:n)* rnorm(n)
s3 <- c(rnorm(n/2), rnorm(100,0,2), rnorm(n/2-100,0,1.5))
S <- cbind(s1,s2,s3)
plot.ts(S)
A<-'matrix(rnorm(9),3,3)
X<-' S'S%t(A)
NSS2 <- NSS.JD(X)
```
NSS2
MD(coef(NSS2), A)
plot(NSS2)
cor(NSS2$S, S)

NSS2b <- NSS.JD(X, Tau=1)
MD(coef(NSS2b), A)

NSS2c <- NSS.JD(X, n.cuts=c(1, 300, 500, 600, 1000))
MD(coef(NSS2c), A)

NSS SD Method for Nonstationary Blind Source Separation

Description
The NSS.SD method for nonstationary blind source separation. The function estimates the unmixing matrix in a nonstationary source separation model by simultaneously diagonalizing two covariance matrices computed for different time intervals.

Usage
NSS.SD(X, ...)

## Default S3 method:
NSS.SD(X, n.cut=NULL, ...)
## S3 method for class 'ts'
NSS.SD(X, ...)

Arguments

X a numeric matrix or a multivariate time series object of class ts. Missing values are not allowed.

n.cut either an integer between 1 and nrow(X) or an vector of length 3 of the form c(1,n.cut,nrow(X)) to specify where to split the time series. If NULL, then c(1,floor(nrow(X)/2),nrow(X)) is used.

... further arguments to be passed to or from methods.

Details
The model assumes that the mean of the p-variate time series is constant but the variances change over time.
Value

A list with class ‘bss’ containing the following components:

- \( W \): estimated unmixing matrix.
- \( EV \): eigenvalues from the eigenvalue-eigenvector decomposition.
- \( n.\, cut \): specifying the intervals where data is split.
- \( S \): estimated sources as time series objected standardized to have mean 0 and that the sources in the first interval are 1.

Author(s)

Klaus Nordhausen

References


See Also

ts, NSS.JD, NSS.TD.JD

Examples

```r
n <- 1000
s1 <- rnorm(n)
s2 <- 2*sin(pi/200*1:n)* rnorm(n)
s3 <- c(rnorm(n/2), rnorm(100,0,2), rnorm(n/2-100,0,1.5))
S <- cbind(s1,s2,s3)
plot.ts(S)
A<-matrix(rnorm(9),3,3)
X<- S%*%t(A)

NSS1 <- NSS.SD(X)
NSS1
MD(coef(NSS1),A)
plot(NSS1)
cor(NSS1$S,S)

NSS1b <- NSS.SD(X, n.cut=400)
MD(coef(NSS1b),A)
```
NSS.TD.JD

NSS.TD.JD Method for Nonstationary Blind Source Separation

Description

The NSS.TD.JD method for nonstationary blind source separation. The method first whitens the complete data and then divides it into K time intervals. It is then assumed that within each interval the time series is approximately second order stationary and within each interval L autocovariance are computed. The underlying sources are then found by jointly diagonalizing the K*L autocovariance matrices using frjd.

Usage

NSS.TD.JD(X, ...)  
## Default S3 method:  
NSS.TD.JD(X, K=12, Tau=0:11, n.cuts=NULL, eps = 1e-06, maxiter = 100, ...)  
## S3 method for class 'ts'  
NSS.TD.JD(X, ...)

Arguments

- **X**: a numeric matrix or a multivariate time series object of class ts. Missing values are not allowed.
- **K**: number of intervals to be used.
- **Tau**: lags for the autocovariance matrices to be computed within each interval.
- **n.cuts**: if NULL, then the time series is divided into K equally long intervals. To specify intervals n.cuts should be given in the form c(1,n.cut.1,...,n.cut.k, nrow(X)) to specify where to split the time series.
- **eps**: maximum number of iterations for frjd.
- **maxiter**: convergence tolerance for frjd.
- **...**: further arguments to be passed to or from methods.

Details

The model assumes that the mean of the p-variate time series is constant but the variances change over time.

NSS1c <- NSS.SD(X, n.cut=c(1,600,1000))  
MD(coef(NSS1c),A)
Value

A list with class 'bss' containing the following components:

- \( \mathbf{W} \) estimated unmixing matrix.
- \( k \) the lags used for the autocovariance matrix used in each interval.
- \( nNcut \) specifying the intervals where data is split
- \( K \) the number of intervals used
- \( S \) estimated sources as time series objected standardized to have mean 0 and that the sources 1.

Author(s)

Klaus Nordhausen

References


See Also

ts, NSS.JD, NSS.JD, SOBI

Examples

