Package 'tensorEVD'

February 8, 2024

Title A Fast Algorithm to Factorize High-Dimensional Tensor Product Matrices

Version 0.1.1

Date 2024-02-07

Description Here we provide tools for the computation and factorization of high-dimensional tensor products that are formed by smaller matrices. The methods are based on properties of Kronecker products (Searle 1982, p. 265, ISBN-10: 0470009616). We evaluated this methodology by benchmark testing and illustrated its use in Gaussian Linear Models ('Lopez-Cruz et al., 2024') <doi:10.1093/g3journal/jkae001>.

LazyLoad true

Depends R (>= 3.6.0)

Suggests knitr, rmarkdown, ggplot2, ggnewscale, reshape2, RColorBrewer, pryr

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

License GPL-3

NeedsCompilation yes

Author Marco Lopez-Cruz [aut, cre], Gustavo de los Campos [aut], Paulino Perez-Rodriguez [aut]

Maintainer Marco Lopez-Cruz <lopezcru@msu.edu>

Repository CRAN

Date/Publication 2024-02-08 22:10:02 UTC

R topics documented:

Hadamard product *Hadamard product*

Description

Computes the Hadamard product between two matrices

Usage

```
Hadamard(A, B, rowsA, rowsB,
         colsA = NULL, colsB = NULL,
         make.dimnames = FALSE,
         drop = TRUE, inplace = FALSE)
```

Arguments

A	(numeric) Left numeric matrix
B	(numeric) Right numeric matrix
rowsA	(integer/character) Vector of length m with either indices or row names mapping from rows of A into the resulting hadamard product. If 'missing', it is assumed to be equal to 1,...,nrow(A)
rowsB	(integer/character) Vector of length m with either indices or row names mapping from rows of B into the resulting hadamard product. If 'missing', it is assumed to be equal to 1,...,nrow(B)
colsA	(integer/character) (Optional) Similar to rowsA, vector of length n for columns. If NULL, it is assumed to be equal to 1,...,ncol(A)
colsB	(integer/character) (Optional) Similar to rowsB, vector of length n for columns. If NULL, it is assumed to be equal to 1,...,ncol(B)
drop	Either TRUE or FALSE to whether return a uni-dimensional vector when output is a matrix with either 1 row or 1 column as per the rows and cols arguments
make.dimnames	TRUE or FALSE to whether add rownames and colnames attributes to the output
inplace	TRUE or FALSE to whether operate directly on one input matrix (A or B) when this is used as is (i.e., is not indexed; therefore, needs to be of appropiate dimensions) in the Hadamard. When TRUE the output will be overwritten on the same address occupied by the non-indexed matrix. Default inplace=FALSE

Details

Computes the $m \times n$ Hadamard product (aka element-wise or entry-wise product) matrix between matrices $\mathbf{A}_0 = \mathbf{R}_1\mathbf{A}\mathbf{C}_1'$ and $\mathbf{B}_0 = \mathbf{R}_2\mathbf{B}\mathbf{C}_2'$,

$$(\mathbf{R}_1\mathbf{A}\mathbf{C}_1') \odot (\mathbf{R}_2\mathbf{B}\mathbf{C}_2')$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are incidence matrices for rows that can be formed by integer vectors `rowsA` and `rowsB` of length m, respectively, and $\mathbf{C}_1$ and $\mathbf{C}_2$ are incidence matrices for columns that can be formed by integer vectors `colsA` and `colsB` of length n, respectively.

Matrices $\mathbf{A}_0$ and $\mathbf{B}_0$ can be obtained by matrix indexing as `A[rowsA,colsA]` and `B[rowsB,colsB]`, respectively. Therefore, the Hadamard product can be obtained directly as

```
A[rowsA,colsA]*B[rowsB,colsB]
```

The function computes the Hadamard product directly from $\mathbf{A}$ and $\mathbf{B}$ without forming $\mathbf{A}_0$ or $\mathbf{B}_0$ matrices.

Value

Returns a matrix containing the Hadamard product.

