Package ‘timeFA’

January 22, 2021

Type Package

Title Factor Models and Autoregressive Models for Time Series

Version 0.1.0

Author@R person(``Zebang", ``Li", email = ``zebang.li@rutgers.edu",
role = c(``aut", ``cre"))

Description Factor and Autoregressive Models for Tensor Time Series

License GPL (>= 2)

Encoding UTF-8

LazyData true

Imports tensor,
rTensor,
MASS,
abind,
igraph

RoxygenNote 7.1.1

VignetteBuilder knitr

R topics documented:

dynamic_A 2
em 3
generate 4
generateA 4
get.hat 5
grouping.loading 5
iter.tensor.bic 6
iter.tensor.ratio 7
MAR 8
MAR.SE 8
MAR1.LS 9
MAR1.otimes 10
MAR1.projection 11
MAR2.LS 11
MAR2.otimes 12
MAR2.projection 13
matrix_factor 13

mfmda 14
mfmda.estqk 15
mfmda.na.iter 15
mfmda.na.vec 16
mfmda.nona.iter 16
mfmda.nona.noniter 17
mfmda.nona.vec 18
PlotNetwork_AB 18
pm 19
projection 20
rearrange 20
run.test 21
TAR 21
TAR1.LS 22
TAR1.MLE 23
TAR1.projection 23
TAR1.SE.LSE 24
TAR1.SE.MLE 24
TAR1.VAR 25
TAR2.LS 25
TAR2.MLE 26
TAR2.projection 27
TAR2.SE.LSE 27
TAR2.SE.MLE 28
tensor.bic 28
tensor.ratio 29
tipup.init.tensor 29
topup.init.tensor 30
trearrange 30
up.init.tensor 31
var1 31
vector_factor 32
vts 32

Index **34**

`dynamic_A` *Get the adjacency matrix for plotting*

Description

Get the adjacency matrix for plotting

Usage

```
dynamic_A(x, factor_count, simple.flag, threshold)
```

Arguments

`x`	input the original estimated loading matrix
`factor_count`	The number of factors to use
`simple.flag`	if True, only eliminate the entries below threshold and make all row sums to be 1; if False, the approach further eliminates the entries of the rows that are very close to threshold value and only leaves the maximum entry of each row
`threshold`	A parameter to eliminate very small entries of the loading matrix

Value

The new loading matrix with all rows sum to be 1

`em` *Permutation matrix em*

Description

Permutation matrix em.

Usage

```
em(m, n, i, j)
```

Arguments

`m, n, i, j`	set $m \times n$ zero matrix with $A_{ij} = 1$

Value

Permutation matrix em such that $A_{ij} = 1$ and other entries equals 0.

See Also

`trearrange`

Examples

```
A = em(3,3,1,1)
```

generate — *Generate an AR(1) tensor time series with given coefficient matrices*

Description

For test only, with given coefficient matrices and time length t, generate an AR(1) tensor time series.

Usage

```
generate(A, t, setting = "iid")
```

Arguments

`t`	length of time
`setting`	the structure of random error, can be specified as "iid", "mle" and "svd", default value is "iid".
`dim`	an array of dimensions of matrices or tensors

Value

a list containing the following:

`A` matrices $A_1, A_2, \cdots, A_k$

`X` AR(1) tensor time series

See Also

`run.test`

Examples

```
A <- generateA(dim=c(2,2,2), R=1)
xx <- generate(c(m1,m2,m3), T=100)
```

generateA — *Generate coefficient matrices*

Description

For test only, to generate coefficient matrices

Usage

```
generateA(dim, R)
```

Arguments

`dim`	an array of dimensions of matrices or tensors
`R`	number of terms

Value

a list containing coefficient matrices

Examples

```
A <- generateA(dim=c(2,2,2), R=1)
xx <- generate(c(m1,m2,m3), T=100)
```

---

get.hat *functions for algorithm*

Description

functions for algorithm

Usage

```
get.hat(x, Q, d.seq)
```

Arguments

x	tensor of any dimension, x: $d1 * d2 * d3 * \cdots * d_k * n$
Q	list of k matrices
d.seq	1:k

