Package: tensorEVD (via r-universe)

September 3, 2024

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

Version 0.1.4

Date 2024-09-02

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>.

URL https://github.com/MarcooLopez/tensorEVD

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

Repository https://marcoolopez.r-universe.dev

RemoteUrl https://github.com/marcoolopez/tensorevd

RemoteRef HEAD

RemoteSha 45a8ae5d5b1330ca2074f22ecb32390204443fd7

Contents

`Hadamard product` *Hadamard product*

Description

Computes the Hadamard product between two matrices

Usage

```
Hadamard(A, B, IDrowA, IDrowB,
         IDcolA = NULL, IDcolB = NULL,
         a = 1, make.dimnames = FALSE,
         drop = TRUE, inplace = FALSE)
```

Arguments

`A`	(numeric) Numeric matrix
`B`	(numeric) Numeric matrix
`IDrowA`	(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)`
`IDrowB`	(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)`
`IDcolA`	(integer/character) (Optional) Similar to `IDrowA`, vector of length n for columns. If `NULL`, it is assumed to be equal to `IDrowA` if $m = n$
`IDcolB`	(integer/character) (Optional) Similar to `IDrowB`, vector of length n for columns. If `NULL`, it is assumed to be equal to `IDrowB` if $m = n$
`a`	(numeric) A constant to multiply the resulting Hadamard product by
`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 **A** and **B**,

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

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are incidence matrices mapping from rows of the resulting Hadamard to rows of **A** and **B**, respectively; and $\mathbf{C}_1$ and $\mathbf{C}_2$ are incidence matrices mapping from columns of the resulting Hadamard to columns of **A** and **B**, respectively.

Matrix $\mathbf{R}_1\mathbf{A}\mathbf{C}_1'$ can be obtained by matrix indexing as `A[IDrowA,IDcolA]`, where `IDrowA` and `IDcolA` are integer vectors whose entries are, respectively, the row and column number of **A** that are mapped at each row of $\mathbf{R}_1$ and $\mathbf{C}_1$, respectively. Likewise, matrix $\mathbf{R}_2\mathbf{B}\mathbf{C}_2'$ can be obtained as `B[IDrowB,IDcolB]`, where `IDrowB` and `IDcolB` are integer vectors whose entries are, respectively, the row and column number of **B** that are mapped at each row of $\mathbf{R}_2$ and $\mathbf{C}_2$, respectively. Therefore, the Hadamard product can be obtained directly as

```
A[IDrowA,IDcolA]*B[IDrowB,IDcolB]
```

The function computes the Hadamard product directly from **A** and **B** without forming $\mathbf{R}_1\mathbf{A}\mathbf{C}_1'$ or $\mathbf{R}_2\mathbf{B}\mathbf{C}_2'$ matrices. The result can be multiplied by a constant a.

Value

Returns a matrix containing the Hadamard product.

