Package ‘rTensor’

Type Package

Title Tools for Tensor Analysis and Decomposition

Version 1.3

Author James Li and Jacob Bien and Martin Wells

Maintainer James Li <jamesyili@gmail.com>

Description A set of tools for creation, manipulation, and modeling of tensors with arbitrary number of modes. A tensor in the context of data analysis is a multidimensional array. rTensor does this by providing a S4 class 'Tensor' that wraps around the base 'array' class. rTensor provides common tensor operations as methods, including matrix unfolding, summing/averaging across modes, calculating the Frobenius norm, and taking the inner product between two tensors. Familiar array operations are overloaded, such as index subsetting via '[' and element-wise operations. rTensor also implements various tensor decomposition, including CP, GLRAM, MPCA, PVD, and Tucker. For tensors with 3 modes, rTensor also implements transpose, t-product, and t-SVD, as defined in Kilmer et al. (2013). Some auxiliary functions include the Khatri-Rao product, Kronecker product, and the Hamadard product for a list of matrices.

License GPL (>= 2)

Depends methods, R (>= 2.10.0)

LazyData true

Date 2015-12-14

URL http://jamesyili.github.io/rTensor

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2015-12-15 00:56:41
R topics documented:

rTensor-package ........................................... 3
as.tensor .................................................... 4
cp .......................................................... 4
cs_fold ...................................................... 6
cs_unfold-methods ......................................... 6
dim-methods ................................................ 7
faces_tnsr .................................................. 7
fnorm-methods .............................................. 8
fold .......................................................... 9
hadamard_list ............................................. 10
head-methods .............................................. 10
hosvd ......................................................... 11
initialize-methods ......................................... 12
innerProd-methods ......................................... 13
khatri_rao .................................................. 13
khatri_rao_list ........................................... 14
kronecker_list ............................................. 15
k_fold ......................................................... 16
k_unfold-methods ......................................... 17
matvec-methods ............................................ 18
modeMean-methods ......................................... 19
modeSum-methods .......................................... 20
mpca .......................................................... 21
Ops-methods ............................................... 22
plot_orl ...................................................... 23
print-methods ............................................... 24
pvd .......................................................... 24
rand_tensor ............................................... 26
rs_fold ....................................................... 26
rs_unfold-methods ......................................... 27
show-methods ............................................... 27
t-methods .................................................... 28
tail-methods ............................................... 29
Tensor-class ............................................... 29
tperm-methods .............................................. 32
ttl ........................................................... 32
ttm ........................................................... 33
tucker ......................................................... 34
t_mult ......................................................... 36
t_svd ........................................................ 37
t_svd_reconstruct ......................................... 38
unfold-methods ............................................ 38
unmatvec ..................................................... 39
vec-methods ................................................ 40
[-methods .................................................. 41

Index 42
Description

This package is centered around the 

\texttt{Tensor-class}, which defines a S4 class for tensors of arbitrary number of modes. A vignette and/or a possible paper will be included in a future release of this package.

Details

This page will summarize the full functionality of this package. Note that since all the methods associated with S4 class \texttt{Tensor-class} are documented there, we will not duplicate it here.

The remaining functions can be split into two groups: the first is a set of tensor decompositions, and the second is a set of helper functions that are useful in tensor manipulation.

rTensor implements the following tensor decompositions:

\begin{itemize}
  \item \texttt{cp} Canonical Polyadic (CP) decomposition
  \item \texttt{tucker} General Tucker decomposition
  \item \texttt{mpca} Multilinear Principal Component Analysis; note that for 3-Tensors this is also known as Generalized Low Rank Approximation of Matrices (GLRAM)
  \item \texttt{hosvd} (Truncated-)Higher-order singular value decomposition
  \item \texttt{t_svd} Tensor singular value decomposition; 3-Tensors only; also note that there is an associated reconstruction function \texttt{t_svd_reconstruct}
  \item \texttt{pvd} Population value decomposition of images; 3-Tensors only
\end{itemize}

rTensor also provides a set functions for tensors multiplication:

\begin{itemize}
  \item \texttt{ttm} Tensor times matrix, aka m-mode product
  \item \texttt{ttl} Tensor times list (of matrices)
  \item \texttt{t_mult} Tensor product based on block circulant unfolding; only implemented for a pair of 3-Tensors
\end{itemize}

...as well as for matrices:

\begin{itemize}
  \item \texttt{hadamard_list} Computes the Hamadard (element-wise) product of a list of matrices
  \item \texttt{kronecker_list} Computes the Kronecker product of a list of matrices
  \item \texttt{khatri_rao} Computes the Khatri-Rao product of two matrices
  \item \texttt{khatri_rao_list} Computes the Khatri-Rao product of a list of matrices
  \item \texttt{fold} General folding of a matrix into a tensor
  \item \texttt{k_fold} Inverse operation for \texttt{k_unfold}
  \item \texttt{unmatvec} Inverse operation for \texttt{matvec}
\end{itemize}

For more information on any of the functions, please consult the individual man pages.
as.tensor  

Tensor Conversion

Description

Create a Tensor-class object from an array, matrix, or vector.

