Package 'toscca'

November 7, 2025

Title Thresholded Ordered Sparse CCA

Version 0.1.0

Description
Performs Thresholded Ordered Sparse Canonical Correlation Analysis (CCA). For more details see Senar, N. (2024) <doi:10.1093/bioadv/vbae021> and Senar, N. et al. (2025) <doi:10.48550/arXiv.2503.15140>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests ClusterR, RColorBrewer, cluster, knitr, rmarkdown, testthat (>= 3.0.0)

Imports doParallel, foreach, parallel, forecast, ggplot2, lme4, scales, graphics, stats, utils, grDevices, mcompanion, EnvStats, MASS

Depends R (>= 3.5)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Nuria Senar [aut, cre] (ORCID: <`https://orcid.org/0000-0002-1004-5785`>)

Maintainer Nuria Senar <`n.senarvilladeamigo@amsterdamumc.nl`>

Repository CRAN

Date/Publication 2025-11-07 13:40:02 UTC

Contents

`cpev.toscca` *Calculates cpev*

Description

This function calculates cpev w.r.t. a chosen origin.

Usage

```
cpev.toscca(mat, weights)
```

Arguments

`mat`	A matrix.
`weights`	A numberic vector of canonical weights.

Value

Returns cpev values for the kth or 1:K components.

`fastEigen` *Performs eigen decomposition of a matrix in PS space.*

Description

Performs eigen decomposition of a matrix in PS space.

Usage

```
fastEigen(A)
```

Arguments

`A`	A square matrix nxn.

Value

Matrix. Positive definite matrix.

`getCanSubspace` *Performs matrix residualisation over estimated canonical vectors by using the null space of the estimated canonical vector to construct a new matrix.*

Description

Performs matrix residualisation over estimated canonical vectors by using the null space of the estimated canonical vector to construct a new matrix.

Usage

```
getCanSubspace(mat, vec)
```

Arguments

`mat`	An nxp matrix.
`vec`	A vector of dimensions nxk.

Details

For nxp matrix

$$\mathbf{A}$$

and pxk vector

$$\alpha$$

, the canonical is compute as $\mathbf{A}_{sub} = \mathbf{A}\alpha(\alpha^T\alpha)\alpha^T$.

Value

An nxk matrix.

---

getWhich *Get location of required.*

Description

Get location of required.

Usage

```
getWhich(data, fun)
```

Arguments

`data`	Numeric matrix.
`fun`	Function to search data.

Value

Returns value matching function fun.

Examples

```
getWhich(rnorm(100), max)
```

---

initialiseCanVar *Initialised the canonical vector for the iterative process based on positive eigen values. Then, SVD is performed on that PS matrix.*

Description

Initialised the canonical vector for the iterative process based on positive eigen values. Then, SVD is performed on that PS matrix.

Usage

```
initialiseCanVar(A, B)
```

Arguments

`A`	An nxp matrix.
`B`	An nxq matrix.

Value

An pzp vector.

Examples

```
#sample size etc
N = 10
p = 25
q = 5
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,1)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.3)

for(j in 1:length(beta))
  Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.3)
cv = initialiseCanVar(X0, Y0)
```

`modes` *Calculates mode.*

Description

Calculates mode.

Usage

```
modes(d)
```

Arguments

d	density object.

myHeatmap *Plot heatmap of cv w.r.t. the penalty parameter perfotmance.*

Description

This function plots cca for different thresholds

Usage

```
myHeatmap(
  mat,
  palette_values = mpalette,
  blue = NULL,
  xlab = "",
  ylab = "",
  show_axes = TRUE,
  show_labels = TRUE,
  K = NULL
)
```

Arguments

`mat`	A matrix.
`palette_values`	Character string. Vector of colour values for the heatmap. Default is package's palette.
`blue`	Logical. If TRUE, use only scale of blues from palette.
`xlab`	Character. Label for x axis.
`ylab`	Character. Label for y axis.
`show_axes`	Logic. Default is True.
`show_labels`	Logic. Default is True.
`K`	Numeric. Number of components.

Value

No return value, called for plotting heatmap.