Value

a tensor object x.hat

---

grouping.loading *Get the clustering of loading matrix*

Description

Get the clustering of loading matrix

Usage

```
grouping.loading(loading, ncluster, rowname, plot = T)
```

Arguments

loading	The estimated loading matrix
ncluster	The number of clusters to use, usually the dimension of the factor matrix
rowname	The name of the rows
plot	plot the clustering graph, defacult True

Value

Loading matrix after grouping

`iter.tensor.bic` *iterative versions of all three methods for any Dim tensor time series*

Description

iterative versions of all three methods for any Dim tensor time series

Usage

```
iter.tensor.bic()
```

Arguments

`x`	tensor of any dimension, $x : d1 * d2 * d3 * \cdots * d_K * n$
`r`	initial guess of # of factors
`h0`	Pre-scribed parameter h
`method`	the method chosen among UP,TIPUP,TOPUP
`tol`	level of error tolerance
`niter`	number of iterations
`tracetrue`	if TRUE, record the dis value for each iteration

Value

a list containing the following:

`Q` .

`Qinit` initial value of Q

`Qfirst` value of Q at first iteration

`x.hat` .

`x.hat.init` initial value of xhat

`x.hat.first` value of xhat at first iteration

`factor.num` number of factors

`timer` a timer

`norm.percent` x after standardization

`dis` difference of fnorm.resid for each iteration

`niter` number of interations

`fnorm.resid` normalized residual

`iter.tensor.ratio` *iterative versions of all three methods for any Dim tensor time series*

Description

iterative versions of all three methods for any Dim tensor time series

Usage

```
iter.tensor.ratio()
```

Arguments

`x`	tensor of any dimension , $d1 * d2 * d3 * \cdots * d_K * n$
`r`	initial guess of # of factors
`h0`	Pre-scribed parameter h
`method`	the method chosen among UP,TIPUP,TOPUP
`tol`	level of error tolerance
`niter`	number of interations
`tracetrue`	if TRUE, record the dis value for each iteration

Value

a list containing the following:

`Q` .

`Qinit` initial value of Q

`Qfirst` value of Q at first iteration

`x.hat` .

`x.hat.init` initial value of xhat

`x.hat.first` value of xhat at first iteration

`factor.num` number of factors

`timer` a timer

`norm.percent` x after standardization

`dis` difference of fnorm.resid for each iteration

`niter` number of interations

`fnorm.resid` normalized residual

`MAR` *Matrix Method*

Description

Estimation function for matrix-valued time series, including the projection method, the Iterated least squares method and MLE under a structured covariance tensor, as determined by the value of `type`.

Usage

```
MAR(xx, type = c("projection", "LS", "MLE", "ar"))
```

Arguments

`xx`	T * p * q matrix-valued time series
`type`	character string, specifying the type of the estimation method to be used. `"projection",` Projection method. `"LS",` Iterated least squares. `"MLE",` MLE under a structured covariance tensor. `"ar",` Stacked vector AR(1) Model.

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`MAR.SE` *Asymptotic Covariance Matrix of* `MAR1.otimes`

Description

Asymptotic covariance Matrix of `MAR1.otimes` for given A, B and matrix-valued time series xx, see Theory 3 in paper.

Usage

```
MAR.SE(xx, B, A, Sigma)
```

Arguments

`xx`	T * p * q matrix-valued time series
`B`	q by q matrix in MAR(1) model
`A`	p by p matrix in MAR(1) model
`Sig`	covariance matrix cov(vec(E_t)) in MAR(1) model

Value

asymptotic covariance matrix

Examples

```
# given T * p * q time series xx
out2=MAR1.LS(xx)
FnormLL=sqrt(sum(out2$LL))
xdim=p;ydim=q
out2Xi=MAR.SE(xx.nm,out2$RR*FnormLL,out2$LL/FnormLL,out2$Sig)
out2SE=sqrt(diag(out2Xi))
SE.A=matrix(out2SE[1:xdim^2],nrow=xdim)
SE.B=t(matrix(out2SE[-(1:xdim^2)],nrow=ydim))
```

`MAR1.LS` *Least Squares Iterative Estimation for Matrix Time Series*

Description

Iterated least squares estimation in the model $X_t = LL * X_{t-1} * RR + E_t$.