Examples

```
require(tensorEVD)

# ================================================
# Example 1. Indexing using integers
# ================================================
# Generate rectangular matrices A (nrowA x ncolA) and B (nrowB x ncolB)
nA = c(10,15)
nB = c(12,8)
A = matrix(rnorm(nA[1]*nA[2]), nrow=nA[1])
B = matrix(rnorm(nB[1]*nB[2]), nrow=nB[1])

# Define size of the Hadamard n1 x n2
n1 = 1000
n2 = 500
rowsA = sample(seq(nA[1]), n1, replace=TRUE)
rowsB = sample(seq(nB[1]), n1, replace=TRUE)
colsA = sample(seq(nA[2]), n2, replace=TRUE)
colsB = sample(seq(nB[2]), n2, replace=TRUE)

# Direct hadamard product
K1 = A[rowsA,colsA]*B[rowsB,colsB]

# Using 'Hadamard' function
K2 = Hadamard(A, B, rowsA, rowsB, colsA, colsB)

all.equal(K1,K2)  # They should be equal

# ================================================
# Example 2. Indexing using row/column names
# ================================================
# Generate squared symmetric matrices A and B
nA = 20
nB = 15
A = tcrossprod(matrix(rnorm(nA*nA), nrow=nA, dimnames=list(paste0("id",seq(nA)))))
```

```
B = tcrossprod(matrix(rnorm(nB*nB), nrow=nB, dimnames=list(paste0("id",seq(nB)))))

# Define size of the Hadamard n x n
n = 1000
IDA = sample(rownames(A), n, replace=TRUE)
IDB = sample(rownames(B), n, replace=TRUE)

# Direct hadamard product
K1 = A[IDA,IDA]*B[IDB,IDB]
dimnames(K1) = list(paste0(IDA,":",IDB), paste0(IDA,":",IDB))

# Using 'Hadamard' function
K2 = Hadamard(A, B, IDA, IDB, make.dimnames=TRUE)

all.equal(K1,K2)  # They should be equal
```

`Kronecker covariance` *Kronecker variance matrix penalization*

Description

Ridge penalization of a Kronecker covariance matrix

Usage

```
Kronecker_cov(K, Sigma = 1, Theta, byrow = FALSE,
              rows = NULL, cols = NULL, drop = TRUE,
              inplace = FALSE)
```

Arguments

`K`	(numeric) Variance matrix among subjects
`Sigma`	(numeric) A variance matrix among features. Default `Sigma=NULL` will consider an identity matrix with the same dimension as `Theta`
`Theta`	(numeric) A diagonal-shifting parameter, value to be added to the diagonals of the Kronecker variance matrix. It can be a (symmetric) matrix with the same dimension as `Sigma` for within (diagonal) and between (off-diagonal) features shifting
`byrow`	(logical) If `FALSE` (default) the output Kronecker covariance matrix corresponds to a vectorized random matrix stacked by columns, otherwise, it is assumed to be stacked by rows
`rows`	(integer) Index which rows of the Kronecker variance are to be returned. Default `rows=NULL` will return all the rows
`cols`	(integer) Index which columns of the Kronecker variance are to be returned. Default `cols=NULL` return all the columns

drop	Either TRUE or FALSE to whether return a uni-dimensional vector when output is a matrix with either 1 row or 1 column as per the rows and cols arguments
inplace	TRUE or FALSE to whether operate directly on matrix K when Sigma and Theta are scalars. This is possible only when rows=NULL and cols=NULL. When TRUE the output will be overwritten on the same address occupied by K. Default inplace=FALSE

Details

Assume that a multi-variate random matrix $\mathbf{X}$ with n subjects in rows and p features in columns follows a matrix Gaussian distribution with certain matrix of means $\mathbf{M}$ and variance-covariance matrix $\mathbf{K}$ of dimension $n \times n$ between subjects, and $\mathbf{\Sigma}$ of dimension $p \times p$ between features, then its vectorized form $vec(\mathbf{X})$ will also follow a Gaussian distribution with mean $vec(\mathbf{M})$ and variance covariance matrix equal to the Kronecker

$$\mathbf{\Sigma} \otimes \mathbf{K}$$

if the random matrix is vectorized column-wise or

$$\mathbf{K} \otimes \mathbf{\Sigma}$$

if the random matrix is vectorized row-wise.

In the uni-variate case, the problem of near-singularity can be alleviated by penalizing the variance matrix $\mathbf{K}$ by adding positive elements θ to its diagonal, i.e., $\mathbf{K} + \theta\mathbf{I}$, where $\mathbf{I}$ is an identity matrix. The same can be applied to the multi-variate case where the Kronecker variance matrix is penalized with $\mathbf{\Theta} = \{\theta_{ij}\}$ of dimensions $p \times p$, where diagonal entries will penalize within feature i and off-diagonals will penalize between features i and j. This is,

$$\mathbf{\Sigma} \otimes \mathbf{K} + \mathbf{\Theta} \otimes \mathbf{I}$$

if the random matrix is vectorized column-wise or

$$\mathbf{K} \otimes \mathbf{\Sigma} + \mathbf{I} \otimes \mathbf{\Theta}$$

if the random matrix is vectorized row-wise.

