Package ‘singR’

February 9, 2024

Type Package

Title Simultaneous Non-Gaussian Component Analysis

Version 0.1.2

Date 2024-02-07

Author Liangkang Wang [aut, cre] (https://orcid.org/0000-0003-3393-243X),
Irina Gaynanova [aut] (https://orcid.org/0000-0002-4116-0268),
Benjamin Risk [aut] (https://orcid.org/0000-0003-1090-0777)

Maintainer Liangkang Wang <liangkang_wang@brown.edu>

Description Implementation of SING algorithm to extract joint and individual non-Gaussian components from two datasets. SING uses an objective function that maximizes the skewness and kurtosis of latent components with a penalty to enhance the similarity between subject scores. Unlike other existing methods, SING does not use PCA for dimension reduction, but rather uses non-Gaussianity, which can improve feature extraction. Benjamin B. Risk, Irina Gaynanova (2021) <doi:10.1214/21-AOAS1466>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo

Imports MASS (>= 7.3-57), Rcpp (>= 1.0.8.3), clue (>= 0.3-61), gam (>= 1.20.1), ICtest (>= 0.3-5)

Suggests knitr, covr, testthat (>= 3.0.0), rmarkdown

Config/testthat/edition 3

Depends R (>= 2.10)

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-02-09 00:20:02 UTC
angleMatchICA

**Description**

angleMatchICA match the columns of Mx and My, using the n x p parameterization of the JIN decomposition assumes

**Usage**

angleMatchICA(Mx, My, Sx = NULL, Sy = NULL)

**Arguments**

- **Mx**: Subject score for X matrix of n x n.comp
- **My**: Subject score for Y matrix of n x n.comp
- **Sx**: Variable loadings for X matrix of n.comp x px
- **Sy**: Variable loadings for Y matrix of n.comp x py
aveM

Value

a list of matrixes: ## Mx: ## My: ## matchedangles: ## allangles: ## perm: ## omangles:

aveM

Average Mj for Mx and My Here subjects are by rows, columns correspond to components

Description

Average Mj for Mx and My Here subjects are by rows, columns correspond to components

Usage

aveM(mjX, mjY)

Arguments

mjX n x rj
mjY n x rj

Value

a new Mj

Examples

#get simulation data
data(exampledata)
data=exampledata

# To get n.comp value, we can use NG_number function.
# use JB statistic as the measure of nongaussianity to run lngca with df=0
output_JB=singR(dX=exampledata$dX,dY=exampledata$dY,
df=0,rho_extent="small",distribution="JB",individual=TRUE)
est.Mj = aveM(output_JB$est.Mjx,output_JB$est.Mjy)
**calculateJB**  
Calculates the sum of the JB scores across all components, useful for determining rho.

**Description**

We measure non-Gaussianity using Jarque-Bera (JB) statistic, which is a weighted combination of squared skewness and kurtosis, *JB paper*. The data has to be standardized and mean 0 and sd to 1.

**Usage**

```r
calculateJB(S = NULL, U = NULL, X = NULL, alpha = 0.8)
```

**Arguments**

- `S`: the variable loadings r x px.
- `U`: U matrix for matched columns rj x n
- `X`: whitened data matrix n x px, data = whitenerXA %*% dXcentered
- `alpha`: JB weighting of skewness and kurtosis. default = 0.8

**Value**

the sum of JB score across all components.

---

**covwhitener**  
Returns square root of the precision matrix for whitening

**Description**

Returns square root of the precision matrix for whitening

**Usage**

```r
covwhitener(X, n.comp = ncol(X), center.row = FALSE)
```

**Arguments**

- `X`: Matrix
- `n.comp`: the number of components
- `center.row`: whether to center

**Value**

square root of the precision matrix for whitening
create.graph.long  

create.graph.long create graph dataset with netmat and mmp_order a data.frame called with vectorization of reordered netmat by mmp_order.

Description

create graph dataset with netmat and mmp_order a data.frame called with vectorization of reordered netmat by mmp_order.

Usage

create.graph.long(gmatrix, sort_indices = NULL)

Arguments

- **gmatrix**: netmat
- **sort_indices**: mmp_order

Value

a data.frame with vectors: ## X1: vector of numerics. ## X2: vector of numerics. ## value: vectorization of reordered netmat by mmp_order.

curvilinear  

Curvilinear algorithm with r0 joint components

Description

The curvilinear algorithm is modified from Wen and Yin paper.