Usage

```
MAR1.LS(xx, niter = 50, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	T * p * q matrix-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	printe LL and RR

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`dis` Frobenius norm difference of last update

`niter` number of iterations

`MAR1.otimes` *MLE under a structured covariance tensor*

Description

MAR(1) iterative estimation with Kronecker covariance structure: $X_t = LL * X_{t-1} * RR + E_t$ such that $\Sigma = cov(vec(E_t)) = \Sigma_r \otimes \Sigma_l$.

Usage

```
MAR1.otimes(
  xx,
  LL.init = NULL,
  Sigl.init = NULL,
  Sigr.init = NULL,
  niter = 50,
  tol = 1e-06,
  print.true = FALSE
)
```

Arguments

`xx`	T * p * q matrix-valued time series
`LL.init`	initial value of LL
`Sigl.init`	initial value of Sigl
`Sigr.init`	initial value of Sigr
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	print LL and RR

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`Sigl` one part of structured covariance matrix $\Sigma = \Sigma_r \otimes \Sigma_l$

`Sigr` one part of structured covariance matrix $\Sigma = \Sigma_r \otimes \Sigma_l$

`dis` Frobenius norm difference of the final update step

`niter` number of iterations

`MAR1.projection` *Projection Method*

Description

MAR(1) one step projection estimation in the model $X_t = LL * X_{t-1} * RR + E_t$.

Usage

```
MAR1.projection(xx)
```

Arguments

`xx`	T * p * q matrix-valued time series

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`MAR2.LS` *Least Squares Iterative Estimation for Matrix Time Series with Multiple terms*

Description

Iterated least squares estimation in the model $X_t = \sum_{r=1}^{R} LL_r * X_{t-1} * RR_r + E_t$.

Usage

```
MAR2.LS(xx, r, niter = 80, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	T * p * q matrix-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	printe LL and RR

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`dis` Frobenius norm difference of last update

`niter` number of iterations

`MAR2.otimes` *MLE under a structured covariance tensor for Matrix Time Series with Multiple Terms*

Description

MAR(1) iterative estimation with Kronecker covariance structure: $X_t = \sum_{r=1}^{R} LL_r * X_{t-1} * RR_r + E_t$ such that $\Sigma = cov(vec(E_t)) = \Sigma_r \otimes \Sigma_l$.

Usage

```
MAR2.otimes(xx, r, niter = 200, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	T * p * q matrix-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	print LL and RR
`LL.init`	initial value of LL
`Sigl.init`	initial value of Sigl
`Sigr.init`	initial value of Sigr

Value

a list containing the following:

`LL` estimator of LL, a p by p matrix

`RR` estimator of RR, a q by q matrix

`res` residual of the MAR(1)

`SIGMA` structured covariance matrix $\Sigma = \Sigma_1, \cdots, \Sigma_R$

`dis` Frobenius norm difference of the final update step

`niter` number of iterations

`MAR2.projection` *Projection Method for matrix time series with multiple terms*

Description

MAR(1) one step projection estimation in the model $X_t = \sum_{r=1}^R LL_r * X_{t-1} * RR_r + E_t$.

Usage

```
MAR2.projection(xx, r)
```

Arguments

`xx`	T * p * q matrix-valued time series

Value

a list containing the estimated coefficient matrices

`matrix_factor` *The main estimation function*

Description

The main estimation function

Usage

```
matrix_factor(Yt, inputk1, inputk2, iscentering = 1, hzero = 1)
```

Arguments

`Yt`	Time Series data for a matrix
`inputk1`	The pre-determined row dimension of the factor matrix
`inputk2`	The pre-determined column dimension of the factor matrix
`iscentering`	The data is subtracted by its mean value
`hzero`	Pre-determined parameter h_0

Value

a list containing the following:

`eigval1` estimated row dimension of the factor matrix

`eigval2` estimated column dimension of the factor matrix

`loading1` estimated left loading matrix

`loading2` estimated right loading matrix

`Ft` Estimated factor matrix with pre-determined number of dimensions

`Ft.all` Sum of Ft

`Et` The estimated residual, by subtracting estimated signal term from the data

Examples