Examples

```
require(tensorEVD)

# (a) Example 1. Indexing using row/column names
# 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])
dimnames(A) = list(paste0("row",seq(nA[1])), paste0("col",seq(nA[2])))
dimnames(B) = list(paste0("row",seq(nB[1])), paste0("col",seq(nB[2])))

# Define IDs for a Hadamard of size n1 x n2
n = c(1000,500)
IDrowA = sample(rownames(A), n[1], replace=TRUE)
IDrowB = sample(rownames(B), n[1], replace=TRUE)
IDcolA = sample(colnames(A), n[2], replace=TRUE)
IDcolB = sample(colnames(B), n[2], replace=TRUE)

K1 = Hadamard(A, B, IDrowA, IDrowB, IDcolA, IDcolB, make.dimnames=TRUE)

# (it must equal to:)
K2 = A[IDrowA,IDcolA]*B[IDrowB,IDcolB]
dimnames(K2) = list(paste0(IDrowA,":",IDrowB), paste0(IDcolA,":",IDcolB))
```

```
all.equal(K1,K2)

# (b) Example 2. Indexing using integers
# Generate squared symmetric matrices A and B
nA = 20
nB = 15
A = tcrossprod(matrix(rnorm(nA*nA), nrow=nA))
B = tcrossprod(matrix(rnorm(nB*nB), nrow=nB))

# Define IDs for a Hadamard of size n x n
n = 1000
IDA = sample(seq(nA), n, replace=TRUE)
IDB = sample(seq(nB), n, replace=TRUE)

K1 = Hadamard(A, B, IDA, IDB)

# (it must equal to:)
K2 = A[IDA,IDA]*B[IDB,IDB]
all.equal(K1,K2)

# (c) Inplace calculation
# overwrite the output at the same address as the input:
IDB = sample(seq(nB), nA, replace=TRUE)

K1 = A[]                    # copy of A to be used as input
add  = pryr::address(K1)    # address of K on entry
K1 = Hadamard(K1, B, IDrowB=IDB)
pryr::address(K1) == add    # on exit, K was moved to a different address

K2 = A[]
add  = pryr::address(K2)
K2 = Hadamard(K2, B, IDrowB=IDB, inplace=TRUE)
pryr::address(K2) == add    # on exit, K remains at the same address
all.equal(K1,K2)
```

`Kronecker product` *Kronecker product*

Description

Computes the direct Kronecker product between two matrices

Usage

```
Kronecker(A, B, rows = NULL, cols = NULL, a = 1,
          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
`a`	(numeric) A constant to multiply the resulting Kronecker product by
`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 $mp \times nq$.

A sub-matrix formed by selecting specific rows and columns from the Kronecker can be obtained by pre- and post- multiplication with incidence matrices

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

where $\mathbf{R}$ is an incidence matrix mapping from rows of the resulting sub-matrix to rows of the Kronecker product, and $\mathbf{C}$ is an incidence matrix mapping from columns of the resulting sub-matrix to columns of the Kronecker product. This sub-matrix of the Kronecker can be obtained by matrix indexing as

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

where `rows` and `cols` are integer vectors whose entries are, respectively, the row and column number of the Kronecker that are mapped at each row of $\mathbf{R}$ and $\mathbf{C}$.

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)

# (a) Kronecker product of 2 vectors
A = rnorm(3)
B = rnorm(2)
(K1 = Kronecker(A, B))
# it must equal when using from the R-base package:
(K2 = kronecker(A, B))

# (b) Kronecker product of 2 matrices
A = matrix(rnorm(12), ncol=3)
B = matrix(rnorm(4), ncol=2)
K1 = Kronecker(A, B)
# (it must equal (but faster) to:)
K2 = kronecker(A, B)
all.equal(K1,K2)

# (c) 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)
K1 = Kronecker(A, B, rows=rows, cols=cols)
# (it must equal (but faster) to:)
K2 = Kronecker(A, B)[rows,cols]
all.equal(K1,K2)

# (d) Inplace calculation
# overwrite the output at the same address as the input:
K1 = A[]                     # copy of A to be used as input
add  = pryr::address(K1)     # address of K on entry
K1 = Kronecker(K1, B=0.5)
pryr::address(K1) == add     # on exit, K was moved to a different address

K2 = A[]
add  = pryr::address(K2)
K2 = Kronecker(K2, B=0.5, inplace=TRUE)
pryr::address(K2) == add     # on exit, K remains at the same address
all.equal(K1,K2)
```

Multivariate variance matrix

Multivariate variance matrix penalization

Description

Ridge penalization of a multi-variate (co)variance matrix taking the form of either a Kronecker or Hadamard product