Usage

curvilinear(  
  Ux,  
  Uy,  
  xData,  
  yData,  
  invLx,  
  invLy,  
  rho,  
  tau = 0.01,  
  alpha = 0.8,  
  maxiter = 1000,  
  tol = 1e-06,  
  rj  
)


## Arguments

- `Ux` : Matrix with `n.comp x n`, initial value of `Ux`, comes from `greedyMatch`.
- `Uy` : Matrix with `n.comp x n`, initial value of `Uy`, comes from `greedyMatch`.
- `xData` : Matrix with `n x px`, `Xw = Lx %*% Xc`.
- `yData` : Matrix with `n x py`, `Yw = Ly %*% Yc`.
- `invLx` : Inverse matrix of `Lx`, matrix `n x n`.
- `invLy` : Inverse matrix of `Ly`, matrix `n x n`.
- `rho` : The weight parameter of matching relative to non-gaussianity.
- `tau` : Initial step size, default value is 0.01.
- `alpha` : Controls weighting of skewness and kurtosis. Default value is 0.8, which corresponds to the Jarque-Bera test statistic with 0.8 weighting on squared skewness and 0.2 on squared kurtosis.
- `maxiter` : Default value is 1000.
- `tol` : The threshold of change in `Ux` and `Uy` to stop the curvilinear function.
- `rj` : The joint rank, comes from `greedyMatch`.

## Value

A list of matrices:

- `Ux` : Optimized `Ux` with matrix `n.comp x n`.
- `Uy` : Optimized `Uy` with matrix `n.comp x n`.
- `tau` : Step size.
- `iter` : Number of iterations.
- `error` : `PMSE(Ux,Uxnew)+PMSE(Uy,Uynew)`.
- `obj` : Objective Function value.

---

**curvilinear_c**  
*Curvilinear algorithm based on C code with r0 joint components*

## Description

#' The curvilinear algorithm is modified from Wen and Yin paper.
Usage

```r
curvilinear_c(
  Ux, 
  Uy, 
  xData, 
  yData, 
  invLx, 
  invLy, 
  rho, 
  tau = 0.01, 
  alpha = 0.8, 
  maxiter = 1000, 
  tol = 1e-06, 
  rj
)
```

Arguments

- **Ux**: Matrix with n.comp x n, initial value of Ux, comes from greedyMatch.
- **Uy**: Matrix with n.comp x n, initial value of Uy, comes from greedyMatch.
- **xData**: matrix with n x px, Xw = Lx %*% Xc.
- **yData**: matrix with n x py, Yw = Ly %*% Yc.
- **invLx**: Inverse matrix of Lx, matrix n x n.
- **invLy**: Inverse matrix of Ly, matrix n x n.
- **rho**: the weight parameter of matching relative to non-gaussianity.
- **tau**: initial step size, default value is 0.01
- **alpha**: controls weighting of skewness and kurtosis. Default value is 0.8, which corresponds to the Jarque-Bera test statistic with 0.8 weighting on squared skewness and 0.2 on squared kurtosis.
- **maxiter**: default value is 1000
- **tol**: the threshold of change in Ux and Uy to stop the curvilinear function
- **rj**: the joint rank, comes from greedyMatch.

Value

a list of matrices:

- **Ux**: Optimized Ux with matrix n.comp x n.
- **Uy**: Optimized Uy with matrix n.comp x n.
- **tau**: step size
- **iter**: number of iterations.
- **error**: PMSE(Ux,Uxnew)+PMSE(Uy,Uynew)
- **obj**: Objective Function value
est.M.ols  

Estimate mixing matrix from estimates of components

Description

Estimate mixing matrix from estimates of components

Usage

est.M.ols(sData, xData, intercept = TRUE)

Arguments

- sData: S rx x px
- xData: dX n x px
- intercept: default = TRUE

Value

a matrix Mx, dimension n x rx.

exampledata  

Data for simulation example 1

Description

Data for simulation example 1

Usage

exampledata

Format

A data list with 10 subsets:

- dX: original data matrix for X with n x px, 48x3602
- dY: original data matrix for Y with n x py, 48x4950
- mj: true mj matrix, n x rj, 48x2
- sIx: true S matrix of independent non-Gaussian components in X, ri_x x px, 2x3602
- sIy: true S matrix of independent non-Gaussian components in Y, ri_y x py, 2x4950
- sjx: true S matrix of joint non-Gaussian components in X, rj x px, 2x3602
- sjy: true S matrix of joint non-Gaussian components in Y, rj x py, 2x4950
- snr: signal to noise ratio
- R2x: R2 for x data
- R2y: R2 for y data
gen.inits

Generate initialization from specific space

Description

Generate initialization from specific space

Usage

```r
gen.inits(p, d, runs, orth.method = c("svd", "givens"))
```

Arguments

- `p`: `p*p` orthodox matrix
- `d`: `p*d` orthodox matrix
- `runs`: the number of orthodox matrix
- `orth.method`: orthodox method

Value

a list of initialization of mixing matrices.

Examples

```r
gen.inits(2,3,3,'svd')
```

greedymatch

Greedy Match

Description

Greedy Match matches a column of Mx and My by minimizing chordal distance between vectors, removes the matched columns and then finds the next pair. This equivalent to maximizing absolute correlation for data in which each column has mean equal to zero. Returns permuted columns of Mx and My. This function does not do any scaling or sign flipping. For this matching to coincide with angle matching, the columns must have zero mean.

Usage

```r
greedymatch(Mx, My, Ux, Uy)
```
lngca

Decompose the original data through LNGCA method.

Description

Implements the methods of linear non-Gaussian component analysis (LNGCA) and likelihood component analysis (when using a density, e.g., tilted Gaussian) from the LNGCA paper

Usage

lngca(
  xData,
  n.comp = NULL,
  Ux.list = NULL,
  whiten = c("sqrtprec", "eigenvect", "none"),
  maxit = 1000,
  eps = 1e-06,
  verbose = FALSE,
  restarts.pbyd = 0,
  restarts.dbyd = 0,
  distribution = c("JB", "tiltedgaussian", "logistic"),
  density = FALSE,
lngca

```r
g out.all = FALSE,
 orth.method = c("svd", "givens"),
 df = 0,
 stand = FALSE,
...
```

**Arguments**

- **xData**
  - the original dataset for decomposition, matrix of n x px.

- **n.comp**
  - the number of components to be estimated.

- **Ux.list**
  - list of user specified initial values for Ux. If null, will generate random orthogonal matrices. See restarts.pbyd and restarts.dbyd

- **whiten**
  - whitening method. Defaults to "svd" which uses the n left eigenvectors divided by sqrt(px-1) by 'eigenvec'. Optionally uses the square root of the n x n "precision" matrix by 'sqrtprec'.

- **maxit**
  - max iteration, default = 1000

- **eps**
  - default = 1e-06

- **verbose**
  - default = FALSE

- **restarts.pbyd**
  - default = 0. Generates p x d random orthogonal matrices. Use a large number for large datasets. Note: it is recommended that you run lngca twice with different seeds and compare the results, which should be similar when a sufficient number of restarts is used. In practice, stability with large datasets and a large number of components can be challenging.

- **restarts.dbyd**
  - default = 0. These are d x d initial matrices padded with zeros, which results in initializations from the principal subspace. Can speed up convergence but may miss low variance non-Gaussian components.

- **distribution**
  - distribution methods with default to tilted Gaussian. "logistic" is similar to infomax ICA, JB is capable of capture super and sub Gaussian distribution while being faster than tilted Gaussian. (tilted Gaussian tends to be most accurate, but computationally much slower.)

- **density**
  - return the estimated tilted Gaussian density? default = FALSE

- **out.all**
  - default = FALSE

- **orth.method**
  - default = 'svd'. Method to generate random initial matrices. See [gen.inits()]

- **df**
  - default = 0, df of the spline used in fitting the non-parametric density. use df=8 or so for tilted gaussian. set df=0 for JB and logistic.

- **stand**
  - whether to standardize the data to have row and column means equal to 0 and the row standard deviation equal to 1 (i.e., all variables on same scale). Often used when combined with singR for data integration.