Specific rows and columns from this Kronecker can be obtained as per the rows and cols arguments without forming the whole Kronecker product (see help(Kronecker)).

Value

Returns the penalized Kronecker covariance matrix. It can be a sub-matrix of it as per the rows and cols arguments.

Examples

```
require(tensorEVD)

# Random matrix witn n subjects in rows and p features in columns
n = 20
p = 5
X = matrix(rnorm(n*p), ncol=p)

# Variance matrix among rows/columns
K = tcrossprod(X)      # for rows
Sigma = crossprod(X)   # for columns
dim(K)      # n x n matrix
dim(Sigma)  # p x p matrix

# Several examples of penalizing the Kronecker variance
# ================================================
# Example 1. Add unique value
# ================================================
theta = 10.0
G = Kronecker_cov(K, Sigma, Theta = theta)

# it must equal to:
I0 = diag(n)    # diagonal matrix of dimension n
Theta0 = matrix(theta, nrow=p, ncol=p)
G0 = kronecker(Sigma, K) + kronecker(Theta0, I0)
all.equal(G,G0)

# ================================================
# Example 2. Add feature-specific value
# ================================================
theta = rnorm(p)^2    # One value for each feature
G = Kronecker_cov(K, Sigma, Theta = theta)

# it must equal to:
Theta0 = diag(theta)
G0 = kronecker(Sigma, K) + kronecker(Theta0, I0)
all.equal(G,G0)

# ================================================
# Example 3. Add specific values within same feature
#            and between different features
# ================================================
Theta = crossprod(matrix(rnorm(p*p), ncol=p))
G = Kronecker_cov(K, Sigma, Theta = Theta)

# it must equal to:
G0 = kronecker(Sigma, K) + kronecker(Theta, I0)
all.equal(G,G0)

# Assume that random matrix X is stacked row-wise
G = Kronecker_cov(K, Sigma, Theta = Theta, byrow = TRUE)
```

```
# in this case the kronecker is inverted:
G0 = kronecker(K, Sigma) + kronecker(I0, Theta)
all.equal(G,G0)

# ============================================
# Extra: Selecting specific entries of the output
# ============================================
n = 150
p = 120
X = matrix(rnorm(n*p), ncol=p)
K = tcrossprod(X)
Sigma = crossprod(X)
Theta = crossprod(matrix(rnorm(p*p), ncol=p))

# We want only some rows and columns
rows = c(1,3,5)
cols = c(10,30,50)
G = Kronecker_cov(K, Sigma, Theta = Theta, rows=rows, cols=cols)

# this is preferable instead of:
# I0 = diag(n)
# G0 = (kronecker(Sigma, K) + kronecker(Theta, I0))[rows,cols]
# all.equal(G,G0)
```

`Kronecker product` *Kronecker product*

Description

Computes the direct Kronecker product between two matrices

Usage

```
Kronecker(A, B, rows = NULL, cols = NULL,
          make.dimnames = FALSE, drop = TRUE,
          inplace = FALSE)
```

Arguments

`A`	(numeric) Left numeric matrix
`B`	(numeric) Right numeric matrix
`rows`	(integer) Index which rows of the Kronecker are to be returned. They must range from 1 to `nrow(A)*nrow(B)`. Default `rows=NULL` will return all the rows
`cols`	(integer) Index which columns of the Kronecker are to be returned. They must range from 1 to `ncol(A)*ncol(B)`. Default `cols=NULL` return all the columns

drop	Either TRUE or FALSE to whether return a uni-dimensional vector when output is a matrix with either 1 row or 1 column as per the rows and cols arguments
make.dimnames	TRUE or FALSE to whether add rownames and colnames attributes to the output
inplace	TRUE or FALSE to whether operate directly on one input matrix (A or B) when the other one is a scalar. This is possible only when rows=NULL and cols=NULL. When TRUE the output will be overwritten on the same address occupied by the input that is not scalar. Default inplace=FALSE

Details

For any two matrices $\mathbf{A} = \{a_{ij}\}$ of dimensions $m \times n$ and $\mathbf{B} = \{b_{ij}\}$ of dimensions $p \times q$, the direct Kronecker product between them is a matrix defined as the block matrix

$$\mathbf{A} \otimes \mathbf{B} = \{a_{ij}\mathbf{B}\}$$

which is of dimensions $mp \times nq$.