Usage

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

Hadamard_cov(Sigma = 1, K, Theta, IDS, IDK,
             drop = TRUE, inplace = FALSE)
```

Arguments

`Sigma`	(numeric) A variance matrix among features. If is scalar, a scaled identity matrix with the same dimension as `Theta` is used
`K`	(numeric) Variance matrix among subjects
`Theta`	(numeric) A diagonal-shifting parameter, value to be added to the diagonals of the resulting (co)variance matrix. It should be a (symmetric) matrix with the same dimension as `Sigma`
`rows`	(integer) Index which rows of the (Kronecker product) (co)variance matrix are to be returned. Default `rows=NULL` will return all the rows
`cols`	(integer) Index which columns of the (Kronecker product) (co)variance are to be returned. Default `cols=NULL` return all the columns
`IDS`	(integer/character) Vector with either indices or row names mapping from rows/columns of `Sigma` and `Theta` into the resulting (Hadamard product) (co)variance matrix
`IDK`	(integer/character) Vector with either indices or row names mapping from rows/columns of `K` into the resulting (Hadamard product) (co)variance matrix
`swap`	(logical) Either `TRUE` or `FALSE` (default) to whether swap the order of the matrices in the resulting (Kronecker product) (co)variance matrix
`drop`	(logical) 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`	(logical) Either `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 matrix $\mathbf{K}$ of dimension $n \times n$ between subjects, and $\mathbf{\Sigma}$ of dimension $p \times p$ between features.

Kronecker product form.

The random variable $\mathbf{x} = vec(\mathbf{X})$, formed by stacking columns of $\mathbf{X}$, is a vector of length np that also follow a Gaussian distribution with mean $vec(\mathbf{M})$ and (co)variance covariance matrix taking the Kronecker form

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

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 product (co)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,

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

The second Kronecker summand $\mathbf{\Theta} \otimes \mathbf{I}$ is a sparse matrix consisting of non-zero diagonal and sub-diagonals. The `Kronecker_cov` function derives the penalized Kronecker (co)variance matrix by computing densely only the first Kronecker summand $\mathbf{\Sigma} \otimes \mathbf{K}$, and then calculating and adding accordingly only the non-zero entries of $\mathbf{\Theta} \otimes \mathbf{I}$.

Note: Swapping the order of the matrices in the above Kronecker operations will yield a different result. In this case the penalized matrix

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

corresponds to the penalized multi-variate (co)variance matrix of the transposed of the above multi-variate random matrix $\mathbf{X}$, now with features in rows and subjects in columns. This can be achieved by setting `swap=TRUE` in the `Kronecker_cov` function.

Hadamard product form.

Assume the random variable $\mathbf{x}_0$ is a subset of $\mathbf{x}$ containing entries corresponding to specific combinations of subjects and features, then the (co)variance matrix of the vector $\mathbf{x}_0$ will be a Hadamard product formed by the entry-wise product of only the elements of $\mathbf{\Sigma}$ and $\mathbf{K}$ involved in the combinations contained in $\mathbf{x}_0$; this is

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

where $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are incidence matrices mapping from entries of the random variable $\mathbf{x}_0$ to rows (and columns) of $\mathbf{\Sigma}$ and $\mathbf{K}$, respectively. This (co)variance matrix can be obtained using matrix indexing (see `help(Hadamard)`), as

```
Sigma[IDS,IDS]*K[IDK,IDK]
```

where `IDS` and `IDK` are integer vectors whose entries are the row (and column) number of $\mathbf{\Sigma}$ and $\mathbf{K}$, respectively, that are mapped at each row of $\mathbf{Z}_1$ and $\mathbf{Z}_2$, respectively.

The penalized version of this Hadamard product (co)variance matrix will be

$$(\mathbf{Z}_1\mathbf{\Sigma}\mathbf{Z}_1') \odot (\mathbf{Z}_2\mathbf{K}\mathbf{Z}_2') + (\mathbf{Z}_1\mathbf{\Theta}\mathbf{Z}_1') \odot (\mathbf{Z}_2\mathbf{I}\mathbf{Z}_2')$$

The `Hadamard_cov` function derives this penalized (co)variance matrix using matrix indexing, as