Usage

as.tensor(x, drop = FALSE)

Arguments

- **x**: an instance of array, matrix, or vector
- **drop**: whether or not modes of 1 should be dropped

Value

a Tensor-class object

Examples

```r
# From vector
vec <- runif(100); vecT <- as.tensor(vec); vecT
# From matrix
mat <- matrix(runif(1000), nrow=100, ncol=10)
matT <- as.tensor(mat); matT
# From array
indices <- c(10,20,30,40)
arr <- array(runif(prod(indices)), dim = indices)
arrT <- as.tensor(arr); arrT
```

cp  

Canonical Polyadic Decomposition

Description

Canonical Polyadic (CP) decomposition of a tensor, aka CANDECOMP/PARAFAC. Approximate a K-Tensor using a sum of num_components rank-1 K-Tensors. A rank-1 K-Tensor can be written as an outer product of K vectors. There are a total of num_components * tnsr@num_modes vectors in the output, stored in tnsr@num_modes matrices, each with num_components columns. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below tol, or the max_iter number of iterations has been reached. For more details on CP decomposition, consult Kolda and Bader (2009).

Usage

\[ \text{cp}(\text{tnsr}, \text{num\_components} = \text{NULL}, \text{max\_iter} = 25, \text{tol} = 1e-05) \]

Arguments

- tnsr: Tensor with K modes
- num\_components: the number of rank-1 K-Tensors to use in approximation
- max\_iter: maximum number of iterations if error stays above tol
- tol: relative Frobenius norm error tolerance

Details

Uses the Alternating Least Squares (ALS) estimation procedure. A progress bar is included to help monitor operations on large tensors.

Value

A list containing the following:

- lambdas: a vector of normalizing constants, one for each component
- U: a list of matrices - one for each mode - each matrix with num\_components columns
- conv: whether or not resid < tol by the last iteration
- norm\_percent: the percent of Frobenius norm explained by the approximation
- est: estimate of tnsr after compression
- fnorm\_resid: the Frobenius norm of the error fnorm(est-tnsr)
- all\_resids: vector containing the Frobenius norm of error for all the iterations

References


See Also

tucker

Examples

subject <- faces\_tnsr[,14,]
cpd <- cp(subject, num\_components=10)
cpd\$conv
cpd\$norm\_percent
plot(cpd\$all\_resids)
## cs_fold

### Description

DEPRECATED. Please see `unmatvec`.

### Usage

```r
cs_fold(mat, m = NULL, modes = NULL)
```

### Arguments

- `mat`: matrix to be folded
- `m`: the mode corresponding to `cs_unfold`
- `modes`: the original modes of the tensor

## cs_unfold-methods

### Description

DEPRECATED. Please see `matvec-methods` and `unfold-methods`.

### Usage

```r
cs_unfold(tnsr, m)
```

#### Details

```r
## S4 method for signature 'Tensor'
cs_unfold(tnsr, m = NULL)
```

### Arguments

- `tnsr`: Tensor instance
- `m`: mode to be unfolded on

### Details

```r
cs_unfold(tnsr, m=NULL)
```
dim-methods

Mode Getter for Tensor

Description
Return the vector of modes from a tensor

Usage
```r
## S4 method for signature 'Tensor'
dim(x)
```

Arguments
- `x` the Tensor instance

Details
```r
dim(x)
```

Value
an integer vector of the modes associated with `x`

Examples
```r
tnsr <- rand_tensor()
dim(tnsr)
```

faces_tnsr

ORL Database of Faces

Description
A dataset containing pictures of 40 individuals under 10 different lightings. Each image has 92 x 112 pixels. Structured as a 4-tensor with modes 92 x 112 x 40 x 10.

Usage
```r
faces_tnsr
```

Format
A Tensor object with modes 92 x 112 x 40 x 10. The first two modes correspond to the image pixels, the third mode corresponds to the individual, and the last mode corresponds to the lighting.
Source

http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

See Also

plot_orl

---

### fnorm-methods

**Tensor Frobenius Norm**

**Description**

Returns the Frobenius norm of the Tensor instance.

**Usage**

```r
fnorm(tnsr)
```

**Arguments**

- `tnsr` the Tensor instance

**Details**

```r
fnorm(tnsr)
```

**Value**

numeric Frobenius norm of `x`

**Examples**

```r
tnsr <- rand_tensor()
fnorm(tnsr)
```
Description

General folding of a matrix into a Tensor. This is designed to be the inverse function to `unfold-methods`, with the same ordering of the indices. This amounts to following: if we were to unfold a Tensor using a set of `row_idx` and `col_idx`, then we can fold the resulting matrix back into the original Tensor using the same `row_idx` and `col_idx`.