```
A <- runif(2000)
dim(A) <- c(20,10,10)
out = matrix_factor(A,3,3)
eig1 = out$eigval1
loading1 = out$loading1
Ft = out$Ft.all
```

---

mfmda *This is a wrapper for all approaches*

Description

This is a wrapper for all approaches

Usage

```
mfmda(Yt, approach = "3", hzero = 1, iscentering = 1)
```

Arguments

approach	Select estimation approaches, 1 for noniterative approach with no NaNs, 2 for iterative approach with NaNs, 3 for iterative approach allowing NaNs.
hzero	Pre-determined parameter
iscentering	The data is subtracted by its mean value
Yc	Time Series data for a matrix

Value

The sample version of M matrix

See Also

matrix_factor

Examples

```
#Use iterative method with $h_0=1$
A <- runif(2000)
dim(A) <- c(20,10,10)
M <- mfmda(A,"3",1,0)
```

`mfmda.estqk` *Compute the estimated number of factors and the corresponding eigen-space*

Description

Compute the estimated number of factors and the corresponding eigen-space

Usage

```
mfmda.estqk(Mhat, inputk = 1)
```

Arguments

`Mhat`	The estimated value for matrix M
`inputk`	The pre-determined number of dimension of factor matrix

Value

The estimated number of factors to use, the corresponding estimated Q matrix, the eigenvalue, the estimated Q matrix with requested number of factors

See Also

`matrix_factor`

Examples

```
#A 10*10 Matrix time series example with t=20 time points
A <- runif(2000)
dim(A) <- c(20,10,10)
M <- mfmda(A,"3",1,0)
inputk <- 3
eig.ans <- mfmda.estqk(M,inputk)
khat <- eig.ans$estk
Qhat <- eig.ans$Qhatestk
eigval <- eig.ans$eigval
Qhatinputk <- eig.ans$Qhatinputk
```

`mfmda.na.iter` *The input data could have NaNs. The estimation approach is iterative.*

Description

The input data could have NaNs. The estimation approach is iterative.

Usage

```
mfmda.na.iter(Yc, hzero)
```

Arguments

`Yc`	Time Series data for a matrix allowing NaNs
`hzero`	Pre-determined parameter

Value

The sample version of M matrix

See Also

`mfmda`

`mfmda.na.vec` *This approach is for the vector-valued estimation with NaNs.*

Description

This approach is for the vector-valued estimation with NaNs.

Usage

```
mfmda.na.vec(Yc, hzero)
```

Arguments

`Yc`	Time Series data for a matrix(dimensions n*p*q), allowing NA input
`hzero`	Pre-scribed parameter h_0

Value

The sample version of M matrix

See Also

`vector_factor`

`mfmda.nona.iter` *The input data do not have zeros. The estimation approach is iterative.*

Description

The input data do not have zeros. The estimation approach is iterative.

Usage

```
mfmda.nona.iter(Yc, hzero)
```

Arguments

`Yc`	Time Series data for a matrix(dimensions n*p*q), no NA input allowed
`hzero`	Pre-scribed parameter

Value

The sample version of M matrix

See Also

`mfmda`

`mfmda.nona.noniter` *The input data do not have zeros. The estimation approach is noniterative.*

Description

The input data do not have zeros. The estimation approach is noniterative.

Usage

```
mfmda.nona.noniter(Yc, hzero)
```

Arguments

`Yc`	Time Series data for a matrix(dimensions n*p*q), no NA input allowed
`hzero`	Pre-scribed parameter h_0

Value

The sample version of M matrix

See Also

`mfmda`

mfmda.nona.vec *This approach is for the vector-valued estimation WITHOUT NaNs.*

Description

This approach is for the vector-valued estimation WITHOUT NaNs.

Usage

```
mfmda.nona.vec(Yc, hzero)
```

Arguments

Yc	Time Series data for a matrix(dimensions n*p*q), no NA input allowed
hzero	Pre-scribed parameter h_0

Value

The sample version of M matrix

See Also

vector_factor

Examples