```
Sigma[IDS,IDS]*K[IDK,IDK] + Theta[IDS,IDS]*I[IDK,IDK]
```

Likewise, this function computes densely only the first Hadamard summand and then calculates and adds accordingly only the non-zero entries of the second summand.

Value

Returns the penalized (co)variance matrix formed either as a Kronecker or Hadamard product. For the Kronecker product case, it can be a sub-matrix of the Kronecker product as per the `rows` and `cols` arguments.

Examples

```
require(tensorEVD)

# Generate rectangular some covariance matrices
n = 30;  p = 10

K = crossprod(matrix(rnorm(n*p), ncol=n))      # n x n matrix
Sigma = crossprod(matrix(rnorm(n*p), ncol=p))  # p x p matrix
Theta = crossprod(matrix(rnorm(n*p), ncol=p))  # p x p matrix

# ============================================
# Kronecker covariance
# ============================================
G1 = Kronecker_cov(Sigma, K, Theta = Theta)

# it must equal to:
D = diag(n)    # diagonal matrix of dimension n
G2 = Kronecker(Sigma, K) + Kronecker(Theta, D)
all.equal(G1,G2)

# (b) Swapping the order of the matrices
G1 = Kronecker_cov(Sigma, K, Theta, swap = TRUE)

# in this case the kronecker is swapped:
G2 = Kronecker(K, Sigma) + Kronecker(D, Theta)
all.equal(G1,G2)

# (c) Selecting specific entries of the output
# We want only some rows and columns
rows = c(1,3,5)
cols = c(10,30,50)
G1 = Kronecker_cov(Sigma, K, Theta, rows=rows, cols=cols)

# this can be preferable instead of:
G2 = (Kronecker(Sigma, K) + Kronecker(Theta, D))[rows,cols]
all.equal(G1,G2)

# (d) Inplace calculation
# overwrite the output at the same address as the input:
G1 = K[]                     # copy of K to be used as input
add  = pryr::address(G1)     # address of G on entry
G1 = Kronecker_cov(Sigma=0.5, G1, Theta=1.5)
pryr::address(G1) == add     # on exit, G was moved to a different address

G2 = K[]
add  = pryr::address(G2)
```

```
G2 = Kronecker_cov(Sigma=0.5, G2, Theta=1.5, inplace=TRUE)
pryr::address(G2) == add     # on exit, G remains at the same address
all.equal(G1,G2)

# ============================================
# Hadamard covariance
# ============================================
# Define IDs for a Hadamard of size m x m
m = 1000
IDS = sample(1:p, m, replace=TRUE)
IDK = sample(1:n, m, replace=TRUE)

G1 = Hadamard_cov(Sigma, K, Theta, IDS=IDS, IDK=IDK)

# it must equal to:
G2 = Sigma[IDS,IDS]*K[IDK,IDK] + Theta[IDS,IDS]*D[IDK,IDK]
all.equal(G1,G2)

# (b) Inplace calculation
# overwrite the output at the same address as the input:
G1 = K[]                     # copy of K to be used as input
add  = pryr::address(G1)     # address of G on entry
G1 = Hadamard_cov(Sigma=0.5, G1, Theta=1.5, IDS=rep(1,n))
pryr::address(G1) == add     # on exit, G was moved to a different address

G2 = K[]
add  = pryr::address(G2)
G2 = Hadamard_cov(Sigma=0.5, G2, Theta=1.5, IDS=rep(1,n), inplace=TRUE)
pryr::address(G2) == add     # on exit, G remains at the same address
all.equal(G1,G2)
```

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 mapping from rows (and columns) of the resulting Hadamard to rows (and columns) of $\mathbf{K}_1$ and $\mathbf{K}_2$, 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

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

The `tensorEVD` function derives each of these eigenvectors $\mathbf{v}_k$ by matrix indexing using integer vectors `ID1` and `ID2`. The entries of these vectors are the row (and column) number of $\mathbf{K}_1$ and $\mathbf{K}_2$ that are mapped at each row of $\mathbf{Z}_1$ and $\mathbf{Z}_2$, respectively.

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))

# (a) 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))

# (b) 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])

# (c) 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

# (d) 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))

# (e) 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))
```