- **...**
  - other arguments to tiltedgaussian estimation
**Value**

Function outputs a list including the following:

- **U** matrix $r \times n$, part of the expression that $Ax = Ux \cdot Lx$ and $Ax \cdot Xc = Sx$, where $Lx$ is the whitener matrix.
- **loglik** the value of log-likelihood in the lngca method.
- **S** the variable loading matrix $r \times px$, each row is a component, which can be used to measure nongaussianity.
- **df** degree of freedom.
- **distribution** the method used for data decomposition.
- **whitener** A symmetric whitening matrix $n \times n$ from $dX$, the same with $whitenerXA = \text{est.sigmaXA}^{\%^\% -0.5}$.
- **M** $Mx$ matrix with $n \times rx$.
- **nongaussianity** the nongaussianity score for each component saved in $S$ matrix.

**Examples**

```r
# get simulation data
data(exampledata)
data=exampledata

# To get n.comp value, we can use NG_number function.
# use JB statistic as the measure of nongaussianity to run lngca with df=0
estX_JB = lngca(xData = data$dX, n.comp = 4,
whiten = 'sqrtprec', restarts.pbyd = 20, distribution='JB',df=0)

# use the tiltedgaussian distribution to run lngca with df=8. This takes a long time:
estX_tilt = lngca(xData = data$dX, n.comp = 4,
whiten = 'sqrtprec', restarts.pbyd = 20, distribution='tiltedgaussian',df=8)

# true non-gaussian component of $Sx$, include individual and joint components
trueSx = rbind(data$sjX,data$siX)

# use pmse to compare the difference of the two methods
pmse(S1 = t(trueSx),S2=t(estX_JB$S),standardize = TRUE)
pmse(S1 = t(trueSx),S2=t(estX_tilt$S),standardize = TRUE)

# the lngca using tiltedgaussian tends to be more accurate
# with smaller pmse value, but takes longer to run.
```
**matchICA**

Description
match ICA

Usage
matchICA(S, template, M = NULL)

Arguments
- **S**: loading variable matrix
- **template**: template for match
- **M**: subject score matrix

Value
the match result

**NG_number**

find the number of non-Gaussian components in the data.

Description
find the number of non-Gaussian components in the data.

Usage
NG_number(data, type = "S3")

Arguments
- **data**: original matrix with n x p.
- **type**: 'S1', 'S2' or 'S3'

Value
the number of non-Gaussian components in the data.

Examples
library(singR)
data("exampledata")
data=exampledata
NG_number(data$dx)
**orthogonalize**

**Orthogonalization of matrix**

**Description**

Orthogonalization of matrix

**Usage**

`orthogonalize(W)`

**Arguments**

`W` arbitrary matrix

**Value**

orthogonalized matrix

---

**permmatRank_joint**

**Permutation test to get joint components ranks**

**Description**

Permutation test to get joint components ranks

**Usage**

`permmatRank_joint(matchedResults, nperms = 100)`

**Arguments**

`matchedResults` results generated by angleMatchICA
`nperms` the number of permutation

**Value**

a list of matrixes

## pvalues: pvalues for the matched columns don't have correlation.

## corperm: correlation value for original Mx with each random permutation of My.

## cormatched: the correlation for each pair of matched columns.
**permTestJointRank**  

*Permutation test with Greedymatch*

**Description**
Permutation test with Greedymatch

**Usage**

```r
permTestJointRank(
  MatchedMx,
  MatchedMy,
  nperm = 1000,
  alpha = 0.01,
  multicore = 0
)
```

**Arguments**

- `MatchedMx`: matrix with nsubject x n.comp.X, comes from greedymatch
- `MatchedMy`: matrix with nsubject2 x n.comp.Y, comes from greedymatch
- `nperm`: default value = 1000
- `alpha`: default value = 0.01
- `multicore`: default value = 0

**Value**

- a list of matrixes  
  - rj: joint component rank  
  - pvalues: pvalue for the components(columns) not matched  
  - fwer_alpha: quantile of corr permutation with 1- alpha