Examples

```
mat <- matrix(rexp(200, rate=.1), ncol=20)/200
myHeatmap(mat)
```

`plt.selstab` *Plot slection stability for penalty parameter performance.*

Description

This function plots cv for different thresholds

Usage

```
plt.selstab(
  object,
  X,
  Y,
  nonz_x,
  nonz_y,
  palette_values = mpalette,
  blue = TRUE,
  mm = TRUE,
  k = 1
)
```

Arguments

`object`	A toscca object.
`X`	nxp matrix. Observation matrix.
`Y`	A nxq matrix. Observation matrix.
`nonz_x`	Numeric vector. Sparsity levels of X.
`nonz_y`	Numeric vector. Sparsity levels of Y.
`palette_values`	Character. Name of a palette for the heatmap. Default is "Teal".
`blue`	Logical. If TRUE, use only scale of blues from palette.
`mm`	Logic. Indicates whether there are multiple measurements or not. Default is True.
`k`	Numeric. Component, default k =1.

Value

No return value, called for selection stability pot.

Examples

```
# example code
#sample size etc
N = 10
p = 25
q = 5
```

```
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,10)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.03)

for(j in 1:length(beta))
  Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.03)

# performa toscca
X = standardVar(X0)
Y = standardVar(Y0)
K = 2                                       # number of components to be estimated
nonz_x = c(2,5, 10, 20)                     # number of nonzero variables for X
nonz_y = c(2, 3, 4)                      # number of nonzero variables for Y
init   = "uniform"                          # type of initialisation
cca_toscca  = toscca(X, Y, nonz_x, nonz_y, K, alpha_init = init,
combination = TRUE, K=1, silent = TRUE, toPlot = FALSE)
plt <- plt.selstab(cca_toscca,X, Y, nonz_x = nonz_x, nonz_y = nonz_y, mm=FALSE)
```

powerMethod *Performs power method.*

Description

Performs power method.

Usage

```
powerMethod(mat, vec, tol = 10^(-6), maxIter = 500, silent = TRUE)
```

Arguments

`mat`	A square matrix nxn.
`vec`	A vector of dimensions nx1.
`tol`	Convergence criterion. Default is 10^(-6).
`maxIter`	Maximum iterations. Default is 500.
`silent`	Logical. If TRUE, convergence performance will be printed.

Value

List: vec: eigen vector; lambda: eigen value; t: total iterations.

`progressBar` *Progress bar*

Description

Shows progress of a process.

Usage

```
progressBar(end, round)
```

Arguments

`end`	maximum number of times a process will run.
`round`	current round

Value

Display in consol of current status.

`residualisation` *Performs matrix residualisation over estimated canonical vectors. There are three types: basic (subtracts scaled estimated latent variable from data), null (uses the null space of the estimated canonical vector to construct a new matrix) and LV (uses SVD to residualise).*

Description

Performs matrix residualisation over estimated canonical vectors. There are three types: basic (subtracts scaled estimated latent variable from data), null (uses the null space of the estimated canonical vector to construct a new matrix) and LV (uses SVD to residualise).

Usage

```
residualisation(
  mat,
  vec,
  spaceMat = NULL,
  type = c("LV", "null", "basic"),
  na.allow = TRUE
)
```

Arguments

`mat`	An nxp matrix.
`vec`	A vector of dimensions nxk.
`spaceMat`	Only for "null" type residualisation. Default is NULL.
`type`	Character. It can be LV, null or basic depending on which type of residualisation will be performed.
`na.allow`	Logical. If TRUE, NAs will be allowed.

Value

Matrix.

`scaledResidualMat` *Performs scalling for matrix residualisation based on calculated coefficients.*

Description

Performs scalling for matrix residualisation based on calculated coefficients.

Usage

```
scaledResidualMat(A)
```

Arguments

`A`	An nxp matrix.

Value

scaled matrix.

scale_rm *Standardises matrices with multiple measurements per individual.*

Description

This function stardadises matrices with multiple measurements w.r.t. a chosen origin.

Usage

```
scale_rm(mat, origin = NULL, centre = FALSE)
```

Arguments

mat	A matrix.
origin	Measurement of reference for stardadisation.
centre	Logical. TRUE to centre data. Default is FALSE.

Value

Returns scaled and/or centred values for repeated measurements.