Usage

```
fold(mat, row_idx = NULL, col_idx = NULL, modes = NULL)
```

Arguments

- `mat`: matrix to be folded into a Tensor
- `row_idx`: the indices of the modes that are mapped onto the row space
- `col_idx`: the indices of the modes that are mapped onto the column space
- `modes`: the modes of the output Tensor

Details

This function uses `aperm` as the primary workhorse.

Value

Tensor object with modes given by `modes`

References


See Also

`unfold-methods, k_fold, unmatvec`

Examples

```
tnsr <- new("tensor",3L,c(3L,4L,5L),data=runif(60))
matT3<-unfold(tnsr,row_idx=2,col_idx=c(3,1))
identical(fold(matT3,row_idx=2,col_idx=c(3,1),modes=c(3,4,5)),tnsr)
```
### hadamard_list

**List hadamard Product**

**Description**

Returns the hadamard (element-wise) product from a list of matrices or vectors. Commonly used for n-mode products and various Tensor decompositions.

**Usage**

```r
hadamard_list(L)
```

**Arguments**

- `L`: list of matrices or vectors

**Value**

matrix that is the hadamard product

**Note**

The modes/dimensions of each element in the list must match.

**See Also**

`kronecker_list`, `khatri_rao_list`

**Examples**

```r
lizt <- list('mat1' = matrix(runif(40), ncol=4),
'mat2' = matrix(runif(40), ncol=4),
'mat3' = matrix(runif(40), ncol=4))
dim(hadamard_list(lizt))
```

### head-methods

**Head for Tensor**

**Description**

Extend head for Tensor

**Usage**

```r
## S4 method for signature 'Tensor'
head(x, ...)
```
hosvd

Arguments

x the Tensor instance
... additional parameters to be passed into head()

Details

head(x, ...)

See Also
tail-methods

Examples

tnsr <- rand_tensor()
head(tnsr)

hosvd

(Truncated-)Higher-order SVD

Description

Higher-order SVD of a K-Tensor. Write the K-Tensor as a (m-mode) product of a core Tensor (possibly smaller modes) and K orthogonal factor matrices. Truncations can be specified via ranks (making them smaller than the original modes of the K-Tensor will result in a truncation). For the mathematical details on HOSVD, consult Lathauwer et al. (2000).

Usage

hosvd(tnsr, ranks = NULL)

Arguments

tnsr Tensor with K modes
ranks a vector of desired modes in the output core tensor, default is tnsr@modes

Details

A progress bar is included to help monitor operations on large tensors.

Value

a list containing the following:
Z core tensor with modes specified by ranks
U a list of orthogonal matrices, one for each mode
est estimate of tnsr after compression
fnorm_resid the Frobenius norm of the error fnorm(est - tnsr) - if there was no truncation, then this is on the order of mach_eps * fnorm.
Note

The length of ranks must match tnsr@num_modes.

References


See Also
tucker

Examples

```r
tnsr <- rand_tensor(c(6,7,8))
hosvdD <- hosvd(tnsr)
hosvdD$fnorm_resid
hosvdD2 <- hosvd(tnsr, ranks=c(3,3,4))
hosvdD2$fnorm_resid
```

initialize-methods

**Initializes a Tensor instance**

Description

Not designed to be called by the user. Use as.tensor instead.

Usage

```r
## S4 method for signature 'Tensor'
initialize(.Object, num_modes = NULL, modes = NULL, data = NULL)
```

Arguments

- `.Object` the tensor object
- `num_modes` number of modes of the tensor
- `modes` modes of the tensor
- `data` can be vector, matrix, or array

See Also

as.tensor
innerProd-methods

Tensors Inner Product

Description

Returns the inner product between two Tensors

Usage

innerProd(tnsr1, tnsr2)

## S4 method for signature 'Tensor,Tensor'
innerProd(tnsr1, tnsr2)

Arguments

tnsr1 first Tensor instance

tnsr2 second Tensor instance

Details

innerProd(tnsr1, tnsr2)

Value

inner product between x1 and x2

Examples

tnsr1 <- rand_tensor()

tnsr2 <- rand_tensor()

innerProd(tnsr1, tnsr2)

---

khatri_rao

Khatri-Rao Product

Description

Returns the Khatri-Rao (column-wise Kronecker) product of two matrices. If the inputs are vectors then this is the same as the Kronecker product.

Usage

khatri_rao(x, y)
**khatri_rao_list**

Arguments

  - x: first matrix
  - y: second matrix

Value

  matrix that is the Khatri-Rao product

Note

  The number of columns must match in the two inputs.