Selecting specific rows and columns from the Kronecker can be done by pre- and post- multiplication with incidence matrices

$$\mathbf{R}(\mathbf{A} \otimes \mathbf{B})\mathbf{C}'$$

where $\mathbf{R}$ is an incidence matrix for rows that can be formed by an integer vector rows, and $\mathbf{C}$ is an incidence matrix for columns that can be formed by an integer vector cols. This sub-matrix of the Kronecker can be obtained by matrix indexing using rows and cols as

```
Kronecker(A,B)[rows,cols]
```

The function computes this sub-matrix of the Kronecker product directly from $\mathbf{A}$ and $\mathbf{B}$ without forming the whole Kronecker product. This is very useful if a relatively small number of row/columns are to be selected.

Value

Returns the Kronecker product matrix. It can be a sub-matrix of it as per the rows and cols arguments.

Examples

```
require(tensorEVD)

# Kronecker product of 2 vectors
A = rnorm(3)
B = rnorm(2)
(K = Kronecker(A, B))
# it must equal when using from the R-base package:
(K0 = kronecker(A, B))

# Kronecker product of 2 matrices
A = matrix(rnorm(12), ncol=3)
```

```
B = matrix(rnorm(4), ncol=2)
K = Kronecker(A, B)
# (it must equal (but faster) to:)
K0 = kronecker(A, B)
all.equal(K,K0)

# Subsetting rows/columns from the Kronecker
A = matrix(rnorm(100*150), ncol=150)
B = matrix(rnorm(100*120), ncol=120)
rows = c(1,3,5,7)
cols = c(10,20,30,50)
K = Kronecker(A, B, rows=rows, cols=cols)
# (it must equal (but much faster) to:)
K0 = kronecker(A, B)[rows,cols]
all.equal(K,K0)
```

`Tensor EVD` *Tensor EVD*

Description

Fast eigen value decomposition (EVD) of the Hadamard product of two matrices

Usage

```
tensorEVD(K1, K2, ID1, ID2, alpha = 1.0,
          EVD1 = NULL, EVD2 = NULL,
          d.min = .Machine$double.eps,
          make.dimnames = FALSE, verbose = FALSE)
```

Arguments

`K1, K2`	(numeric) Covariance structure matrices
`ID1`	(character/integer) Vector of length n with either names or indices mapping from rows/columns of `K1` into the resulting tensor product
`ID2`	(character/integer) Vector of length n with either names or indices mapping from rows/columns of `K2` into the resulting tensor product
`alpha`	(numeric) Proportion of variance of the tensor product to be explained by the tensor eigenvectors
`EVD1`	(list) (Optional) Eigenvectors and eigenvalues of `K1` as produced by the `eigen` function
`EVD2`	(list) (Optional) Eigenvectors and eigenvalues of `K2` as produced by the `eigen` function
`d.min`	(numeric) Tensor eigenvalue threshold. Default is a numeric zero. Only eigenvectors with eigenvalue passing this threshold are returned
`make.dimnames`	`TRUE` or `FALSE` to whether add `rownames` and `colnames` attributes to the output
`verbose`	`TRUE` or `FALSE` to whether show progress

Details

Let the $n \times n$ matrix $\mathbf{K}$ to be the Hadamard product (aka element-wise or entry-wise product) involving two smaller matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ of dimensions n_1 and n_2, respectively,

$$\mathbf{K} = (\mathbf{Z}_1 \mathbf{K}_1 \mathbf{Z}_1') \odot (\mathbf{Z}_2 \mathbf{K}_2 \mathbf{Z}_2')$$

where $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are incidence matrices that can be formed by integer vectors `ID1` and `ID2` of length n, respectively.

Let the eigenvalue decomposition (EVD) of $\mathbf{K}_1$ and $\mathbf{K}_2$ to be $\mathbf{K}_1 = \mathbf{V}_1 \mathbf{D}_1 \mathbf{V}_1'$ and $\mathbf{K}_2 = \mathbf{V}_2 \mathbf{D}_2 \mathbf{V}_2'$. Using properties of the Hadamard and Kronecker products, an EVD of the Hadamard product $\mathbf{K}$ can be approximated using the EVD of $\mathbf{K}_1$ and $\mathbf{K}_2$ as

$$\mathbf{K} = \mathbf{V}\mathbf{D}\mathbf{V}'$$

where $\mathbf{D} = \mathbf{D}_1 \otimes \mathbf{D}_2$ is a diagonal matrix containing $N = n_1 \times n_2$ tensor eigenvalues $d_1 \geq ... \geq d_N \geq 0$ and $\mathbf{V} = (\mathbf{Z}_1 \star \mathbf{Z}_2)(\mathbf{V}_1 \otimes \mathbf{V}_2) = [\mathbf{v}_1, ..., \mathbf{v}_N]$ is matrix containing N tensor eigenvectors $\mathbf{v}_k$; here the term $\mathbf{Z}_1 \star \mathbf{Z}_2$ is the "face-splitting product" (aka "transposed Khatri–Rao product") of matrices $\mathbf{Z}_1$ and $\mathbf{Z}_2$.