```
A <- runif(2000)
dim(A) <- c(20,10,10)
M <- mfmda.nona.vec(A,2)
```

PlotNetwork_AB *Plot the network graph*

Description

Plot the network graph

Usage

```
PlotNetwork_AB(Ft, iterated_A, iterated_B, labels)
```

Arguments

Ft	The estimated factor matrix
iterated_A	The left loading matrix
iterated_B	The right loading matrix
labels	The row labels

Value

Plot the network graph

Examples

```
A <- runif(2000)
dim(A) <- c(20,10,10)
out = matrix_factor(A,3,3)
eig1 = out$eigval1
loading1 = out$loading1
loading2 = out$loading2
Ft = out$Ft.all
iterated_A = dynamic_A(loading1,3,F,0.1)
iterated_B = dynamic_A(loading2,3,F,0.1)
labels = 1:10
PlotNetwork_AB(Ft,iterated_A,iterated_A,labels)
```

---

pm *Permutation matrix pm*

Description

Permutation matrix pm.

Usage

```
pm(m, n)
```

Arguments

m	an array of dimensions of matrices $A_1, A_2, \cdots, A_k$
n	length of time

Value

Permutation matrix pm

See Also

trearrange

Examples

```
P = pm(3,3)
```

`projection` *Kronecker Product Approximation*

Description

Kronecker product approximation used in Projection Method of matrix-value time series.

Usage

```
projection(A, r, m1, m2, n1, n2)
```

Arguments

A	m by n matrix such that $m = m1 * n1$ and $n = m2 * n2$
r	number of terms
m1	ncol of A
m2	ncol of B
n1	nrow of A
n2	nrow of B

Value

a list contaning two estimator (matrix)

See Also

MAR1.projection

Examples

```
A <- matrix(runif(6),ncol=2),
projection(A,3,3,2,2)
```

`rearrange` *Rearrangement Operator*

Description

Rearrangement Operator used for projection method.

Usage

```
rearrange(A, m1, m2, n1, n2)
```

Arguments

A	m by n matrix such that $m = m1 * n1$ and $n = m2 * n2$
m1	ncol of A
m2	ncol of B
n1	nrow of A
n2	nrow of B

Value

rearengement matrix

See Also

MAR1.projection

Examples

```
A <- matrix(runif(6),ncol=2),
B <- matrix(runif(6),ncol=2),
M <- kronecker(B,A)
rearrange(M,3,3,2,2) == t(as.vector(A)) %*% as.vector(B)
'TRUE'
```

---

run.test *Test Function*

Description

For test only

Usage

```
run.test(m1, m2, m3, n = 100, T)
```

TAR *Tensor Method*

Description

Estimation function for tensor-valued time series. Projection method, the Iterated least squares method, MLE under a structured covariance tensor and stacked vector AR(1) model, as determined by the value of type.

Usage

```
TAR(xx, type = c("projection", "LS", "MLE", "ar"))
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`method`	character string, specifying the type of the estimation method to be used. `"projection",` Projection method. `"lse",` Iterated least squares. `"mle",` MLE under a structured covariance tensor. `"ar",` Stacked vector AR(1) Model.
`dim`	dimension of coefficient matrices

Value

a list containing the following:

`A` estimator of coefficient matrices $A_1, A_2, \cdots, A_K$

`res` residual of the model

`Sig` covariance matrix cov(vec(E_t))

`niter` number of iterations

`TAR1.LS` *Least Squares Iterative Estimation for Tensor-Valued Time Series*

Description

Iterated least squares estimation in the model $X_t = X_{t-1} \times A_1 \times \cdots \times A_K + E_t$.