`Weighted sum` *Weighted sum*

Description

Computes the Hadamard product between two matrices

Usage

```
Sum(a = 1, A, b = 1, B, IDrowA, IDrowB,
    IDcolA = NULL, IDcolB = NULL,
```

```
make.dimnames = FALSE, drop = TRUE,
inplace = FALSE)
```

Arguments

a	(numeric) A constant to multiply the first matrix by
A	(numeric) Numeric matrix
b	(numeric) A constant to multiply the second matrix by
B	(numeric) Numeric matrix
IDrowA	(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)
IDrowB	(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)
IDcolA	(integer/character) (Optional) Similar to IDrowA, vector of length n for columns. If NULL, it is assumed to be equal to IDrowA if $m = n$
IDcolB	(integer/character) (Optional) Similar to IDrowB, vector of length n for columns. If NULL, it is assumed to be equal to IDrowB if $m = n$
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 $m \times n$ weighted sum matrix between matrices $\mathbf{A}$ and $\mathbf{B}$,

$$a(\mathbf{R}_1\mathbf{A}\mathbf{C}_1') + b(\mathbf{R}_2\mathbf{B}\mathbf{C}_2')$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are incidence matrices mapping from rows of the resulting sum to rows of $\mathbf{A}$ and $\mathbf{B}$, respectively; and $\mathbf{C}_1$ and $\mathbf{C}_2$ are incidence matrices mapping from columns of the resulting sum to columns of $\mathbf{A}$ and $\mathbf{B}$, respectively.

Matrix $\mathbf{R}_1\mathbf{A}\mathbf{C}_1'$ can be obtained by matrix indexing as A[IDrowA,IDcolA], where IDrowA and IDcolA are integer vectors whose entries are, respectively, the row and column number of $\mathbf{A}$ that are mapped at each row of $\mathbf{R}_1$ and $\mathbf{C}_1$, respectively. Likewise, matrix $\mathbf{R}_2\mathbf{B}\mathbf{C}_2'$ can be obtained as B[IDrowB,IDcolB], where IDrowB and IDcolB are integer vectors whose entries are, respectively, the row and column number of $\mathbf{B}$ that are mapped at each row of $\mathbf{R}_2$ and $\mathbf{C}_2$, respectively. Therefore, the weighted sum can be obtained directly as

```
a*A[IDrowA,IDcolA] + b*B[IDrowB,IDcolB]
```

The function computes the Hadamard product directly from $\mathbf{A}$ and $\mathbf{B}$ without forming $\mathbf{R}_1\mathbf{A}\mathbf{C}_1'$ or $\mathbf{R}_2\mathbf{B}\mathbf{C}_2'$ matrices. The result can be multiplied by a constant a.

Value

Returns a matrix containing the Hadamard product.

Examples

```
require(tensorEVD)

# 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 IDs for a Hadamard of size n1 x n2
n = c(1000,500)
IDrowA = sample(nA[1], n[1], replace=TRUE)
IDrowB = sample(nB[1], n[1], replace=TRUE)
IDcolA = sample(nA[2], n[2], replace=TRUE)
IDcolB = sample(nB[2], n[2], replace=TRUE)

a = rnorm(1)
b = rnorm(1)

K1 = Sum(a, A, b, B, IDrowA, IDrowB, IDcolA, IDcolB)

# (it must equal to:)
K2 = a*A[IDrowA,IDcolA] + b*B[IDrowB,IDcolB]
all.equal(K1,K2)
```

Index