Examples

```
#sample size etc
N = 10
p = 25
q = 5
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,1)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.3)

for(j in 1:length(beta))
```

```
    Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.3)

  X = standardVar(X0)
  Y = standardVar(Y0)
```

---

standardVar *Stardardise a matrix*

Description

This function stardardises a matrix or a vector and gives the option to centre or normalise (only vectors).

Usage

```
standardVar(mat, centre = TRUE, normalise = FALSE)
```

Arguments

`mat`	Matrix or vector to be standardise.
`centre`	Logical, if true, cetre to mean zero.
`normalise`	Logical, if true, performs vector normalisation.

Value

A matrix or vector with the preferred standardarisation

Examples

```
#sample size etc
N = 10
p = 25
q = 5
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,1)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
```

```
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.3)

for(j in 1:length(beta))
  Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.3)

X = standardVar(X0)
Y = standardVar(Y0)
```

---

toscamm.perm *Computes permutatied cc fot TOSCCA-MM*

Description

This function estimates sparse canonical vectors for permuted matrices with multiple measurements.

Usage

```
toscamm.perm(
  A,
  B,
  nonzero_a,
  nonzero_b,
  K = 1,
  folds = 1,
  toPlot = FALSE,
  draws = 1000,
  cancor,
  bootCCA = NULL,
  silent = TRUE,
  parallel_logic = TRUE,
  nuisanceVar = 0,
  testStatType = "CC",
  model = "lme",
  lmeformula = " ~ 0 + poly(time,3) + (1|id)",
  arformula = NULL,
  ncores = NULL
)
```

Arguments

`A`	A matrix.
`B`	A matrix.
`nonzero_a`	Integer. Threshold parameter for A.

nonzero_b	Integer. Threshold parameter for B.
K	Interger. Numner of components.
folds	Integer. Indicates number of folds to perform.
toPlot	Logical. Indicates if plots will be produced. Default is False.
draws	Integer. Number of draws in permutation.
cancor	Numeric vector of length K with estimated canonical correlations.
bootCCA	deprecated.
silent	Logical. TRUE to keep silent output messages. Default is FALSE.
parallel_logic	Logical. TRUE to parallelise folds. Default is FALSE.
nuisanceVar	Numeric. Number of nuisance variables.
testStatType	Character. Choice of test-statistic c("CC", "Wilks", "Roy"),
model	Character. c("lme", "ar"). Model to fit longitudinal latent space.
lmeformula	Character. LME formula. Default is " ~ -1 + time + (1|id)".
arformula	Numeric vector. Choice of ARIMA. Default is c(1,0,0).
ncores	numeric. Number of cores to use in parallelisation. Default is detectCores() -1.

Value

Permuted canonical correlation for ell K and p-values.

List with permuted correlations and p-values.

Examples

```
# example code

# dont run due to parallel processes
#sample size etc
N = 10
p = 25
q = 5
X0 = list()
Y0 = list()

#Some associations with the true signal
cwa = (6:10) / 10
cwb  = -(2:3) / 10

alpha = rep(0, p)
beta = rep(0, q)

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

alpha[loc_alpha] = cwa
beta[loc_beta] = cwb
```

```
sg = matrix(c(1, 0.6, 0.3, rep(0, 2),
              0.6, 1, 0.6, 0.3, rep(0, 1),
              0.3, 0.6, 1, 0.6, 0.3,
              rep(0,1), 0.3, 0.6, 1, 0.6,
              rep(0,2), 0.3, 0.6, 1), ncol = 5)
for(i in 1:N)
{
  times = 1:5
  Zi1 = (sin(100*times))^times +   times * 0.65 +rnorm(1,0,0.95)
  Zi = cbind(Zi1)
  #Simulate data and add some noise
  X0i = sapply(1:p, function(a) MASS::mvrnorm(1, (Zi %*% t(alpha))[,a], Sigma = sg))
  Y0i = sapply(1:q, function(a) MASS::mvrnorm(1, (Zi %*% t(beta))[,a], Sigma = sg))

  colnames(X0i) = paste0("X", 1:ncol(X0i))
  colnames(Y0i) = paste0("Y", 1:ncol(Y0i))
  #Check the simulated cross correlation
  #image(cor(X0i, Y0i))

  #Remove some observations
  # p_observed = 1
  X0i = cbind(id=i, time=times, X0i)#[rbinom(length(times),1,p_observed)==1,]
  Y0i = cbind(id=i, time=times, Y0i)#[rbinom(length(times),1,p_observed)==1,]

  X0[[i]] = X0i
  Y0[[i]] = Y0i
}

X0 = do.call("rbind", X0)
Y0 = do.call("rbind", Y0)

X = data.frame(X0); Y = data.frame(Y0)
nonz_a = c(2, 5, 10, 20)
nonz_b =  c(2, 3, 4)

mod <- tosccamm(X, Y, folds = 2, nonzero_a = nonz_a, nonzero_b = nonz_b, silent = TRUE)
nza <- mod$nonzero_a
nzb <- mod$nonzero_b
cc  <- mod$cancor
perm_cc <- toscamm.perm(X,Y, nonzero_a=nza, nonzero_b=nzb,cancor=cc, ncores=2, draws = 10)
```