```r
n <- 1000
s1 <- rnorm(n)
s2 <- 2*sin(pi/200*1:n)* rnorm(n)
s3 <- c(rnorm(n/2), rnorm(100,0,2), rnorm(n/2-100,0,1.5))
S <- cbind(s1,s2,s3)
plot.ts(S)
A<-matrix(rnorm(9),3,3)
X<- 5%*t(A)

NSS3 <- NSS.TD.JD(X)
NSS3
MD(coef(NSS3),A)
plot(NSS3)
cor(NSS3$S, S)
```
**plot.bss**

Plotting an Object of Class `bss`

Description

Plots the estimated sources resulting from an `bss` method. If the `bss` method is based on second order assumptions and returned the sources as a time series object it will plot the sources using `plot.ts`, otherwise it will plot a scatter plot matrix using `pairs` or `plot` if there are only two sources.

Usage

```r
## S3 method for class 'bss'
plot(x, ...)```

Arguments

- `x` : object of class `bss`
- `...` : further arguments to be passed to or from methods.

Author(s)

Klaus Nordhausen

See Also

- `plot.ts`, `pairs`, `plot`

Examples

```r
A <- matrix(rnorm(9), 3, 3)
s1 <- arima.sim(list(ar=c(0.3, 0.6)), 1000)
s2 <- arima.sim(list(ma=c(-0.3, 0.3)), 1000)
s3 <- arima.sim(list(ar=c(-0.8, 0.1)), 1000)
S <- cbind(s1, s2, s3)
X <- S %*% t(A)
res1 <- AMUSE(X)
plot(res1)
# not so useful:
plot(res1, plot.type = "single", col=1:3)

# not meaningful for this data
```
res2 <- JADE(X)
plot(res2)

print.bss

**Printing an Object of Class bss**

**Description**

Prints an object of class bss. It prints all elements of the list of class bss except the component `S` which is the source matrix.

**Usage**

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

**Arguments**

- `x`: object of class bss.
- `...`: further arguments to be passed to or from methods.

**Author(s)**

Klaus Nordhausen

---

**rjd**

**Joint Diagonalization of Real Matrices**

**Description**

This is an R version of Cardoso’s rjd matlab function for joint diagonalization of k real-valued square matrices. A version written in C is also available and preferrable.

**Usage**

```r
rjd(X, eps = 1e-06, maxiter = 100, na.action = na.fail)
frjd(X, weight = NULL, maxiter = 100, eps = 1e-06, na.action = na.fail)
rjd.fortran(X, weight = NULL, maxiter = 100, eps = 1e-06, na.action = na.fail)
```
Arguments

- **X**: A matrix of k stacked pxp matrices with dimension c(kp,p) or an array with dimension c(p,p,k).
- **weight**: A vector of length k to give weight to the different matrices, if NULL, all matrices have equal weight.
- **eps**: Convergence tolerance.
- **maxiter**: Maximum number of iterations.
- **na.action**: A function which indicates what should happen when the data contain 'NA's. Default is to fail.

Details

Denote the square matrices as \( A_i, i = 1, \ldots, k \). The algorithm searches an orthogonal matrix \( V \) so that \( D_i = V^T A_i V \) is diagonal for all \( i \). If the \( A_i \) commute then there is an exact solution. Otherwise, the function will perform an approximate joint diagonalization by trying to make the \( D_i \) as diagonal as possible.

Cardoso points out that notion of approximate joint diagonalization is ad hoc and very small values of \( \text{eps} \) make in that case not much sense since the diagonality criterion is ad hoc itself.

Value

A list with the components

- **V**: An orthogonal matrix.
- **D**: A stacked matrix with the diagonal matrices or an array with the diagonal matrices. The form of the output depends on the form of the input.
- **iter**: The \( \text{frjd} \) function returns also the number of iterations.

Author(s)

Jean-Francois Cardoso. Ported to \( \text{R} \) by Klaus Nordhausen. C code by Jari Miettinen

References


Examples

```r
Z <- matrix(rnorm(9), ncol = 3)
U <- eigen(Z)@vectors
D1 <- diag(rnorm(3))
D2 <- diag(rnorm(3))
D3 <- diag(rnorm(3))
```
D4 <- diag(runif(3))

X.matrix <- rbind(t(U) %*% D1 %*% U, t(U) %*% D2 %*% U, t(U) %*% D3 %*% U, t(U) %*% D4 %*% U)
res.matrix <- rjd(X.matrix)
res.matrix$V
round(U %*% res.matrix$V, 4) # should be a signed permutation
# matrix if V is correct.

round(res.matrix$D, 4)

# compare to C version
#res.matrix.C <- frjd(X.matrix)
#res.matrix.C$V
#round(U %*% res.matrix.C$V, 4)
#round(res.matrix.C$D, 4)

X.array <- aperm(array(t(X.matrix), dim = c(3,3,4)), c(2,1,3))
res.array <- rjd(X.array)
round(res.array$D, 4)

res.array.C <- frjd(X.array)
round(res.array.C$D, 4)

---

**SIR**  
*Signal to Interference Ratio*

**Description**
Computes the signal to interference ratio between true and estimated signals.

**Usage**
```
SIR(S, S.hat)
```

**Arguments**

- **S**  
  Matrix or dataframe with the true numeric signals.