---

**pmse**  

*Permutation invariant mean squared error*

**Description**
Permutation invariant mean squared error

**Usage**

```r
pmse(M1 = NULL, M2 = NULL, S1 = NULL, S2 = NULL, standardize = FALSE)
```
Arguments

- **M1** Subject score 1 matrix \( r \times n \).
- **M2** Subject score 2 matrix \( r \times n \).
- **S1** Loading 1 with matrix \( p \times r \).
- **S2** Loading 2 with matrix \( p \times r \).
- **standardize** whether to standardize

Value

permutation invariant mean squared error

Examples

```r
# get simulation data
data(exampledata)

# use JB stat to compute with singR
output_JB = singR(dX = exampledata$dX, dY = exampledata$dY, df = 0, rho_extent = "small", distribution = "JB", individual = TRUE)

# use pmse to measure difference from the truth
pmse(M1 = t(output_JB$est.Mj), M2 = t(exampledata$mj), standardize = TRUE)
```

---

**signchange**

*Sign change for S matrix to image*

Description

Sign change for S matrix to image

Usage

`signchange(S, M = NULL)`

Arguments

- **S** \( S, r \times px \).
- **M** \( Mx, n \times r \).

Value

a list of positive S and positive Mx.
SingR

Simultaneous Non-Gaussian Component analysis for data integration.

Description

This function combines all steps from the SING paper

Usage

```r
singR(
  dX,
  dY,
  n.comp.X = NULL,
  n.comp.Y = NULL,
  df = 0,
  rho.extent = c("small", "medium", "large"),
  Cplus = TRUE,
  tol = 1e-10,
  stand = FALSE,
  distribution = "JB",
  maxiter = 1500,
  individual = FALSE,
  whiten = c("sqrtprec", "eigenvect", "none"),
  restarts.dbyd = 0,
  restarts.pbyd = 20
)
```

Arguments

- **dX**: original dataset for decomposition, matrix of n x px.
- **dY**: original dataset for decomposition, matrix of n x py.
- **n.comp.X**: the number of non-Gaussian components in dataset X. If null, will estimate the number using ICtest::FOBIasymp.
- **n.comp.Y**: the number of non-Gaussian components in dataset Y. If null, will estimate the number using ICtest::FOBIasymp.
- **df**: default value=0 when use JB, if df>0, estimates a density for the loadings using a tilted Gaussian (non-parametric density estimate).
- **rho.extent**: Controls similarity of the scores in the two datasets. Numerical value and three options in character are acceptable. small, medium or large is defined from the JB statistic. Try "small" and see if the loadings are equal, then try others if needed. If numeric input, it will multiply the input by JBall to get the rho.
- **Cplus**: whether to use C code (faster) in curvilinear search.
- **tol**: difference tolerance in curvilinear search.
- **stand**: whether to use standardization, if true, it will make the column and row means to 0 and columns sd to 1. If false, it will only make the row means to 0.
distribution: "JB" or "tiltedgaussian"; "JB" is much faster. In SING, this refers to the "density" formed from the vector of loadings. "tiltedgaussian" with large df can potentially model more complicated patterns.

maxiter: the max iteration number for the curvilinear search.

individual: whether to return the individual non-Gaussian components, default value = F.

whiten: whitening method used in lngca. Defaults to "svd" which uses the n left eigenvectors divided by sqrt(px-1) by 'eigenvec'. Optionally uses the square root of the n x n "precision" matrix by 'sqrtprec'.

restarts.dbyd: default = 0. These are d x d initial matrices padded with zeros, which results in initializations from the principal subspace. Can speed up convergence but may miss low variance non-Gaussian components.

restarts.pbyd: default = 20. Generates p x d random orthogonal matrices. Use a large number for large datasets. Note: it is recommended that you run lngca twice with different seeds and compare the results, which should be similar when a sufficient number of restarts is used. In practice, stability with large datasets and a large number of components can be challenging.

Value

Function outputs a list including the following:

Sjx: variable loadings for joint NG components in dataset X with matrix rj x px.
Sjy: variable loadings for joint NG components in dataset Y with matrix rj x py.
Six: variable loadings for individual NG components in dataset X with matrix riX x px.
Siy: variable loadings for individual NG components in dataset Y with matrix riX x py.
Mix: scores of individual NG components in X with matrix n x riX.
Miy: scores of individual NG components in Y with matrix n x riY.
est.Mjx: Estimated subject scores for joint components in dataset X with matrix n x rj.
est.Mjy: Estimated subject scores for joint components in dataset Y with matrix n x rj.
est.Mj: Average of est.Mjx and est.Mjy as the subject scores for joint components in both datasets with matrix n x rj.
C_plus: whether to use C version of curvilinear search.
rho_extent: the weight of rho in search

df: degree of freedom, = 0 when use JB, >0 when use tiltedgaussian.