Usage

```
TAR1.LS(xx, r = 1, niter = 80, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	printe A_i

Value

a list containing the following:

`A` estimator of coefficient matrices $A_1, A_2, \cdots, A_K$

`res` residual of the model

`Sig` covariance matrix cov(vec(E_t))

`niter` number of iterations

`TAR1.MLE`	*MLE for Tensor-Valued Time Series with One Term Model Under a Structured Covariance Tensor*

Description

MLE for the model $X_t = X_{t-1} \times A_1 \times \cdots \times A_K + E_t$.

Usage

```
TAR1.MLE(xx, r = 1, niter = 80, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	printe A_i

Value

a list containing the following:

`A` estimator of coeficient matrices $A_1, A_2, \cdots, A_K$

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`niter` number of iterations

`TAR1.projection`	*Projection Method for Tensor-Valued Time Series*

Description

TAR(1) one step projection estimation in the model $X_t = X_{t-1} \times A_1 \times \cdots \times A_K + E_t$.

Usage

```
TAR1.projection(xx)
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`m1`	dim(A1)
`m2`	dim(A2)
`m3`	dim(A3)

Value

a list containing the estimation of matrices $A_1, A_2, \cdots, A_K$

TAR1.SE.LSE *Asymptotic Covariance Matrix of* TAR1.LS

Description

Asymptotic covariance Matrix of TAR1.LS for given A tensor-valued time series xx, see related Theorems in our paper.

Usage

```
TAR1.SE.LSE(xx, A.true, Sigma)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series
A.true	coefficient matrices in TAR(1) model
Sigma	covariance matrix $cov(vec(E_t))$ in TAR(1) model

Value

asymptotic covariance matrix

Examples

```
dim <- c(2,2,2)
A <- generateA(dim, R=1)
xx <- generate(dim, T=100)
SIGMA <- TAR1.SE.LSE(xx, A, Sigma=diag(prod(dim)))
```

TAR1.SE.MLE *Asymptotic Covariance Matrix of* TAR1.MLE

Description

Asymptotic covariance Matrix of TAR1.MLE for given A tensor-valued time series xx, see related Theorems in our paper.

Usage

```
TAR1.SE.MLE(xx, A.true, Sigma)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series
A.true	coefficient matrices in TAR(1) model
Sigma	covariance matrix cov(vec(E_t)) in TAR(1) model

Value

asmptotic covariance matrix

Examples

```
dim <- c(2,2,2)
A <- generateA(dim, R=1)
xx <- generate(dim, T=100)
SIGMA <- TAR2.SE.MLE(xx, A, Sigma=diag(prod(dim)))
```

TAR1.VAR *Stacked vector AR(1) Model for Tensor-Valued Time Series*

Description

vector AR(1) Model.

Usage

```
TAR1.VAR(xx)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series

Value

a list containing the following:

`coef` coeficient of the fitted VAR(1) model

`res` residual of the VAR(1) model

Examples

```
A <- generateA(dim=c(2,2,2), R=1)
xx <- generate(c(m1,m2,m3), T=100)
SIGMA <- TAR1.VAR(xx)
```

TAR2.LS *Least Squares Iterative Estimation for Tensor-Valued Time Series with Multiple Terms*

Description

Iterated least squares estimation in the model $X_t = \sum_{r=1}^{R} X_{t-1} \times A_1^{(r)} \times \cdots \times A_K^{(r)} + E_t$.

Usage

```
TAR2.LS(xx, r, niter = 80, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	print A_i

Value

a list containing the following:

`A` estimator of coefficient matrices $A_1^{(1)}, A_2^{(1)}, \cdots, A_K^{(R)}$

`res` residual of the model

`Sig` covariance matrix cov(vec(E_t))

`niter` number of iterations

`TAR2.MLE` *MLE for Tensor-Valued Time Series with Multiple Terms Model Under a Structured Covariance Tensor*

Description

MLE for the model $X_t = \sum_{r=1}^{R} X_{t-1} \times A_1^{(r)} \times \cdots \times A_K^{(r)} + E_t$.

Usage

```
TAR2.MLE(xx, r, niter = 200, tol = 1e-06, print.true = FALSE)
```

Arguments

`xx`	$T * m_1 * \cdots * m_K$ tensor-valued time series
`niter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance
`print.true`	print A_i

Value

a list containing the following:

`A` estimator of coefficient matrices $A_1, A_2, \cdots, A_K$

`res` residual of the MAR(1)

`Sig` covariance matrix cov(vec(E_t))

`niter` number of iterations

TAR2.projection — *Projection Method for Tensor-Valued Time Series with Multiple Terms*

Description

TAR(1) one step projection estimation in the model $X_t = \sum_{r=1}^{R} X_{t-1} \times A_1^{(r)} \times \cdots \times A_K^{(r)} + E_t$.

Usage