Each tensor eigenvector k is derived separately as a Hadamard product using the corresponding $i(k)$ and $j(k)$ eigenvectors $\mathbf{v}_{1i(k)}$ and $\mathbf{v}_{2j(k)}$ from $\mathbf{V}_1$ and $\mathbf{V}_2$, respectively, this is

$$\mathbf{v}_k = (\mathbf{Z}_1 \mathbf{v}_{1i(k)}) \odot (\mathbf{Z}_2 \mathbf{v}_{2j(k)})$$

Value

Returns a list object that contains the elements:

- `values`: (vector) resulting tensor eigenvalues.
- `vectors`: (matrix) resulting tensor eigenvectors.
- `totalVar`: (numeric) total variance of the tensor matrix product.

Examples

```
require(tensorEVD)
set.seed(195021)

# Generate matrices K1 and K2 of dimensions n1 and n2
n1 = 10; n2 = 15
K1 = crossprod(matrix(rnorm(n1*(n1+10)), ncol=n1))
K2 = crossprod(matrix(rnorm(n2*(n2+10)), ncol=n2))

# ============================================
# Example 1. Full design (Kronecker product)
# ============================================
ID1 = rep(seq(n1), each=n2)
ID2 = rep(seq(n2), times=n1)

# Direct EVD of the Hadamard product
```

```
K = K1[ID1,ID1]*K2[ID2,ID2]
EVD0 = eigen(K)

# Tensor EVD using K1 and K2
EVD = tensorEVD(K1, K2, ID1, ID2)

# Eigenvectors and eigenvalues are numerically equal
all.equal(EVD0$values, EVD$values)
all.equal(abs(EVD0$vectors), abs(EVD$vectors))

# If a proportion of variance explained is specified,
# only the eigenvectors needed to explain such proportion are derived
alpha = 0.95
EVD = tensorEVD(K1, K2, ID1, ID2, alpha=alpha)
dim(EVD$vectors)

# For the direct EVD
varexp = cumsum(EVD0$values/sum(EVD0$values))
index = 1:which.min(abs(varexp-alpha))
dim(EVD0$vectors[,index])

# ============================================
# Example 2. Incomplete design (Hadamard product)
# ============================================
# Eigenvectors and eigenvalues are no longer equivalent
n = n1*n2   # Sample size n
ID1 = sample(seq(n1), n, replace=TRUE) # Randomly sample of ID1
ID2 = sample(seq(n2), n, replace=TRUE) # Randomly sample of ID2

K = K1[ID1,ID1]*K2[ID2,ID2]
EVD0 = eigen(K)
EVD = tensorEVD(K1, K2, ID1, ID2)

all.equal(EVD0$values, EVD$values)
all.equal(abs(EVD0$vectors), abs(EVD$vectors))

# However, the sum of the eigenvalues is equal to the trace(K)
c(sum(EVD0$values), sum(EVD$values), sum(diag(K)))

# And provide the same approximation for K
K01 = EVD0$vectors%*%diag(EVD0$values)%*%t(EVD0$vectors)
K02 = EVD$vectors%*%diag(EVD$values)%*%t(EVD$vectors)
c(all.equal(K,K01), all.equal(K,K02))

# When n is different from N=n1xn2, both methods provide different
# number or eigenvectors/eigenvalues. The eigen function provides
# a number of eigenvectors equal to the minimum between n and N
# for the tensorEVD, this number is always N

# Sample size n being half of n1 x n2
n = n1*n2/2
ID1 = sample(seq(n1), n, replace=TRUE)
ID2 = sample(seq(n2), n, replace=TRUE)
```

```
K = K1[ID1,ID1]*K2[ID2,ID2]
EVD0 = eigen(K)
EVD = tensorEVD(K1, K2, ID1, ID2)

c(eigen=sum(EVD0$values>1E-10), tensorEVD=sum(EVD$values>1E-10))

# Sample size n being twice n1 x n2
n = n1*n2*2
ID1 = sample(seq(n1), n, replace=TRUE)
ID2 = sample(seq(n2), n, replace=TRUE)

K = K1[ID1,ID1]*K2[ID2,ID2]
EVD0 = eigen(K)
EVD = tensorEVD(K1, K2, ID1, ID2)

c(eigen=sum(EVD0$values>1E-10), tensorEVD=sum(EVD$values>1E-10))
```

Index