- **S.hat**  
  Matrix or dataframe with the estimated numeric signals.

**Details**
The signal to interference ratio is measured in dB and values over 20 are thought to be good. It is scale and permutation invariant and can be seen as measuring the correlation between the matched true and estimated signals.
Value
The value of the signal to interference ratio.

Author(s)
Klaus Nordhausen

References

See Also
amari.error, CommonGAP

Examples
S <- cbind(rt(1000, 4), rnorm(1000), runif(1000))
A <- matrix(rnorm(9), ncol = 3)
X <- S %*% t(A)

S.hat <- JADE(X, 3)$S
SIR(S, S.hat)

SOBI Method for Blind Source Separation

Description
The SOBI method for the second order blind source separation problem. The function estimates the unmixing matrix in a second order stationary source separation model by jointly diagonalizing the covariance matrix and several autocovariance matrices at different lags.

Usage
SOBI(X, ...)

## Default S3 method:
SOBI(X, k=12, method="frjd", eps = 1e-06, maxiter = 100, ...)
## S3 method for class 'ts'
SOBI(X, ...)
Arguments

- **X**: a numeric matrix or a multivariate time series object of class `ts`. Missing values are not allowed.
- **k**: if a single integer, then the lags 1:k are used, if an integer vector, then these are used as the lags.
- **method**: method to use for the joint diagonalization, options are `djd`, `rjd` and `frjd`.
- **eps**: maximum number of iterations.
- **maxiter**: convergence tolerance.
- **...**: further arguments to be passed to or from methods.

Details

The order of the estimated components is fixed so that the sums of squared autocovariances are in the decreasing order.

Value

A list with class 'bss’ containing the following components:

- **w**: estimated unmixing matrix.
- **k**: lags used.
- **method**: method used for the joint diagonalization.
- **S**: estimated sources as time series objected standardized to have mean 0 and unit variances.

Author(s)

Klaus Nordhausen

References


See Also

- `ts`
Examples

# creating some toy data
A <- matrix(rnorm(9), 3, 3)
s1 <- arima.sim(list(ar=c(0.3, 0.6)), 1000)
s2 <- arima.sim(list(ma=c(-0.3, 0.3)), 1000)
s3 <- arima.sim(list(ar=c(-0.8, 0.1)), 1000)

S <- cbind(s1, s2, s3)
X <- S %*% t(A)

res1 <- SOBI(X)
res1
coef(res1)
plot(res1) # compare to plot.ts(S)
MD(coef(res1), A)

# input of a time series
X2 <- ts(X, start=c(1961, 1), frequency=12)
plot(X2)
res2 <- SOBI(X2, k=c(5, 10, 1, 4, 2, 9, 10))
plot(res2)
Index

*Topic array
  cjd, 7
djd, 11
  FG, 13
  rjd, 28
*Topic datasets
  CPPdata, 10
*Topic methods
  coef.bss, 8
  plot.bss, 27
  print.bss, 28
*Topic multivariate
  amari.error, 3
  AMUSE, 4
  bss.components, 6
  ComonGAP, 9
  FOBI, 14
  JADE, 15
  k_JADE, 17
  MD, 19
  multiscatter, 20
  NSS.JD, 21
  NSS SD, 23
  NSS TD Jd, 25
  SIR, 30
  SOBI, 31
*Topic package
  JADE-package, 2
*Topic ts
  AMUSE, 4
  NSS.JD, 21
  NSS SD, 23
  NSS TD Jd, 25
  SOBI, 31
  amari.error, 3, 9, 20, 31
  AMUSE, 4
  bss.components, 6

cjd, 7
  coef.bss, 8
  ComonGAP, 4, 9, 20, 31
  CPPdata, 10
  djd, 11, 32
  FG, 13
  FOBI, 14, 17, 18
  frjd, 15, 17, 18, 21, 25, 32
  frjd (rjd), 28
  ics, 14, 15
  JADE, 3, 9, 15, 17–19
  JADE-package, 2
  k_JADE, 17
  MD, 19
  multiscatter, 20
  NSS.JD, 21, 24, 26
  NSS SD, 22, 23
  NSS TD Jd, 22, 24, 25
  pairs, 27
  plot, 27
  plot.bss, 27
  plot.ts, 27
  print.bss, 28
  rjd, 7, 14, 28, 32
  rjd.fortran, 7, 14
  SIR, 4, 9, 20, 30
  SOBI, 26, 31
  solve_LSAP, 20
  ts, 4, 5, 21–26, 32