```
TAR2.projection(xx, r)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series
r	number of terms

Value

a list containing the estimation of matrices $A_1^{(1)}, A_2^{(1)}, \cdots, A_K^{(R)}$

TAR2.SE.LSE — *Asymptotic Covariance Matrix of* `TAR2.LSE`

Description

Asymptotic covariance Matrix of `TAR2.LSE` for given A tensor-valued time series xx, see related Theorems in our paper.

Usage

```
TAR2.SE.LSE(xx, A.true, Sigma)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series
A.true	coefficient matrices in TAR(1) model
Sigma	covariance matrix cov(vec(E_t)) in TAR(1) model

Value

asmptotic covariance matrix

Examples

```
dim <- c(2,2,2)
A <- generateA(dim, R=1)
xx <- generate(dim, T=100)
SIGMA <- TAR2.SE.LSE(xx, A, Sigma=diag(prod(dim)))
```

TAR2.SE.MLE — *Asymptotic Covariance Matrix of* TAR2.MLE

Description

Asymptotic covariance Matrix of TAR2.MLE for given A tensor-valued time series xx, see related Theorems in our paper.

Usage

```
TAR2.SE.MLE(xx, A.true, Sigma)
```

Arguments

xx	$T * m_1 * \cdots * m_K$ tensor-valued time series
A.true	coefficient matrices in TAR(1) model
Sigma	covariance matrix cov(vec(E_t)) in TAR(1) model

Value

asmptotic covariance matrix

Examples

```
dim <- c(2,2,2)
A <- generateA(dim, R=1)
xx <- generate(dim, T=100)
SIGMA <- TAR2.SE.MLE(xx, A, Sigma=diag(prod(dim)))
```

tensor.bic — *BIC estimators for determing the numbers of factors*

Description

BIC estimators for determing the numbers of factors

Usage

```
tensor.bic(reigen, h0 = 1, p1, p2, n)
```

Arguments

reigen	list of eigenvalues
h0	Pre-scribed parameter h
p1	p1
p2	p2
n	n

Value

factor.p1: Estimated number of factors

`tensor.ratio` *Eigenvalue ratio estimators for determining the numbers of factors*

Description

Eigenvalue ratio estimators for determining the numbers of factors

Usage

```
tensor.ratio(reigen, h0 = 1, p1, p2, n)
```

Arguments

`reigen`	list of eigenvalues
`h0`	Pre-scribed parameter h
`p1`	p1
`p2`	p2
`n`	n

Value

factor.p1: Estimated number of factors

`tipup.init.tensor` *TIPUP one step for any Dim tensor time series*

Description

TIPUP one step for any Dim tensor time series

Usage

```
tipup.init.tensor(x, r, h0 = 1, oneside.true = FALSE, norm.true = FALSE)
```

Arguments

`x`	tensor of any dimension $d1 * d2 * d3 * \cdots * d_d * n$
`r`	initial guess of # of factors
`h0`	Pre-scribed parameter h
`oneside.true`	If oneside.true==TRUE, then only compute the one sided column space, not the other sides, this option is useful for the iterative method
`norm.true`	If norm.true==TRUE, normalize the tensor

Value

a list containing the following:

`M` Estimator

`Q` Orthonormal matrix Q

`lambda` singular values

`norm.percent` x after standardization

`x.hat` A tensor object x.hat

`topup.init.tensor` *TOPUP one step for any Dim tensor time series*

Description

TOPUP one step for any Dim tensor time series

Usage