toscca *Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.*

Description

Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.

Usage

```
toscca(
  A,
  B,
  nonzero_a,
  nonzero_b,
  K = 1,
  alpha_init = c("eigen", "random", "uniform"),
  folds = 1,
  silent = FALSE,
  toPlot = TRUE,
  typeResid = "basic",
  combination = FALSE,
  parallel_logic = FALSE
)
```

Arguments

`A, B`	Data matrices.
`nonzero_a, nonzero_b`	Numeric. Scalar or vector over the number of nonzeroes allowed for a correlation estimate.
`K`	Numeric. Number of components to be computed.
`alpha_init`	Character. Type initialisation for $$\alpha$$. Default is "eigen".
`folds`	Numeric. Number of folds for the cross-validation process.
`silent`	Logical. If FALSE, a progress bar will appear on the console. Default is FALSE.
`toPlot`	Logical. If TRUE, plot will be generated automatically showing the estimated canonical weights. Default is TRUE.
`typeResid`	Character. Choice of residualisation technique. Options are basic (default), null and LV.
`combination`	Logical. If TRUE, the algorithm will search for the best combination of sparsity choice nonzero_a and nonzero_b for each component. This should be used for exploratory analysis. Default is FALSE.
`parallel_logic`	Logical. If TRUE, cross-validation is done in parallel.Default is FALSE.

Value

a list with the following elements:

List with estimated toscca parameters.

Examples

```
#sample size etc
N = 10
p = 25
q = 5
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,1)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.3)

for(j in 1:length(beta))
  Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.3)

X = standardVar(X0)
Y = standardVar(Y0)
K = 2                                       # number of components to be estimated
nonz_x = c(2,5, 10, 20)                     # number of nonzero variables for X
nonz_y = c(2, 3, 4)                      # number of nonzero variables for Y
init   = "uniform"                          # type of initialisation
cca_toscca  = toscca(X, Y, nonz_x, nonz_y, K, alpha_init = init, silent = TRUE, toPlot = FALSE)
```

---

toscca.core

Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.

Description

Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.

Usage

```
toscca.core(
  alphaInit,
  A,
  B,
  nonzero_a,
  nonzero_b,
  iter = 20,
  tol = 10^(-6),
  silent = FALSE
)
```

Arguments

`alphaInit`	Character. Type initialisation for α .
`A, B`	Data matrices.
`nonzero_a, nonzero_b`	Numeric. Scalar or vector over the number of nonzeroes allowed for a correlation estimate.
`iter`	Numeric. Maximum number of iterations. Default is 20.
`tol`	Numeric. Tolerance threshold. Default is 10^6.
`silent`	Logical. If FALSE, a progress bar will appear on the console. Default is FALSE.

Value

a list with the following elements:

`toscca.folds` *Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.*

Description

Sparse Canonical Correlation Analysis. Computation of CC via NIPALS with soft thresholding.