Examples

```r
# get simulation data
data(exampledata)

# use JB stat to compute with singR
output_JB=singR(dX=exampledata$dX,dY=exampledata$dY,
df=0,rho_extent="small",distribution="JB",individual=TRUE)
```
# use tiltedgaussian distribution to compute with singR.
# tiltedgaussian may be more accurate but is considerably slower,
# and is not recommended for large datasets.
output_tilted=singR(dX=exampledata$dX,dY=exampledata$dY,
df=5,rho_extent="small",distribution="tiltedgaussian",individual=TRUE)

# use pmse to measure difference from the truth
pmse(M1 = t(output_JB$est.Mj),M2 = t(exampledata$mj),standardize = TRUE)

pmse(M1 = t(output_tilted$est.Mj),M2 = t(exampledata$mj),standardize = TRUE)

---

**standard**

*Standardization with double centered and column scaling*

**Description**

Standardization with double centered and column scaling

**Usage**

```r
standard(data, dif.tol = 0.001, max.iter = 10)
```

**Arguments**

- **data**
  - input matrix with n x px.
- **dif.tol**
  - the value for the threshold of scaling
- **max.iter**
  - default value = 10

**Value**

standardized matrix with n x px.

**Examples**

```r
spmwm = 3*matrix(rnorm(100000),nrow=100)+1
dim(spmwm)
apply(spmwm,1,mean) # we want these to be 0
apply(spmwm,2,mean) # we want these to be 0
apply(spmwm,2,sd) # we want each of these variances to be 1

spmwm_cp=standard(spmwm)
max(abs(apply(spmwm_cp,1,mean)))
max(abs(apply(spmwm_cp,2,mean)))
max(abs(apply(spmwm_cp,2,sd)-1))
```
theta2W

Convert angle vector into orthodox matrix

Description
Convert angle vector into orthodox matrix

Usage
theta2W(theta)

Arguments
theta vector of angles theta

Value
an orthodox matrix

tiltingaussian

tiltingaussian

Description
tiltingaussian

Usage
tiltingaussian(xData, df = 8, B = 100, ...)

Arguments
xData input data
df degree freedom
B default value=100
... ellipsis
vec2net

Create network matrices from vectorized lower diagonals vec2net.

Description

Create network matrices from vectorized lower diagonals vec2net.

Usage

vec2net(invector, make.diag = 1)

Arguments

invector vectorized lower diagonals.
make.diag default value = 1.

Value

a net matrix

Examples

net = vec2net(1:10)

whitener

Whitening Function

Description

Whitening Function

Usage

whitener(X, n.comp = ncol(X), center.row = FALSE)

Arguments

X dataset p x n.
n.comp the number of components
center.row whether center the row of data

Value

a whitener matrix
Calculate the power of a square matrix

Description
returns a matrix composed of eigenvector x diag(eigenvalue ^ power) x eigenvector’

Usage
S %^% power

Arguments
S a square matrix
power the times of power

Value
a matrix after power calculation that eigenvector x diag(eigenvalue ^ power) x eigenvector’

Examples
a <- matrix(1:9,3,3)
a %^% 2
Index

* datasets
  - exampledata, 8
  - %^%, 22
  - angleMatchICA, 2
  - aveM, 3
  - calculateJB, 4
  - covwhitener, 4
  - create.graph.long, 5
  - curvilinear, 5
  - curvilinear_c, 6
  - est.M.ols, 8
  - exampledata, 8
  - gen.inits, 9
  - greedymatch, 9
  - lngca, 10
  - matchICA, 13
  - NG_number, 13
  - orthogonalize, 14
  - permmatRank_joint, 14
  - permTestJointRank, 15
  - pmse, 15
  - signchange, 16
  - singR, 17
  - standard, 19
  - theta2W, 20
  - tiltedgaussian, 20
  - vec2net, 21
  - whitener, 21