```
topup.init.tensor(x, r, h0 = 1, oneside.true = FALSE, norm.true = FALSE)
```

Arguments

`x`	$d1 * d2 * d3 * \cdots * d_d * n$
`r`	initial guess of # of factors
`h0`	Pre-scribed parameter h
`oneside.true`	If oneside.true==TRUE, then only compute the one sided column space, not the other sides, this option is useful for the iterative method
`norm.true`	If norm.true==TRUE, calculate the normalized residual of the tensor

Value

a list containing the following:

`M` Estimator

`Q` Orthonormal matrix Q

`lambda` singular values

`norm.percent` normalized residual

`x.hat` A tensor object x.hat

`trearrange` *(alpha version) rearrangement operator for tensor.*

Description

(alpha version) rearrangement operator for tensor.

Usage

```
trearrange(A, m1, m2, m3, n1, n2, n3)
```

`up.init.tensor` *UP one step for any Dim tensor time series*

Description

UP one step for any Dim tensor time series

Usage

```
up.init.tensor(x, r, oneside.true = FALSE, norm.true = FALSE)
```

Arguments

`x`	tensor of any dimension : $d1 * d2 * d3 * \cdots * d_k * n$
`r`	initial guess of # of factors
`oneside.true`	If oneside.true==TRUE, then only compute the one sided column space, not the other sides, this option is useful for the iterative method
`norm.true`	If norm.true==TRUE, normalize the tensor

Value

a list containing the following:

`Q` Orthonormal matrix Q

`lambda` Singular values

`norm.percent` x after standardization

`x.hat` A tensor object x.hat

`var1` *Stacked vector AR(1) Model*

Description

vector AR(1) Model.

Usage

```
var1(xx)
```

Arguments

`xx`	T * p * q matrix-valued time series

Value

a list containing the following:

`coef` coeficient of the fitted VAR(1) model

`res` residual of the VAR(1) model

Examples

```
out.var1=var1(xx)
sum(out.var1$res**2)
```

---

vector_factor *The main estimation function, vector version*

Description

The main estimation function, vector version

Usage

```
vector_factor(Yt, inputk.vec, iscentering = 1, hzero = 1)
```

Arguments

Yt	Time Series data for a matrix
inputk.vec	The pre-determined dimensions of the factor matrix in vector
iscentering	The data is subtracted by its mean value
hzero	Pre-determined parameter h_0

Value

a list containing the following:

`eigval1` estimated dimensions of the factor matrix

`loading` estimated loading matrices

`Ft` Estimated factor matrix with pre-determined number of dimensions

`Ft.all` Sum of Ft

`Et` The estimated random term, by subtracting estimated signal term from the data

---

vts *vector time series*

Description

vector time series

Usage

```
vts(x, h0, r)
```

Arguments

x	a n*d matrix
h0	Pre-scribed parameter h
r	First r eigenvectors

Value

a list containing the following:

`M` .

`Q` The eigenvectors of matrix M

Index

dynamic_A, 2

em, 3

generate, 4
generateA, 4
get.hat, 5
grouping.loading, 5

iter.tensor.bic, 6
iter.tensor.ratio, 7

MAR, 8
MAR.SE, 8
MAR1.LS, 9
MAR1.otimes, 10
MAR1.otimes (MAR2.otimes), 12
MAR1.projection, 11, *20, 21*
MAR2.LS, 11
MAR2.otimes, 12
MAR2.projection, 13
matrix_factor, 13, *14, 15*
mfmda, 14, *16, 17*
mfmda.estqk, 15
mfmda.na.iter, 15
mfmda.na.vec, 16
mfmda.nona.iter, 16
mfmda.nona.noniter, 17
mfmda.nona.vec, 18

PlotNetwork_AB, 18
pm, 19
projection, 20

rearrange, 20
run.test, *4*, 21

TAR, 21
TAR1.LS, 22
TAR1.MLE, 23
TAR1.MLE (TAR2.MLE), 26
TAR1.projection, 23
TAR1.SE.LSE, 24
TAR1.SE.MLE, 24
TAR1.VAR, 25
TAR2.LS, 25
TAR2.MLE, 26
TAR2.projection, 27
TAR2.SE.LSE, 27
TAR2.SE.MLE, 28
tensor.bic, 28
tensor.ratio, 29
tipup.init.tensor, 29
topup.init.tensor, 30
trearrange, *3*, *19*, 30

up.init.tensor, 31

var1, 31
vector_factor, *16*, *18*, 32
vts, 32