Usage

```
toscca.folds(
  A,
  B,
  nonzero_a,
  nonzero_b,
  alpha_init,
  folds = 1,
  parallel_logic = FALSE,
  silent = FALSE,
  toPlot = TRUE,
  ATest_res = NULL,
  BTest_res = NULL
)
```

Arguments

A, B	Data matrices.
nonzero_a, nonzero_b	Numeric. Scalar or vector over the number of nonzeroes allowed for a correlation estimate.
alpha_init	Character. Type initialisation for $$\alpha$$. Default is "eigen".
folds	Integer. Indicates number of folds to perform.
parallel_logic	Logical. TRUE to parallelise folds. Default is FALSE.
silent	Logical. TRUE to keep silent output messages. Default is FALSE.
toPlot	Logical. TRUE to plot results.
ATest_res	NULL. Keep NULL.
BTest_res	NULL. Keep NULL.

Value

a list with the following elements:

---

toscca.lv *Get latent variables*

Description

Gets latent variables from data and estimates.

Usage

```
toscca.lv(data, alpha, beta)
```

Arguments

data	List containint both observation amtrices.
alpha	px1 numeric vector. canonical weights for X.
beta	qx1 numeric vector. canonical weights for Y.

toscca.perm *Permutation testing for toscca*

Description

This function performs permutation testing on CC estimates.

Usage

```
toscca.perm(
  A,
  B,
  nonzero_a,
  nonzero_b,
  K,
  alpha_init = c("eigen", "random", "uniform"),
  folds = 1,
  toPlot = FALSE,
  draws = 20,
  cancor,
  silent = TRUE,
  parallel_logic = TRUE,
  nuisanceVar = 0,
  testStatType = "CC",
  ncores = NULL
)
```

Arguments

`A, B`	Data matrices.
`nonzero_a, nonzero_b`	Numeric. Scalar or vector over the number of nonzeroes allowed for a correlation estimate.
`K`	Numeric. Number of components to be computed.
`alpha_init`	Character. Type initialisation for $$\alpha$$. Default is "eigen".
`folds`	Numeric. Number of folds for the cross-validation process.
`toPlot`	Logical. If TRUE, plot will be generated automatically showing the estimated canonical weights. Default is TRUE.
`draws`	Numeric. Number of permutations for each component.
`cancor`	Numeric. Scalar or vector: anonical correlation estimate(s).
`silent`	Logical. If FALSE, a progress bar will appear on the console. Default is FALSE.
`parallel_logic`	Logical. If TRUE, cross-validation is done in parallel.Default is FALSE.

`nuisanceVar`	Data with nuisance variables. For statistic type.
`testStatType`	Character. Choice of statistic. Options are CC (default), Wilks and Roy.
`ncores`	numeric. Number of cores to use in parallelisation. Default is detectCores() -1.

Details

For a exploratory analysis nonzero_a and nonzero_b can be vectors. The algorithm will then search for the best combination of sparsity choice nonzero_a and nonzero_b for each component.

Value

Matrix with permutation estimates.

List with permuted correlations and p-values.

Examples

```
#sample size etc
N = 10
p = 25
q = 5
# noise
X0 = sapply(1:p, function(x) rnorm(N))
Y0 = sapply(1:q, function(x) rnorm(N))

colnames(X0) = paste0("x", 1:p)
colnames(Y0) = paste0("y", 1:q)

# signal
Z1 = rnorm(N,0,1)


#Some associations with the true signal
alpha = (6:10) / 10
beta  = -(2:3) / 10

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

for(j in 1:length(alpha))
  X0[, loc_alpha[j]] =  alpha[j] * Z1 + rnorm(N,0,0.3)

for(j in 1:length(beta))
  Y0[, loc_beta[j]] =  beta[j] * Z1 + rnorm(N,0,0.3)

X = standardVar(X0)
Y = standardVar(Y0)
K = 2                                    # number of components to be estimated
nonz_x = c(2,5, 10, 20)                  # number of nonzero variables for X
nonz_y = c(1, 2, 3, 4)                   # number of nonzero variables for Y
init   = "uniform"                       # type of initialisation
cca_toscca  = toscca(X, Y, nonz_x, nonz_y, K, alpha_init = init, silent = TRUE, toPlot = FALSE)
```

```
#dont run due to parallelisation.
cc = cca_toscca$cancor
perm_toscca = toscca.perm(X, Y, nonz_x, nonz_y, K = K, init, draws = 10, cancor = cc, ncores = 2)
```

toscca.tStat *Get the estatistic for the permutations.*

Description

Get the estatistic for the permutations.

Usage

```
toscca.tStat(cancor, A, B, C = 0, type = c("CC", "Wilks", "Roy"))
```

Arguments

cancor	Numeric. Canonical Correlation estimate.
A	An nxp matrix.
B	An nxq matrix.
C	An nxs matrix. Confounding variables.
type	Character. Choice of statistic: Canonical correlation, Wilks'statistic or Roy's statistic.

Value

Statistic

tosccamm *Computes TOSCCA-MM*

Description

This function estimates sparse canonical vectors for matrices with multiple measurements and the trajectories of the latent variables.

Usage

```
tosccamm(
  A,
  B,
  nonzero_a,
  nonzero_b,
  folds = 1,
  parallel_logic = FALSE,
  silent = FALSE,
  ATest_res = NULL,
  BTest_res = NULL,
  model = "lme",
  lmeformula = " ~ -1 + time + (1|id)",
  arformula = c(1, 0, 0)
)
```

Arguments

`A`	A data.frame with id and time as first two columns.
`B`	A data.frame with id and time as first two columns.
`nonzero_a`	Integer. Threshold parameter for A.
`nonzero_b`	Integer. Threshold parameter for B.
`folds`	Integer. Indicates number of folds to perform.
`parallel_logic`	Logical. TRUE to parallelise folds. Default is FALSE.
`silent`	Logical. TRUE to keep silent output messages. Default is FALSE.
`ATest_res`	NULL. Keep NULL.
`BTest_res`	NULL. Keep NULL.
`model`	Character. c("lme", "ar"). Model to fit longitudinal latent space.
`lmeformula`	Character. LME formula. Default is " ~ -1 + time + (1|id)".
`arformula`	Numeric vector. Choice of ARIMA. Default is c(1,0,0).

Value

Canonical vectors for k components.

List with estimated tosccamm parameters.

Examples

```
# example code
#sample size etc
N = 10
p = 25
q = 5
X0 = list()
Y0 = list()
```

```
#Some associations with the true signal
cwa = (6:10) / 10
cwb  = -(2:3) / 10

alpha = rep(0, p)
beta = rep(0, q)

loc_alpha = 1:length(alpha)
loc_beta  = 1:length(beta)

alpha[loc_alpha] = cwa
beta[loc_beta] = cwb

sg = matrix(c(1, 0.6, 0.3, rep(0, 2),
              0.6, 1, 0.6, 0.3, rep(0, 1),
              0.3, 0.6, 1, 0.6, 0.3,
              rep(0,1), 0.3, 0.6, 1, 0.6,
              rep(0,2), 0.3, 0.6, 1), ncol = 5)
for(i in 1:N)
{
  times = 1:5
  Zi1 = (sin(100*times))^times +   times * 0.65 +rnorm(1,0,0.95)
  Zi = cbind(Zi1)
  #Simulate data and add some noise
  X0i = sapply(1:p, function(a) MASS::mvrnorm(1, (Zi %*% t(alpha))[,a], Sigma = sg))
  Y0i = sapply(1:q, function(a) MASS::mvrnorm(1, (Zi %*% t(beta))[,a], Sigma = sg))

  colnames(X0i) = paste0("X", 1:ncol(X0i))
  colnames(Y0i) = paste0("Y", 1:ncol(Y0i))
  #Check the simulated cross correlation
  #image(cor(X0i, Y0i))

  #Remove some observations
  # p_observed = 1
  X0i = cbind(id=i, time=times, X0i)#[rbinom(length(times),1,p_observed)==1,]
  Y0i = cbind(id=i, time=times, Y0i)#[rbinom(length(times),1,p_observed)==1,]

  X0[[i]] = X0i
  Y0[[i]] = Y0i
}

X0 = do.call("rbind", X0)
Y0 = do.call("rbind", Y0)

X = data.frame(X0); Y = data.frame(Y0)
nonz_a = c(2, 5, 10, 20)
nonz_b =  c(2, 3, 4)

mod <- tosccamm(X, Y, folds = 2, nonzero_a = nonz_a, nonzero_b = nonz_b, silent = TRUE)
```

Index