See Also

  - kronecker
  - khatri_rao_list

Examples

  ```r
dim(khatri_rao(matrix(runif(12), ncol=4), matrix(runif(12), ncol=4)))
```

**khatri_rao_list**  
*List Khatri-Rao Product*

Description

  Returns the Khatri-Rao product from a list of matrices or vectors. Commonly used for n-mode products and various Tensor decompositions.

Usage

  ```r
  khatri_rao_list(L, reverse = FALSE)
  ```

Arguments

  - L: list of matrices or vectors
  - reverse: whether or not to reverse the order

Value

  matrix that is the Khatri-Rao product

Note

  The number of columns must match in every element of the input list.

See Also

  - khatri_rao
**kronecker_list**

**List Kronecker Product**

**Description**

Returns the Kronecker product from a list of matrices or vectors. Commonly used for n-mode products and various Tensor decompositions.

**Usage**

```r
kronecker_list(L)
```

**Arguments**

- `L` list of matrices or vectors

**Value**

matrix that is the Kronecker product

**See Also**

`hadamard_list`, `khatri_rao_list`, `kronecker`

**Examples**

```r
smalllizt <- list('mat1' = matrix(runif(12), ncol=4),
                 'mat2' = matrix(runif(12), ncol=4),
                 'mat3' = matrix(runif(12), ncol=4))
dim(khatri_rao_list(smalllizt))

dim(kronecker_list(smalllizt))
```
### k_fold

#### k-mode Folding of Matrix

**Description**

k-mode folding of a matrix into a Tensor. This is the inverse function to `k_unfold` in the m mode. In particular, \( k\_fold(k\_unfold(tnsr, m), m, getModes(tnsr)) \) will result in the original Tensor.

**Usage**

```r
k_fold(mat, m = NULL, modes = NULL)
```

**Arguments**

- `mat`: matrix to be folded into a Tensor
- `m`: the index of the mode that is mapped onto the row indices
- `modes`: the modes of the output Tensor

**Details**

This is a wrapper function to `fold`.

**Value**

Tensor object with modes given by `modes`

**References**


**See Also**

- `k_unfold-methods`, `fold`, `unmatvec`

**Examples**

```r
tnsr <- new("tensor", 3L, c(3L, 4L, 5L), data=runif(60))
matT2<-k_unfold(tnsr,m=2)
identical(k_fold(matT2,m=2, modes=c(3,4,5)),tnsr)
```
Description

Unfolding of a tensor by mapping the kth mode (specified through parameter \( m \)), and all other modes onto the column space. This the most common type of unfolding operation for Tucker decompositions and its variants. Also known as k-mode matricization.

Usage

\[
\text{k\_unfold}(\text{tnsr}, m)
\]

## S4 method for signature 'Tensor'
\[
\text{k\_unfold}(\text{tnsr}, m = \text{NULL})
\]

Arguments

- **tnsr**: the Tensor instance
- **m**: the index of the mode to unfold on

Details

\[
\text{k\_unfold}(\text{tnsr}, m=\text{NULL})
\]

Value

matrix with \( \times \text{modes}[m] \) rows and \( \text{prod}(\times \text{modes}[-m]) \) columns

References


See Also

- `matvec-methods` and `unfold-methods`

Examples

\[
\text{tnsr} \leftarrow \text{rand\_tensor()}
\]
\[
\text{matT2} \leftarrow \text{rs\_unfold(tnsr, m=2)}
\]
Description

For 3-tensors only. Stacks the slices along the third mode. This is the prevalent unfolding for T-SVD and T-MULT based on block circulant matrices.

Usage

matvec(tnsr)

## S4 method for signature 'Tensor'
matvec(tnsr)

Arguments

tnsr the Tensor instance

Details

matvec(tnsr)

Value

matrix with prod(x@modes[-m]) rows and x@modes[m] columns

References


See Also

k_unfold-methods and unfold-methods

Examples

tnsr <- rand_tensor(c(2,3,4))
matT1<- matvec(tnsr)
modeMean-methods

Tensor Mean Across Single Mode

Description

Given a mode for a K-tensor, this returns the K-1 tensor resulting from taking the mean across that particular mode.

Usage

modeMean(tnsr, m, drop)

## S4 method for signature 'Tensor'

modeMean(tnsr, m = NULL, drop = FALSE)

Arguments

- tnsr: the Tensor instance
- m: the index of the mode to average across
- drop: whether or not mode m should be dropped

Details

modeMean(tnsr, m=NULL, drop=FALSE)

Value

K-1 or K Tensor, where K = x@num_modes

See Also

- modeSum

Examples

```r
tnsr <- rand_tensor()
modeMean(tnsr, 1, drop=TRUE)
```
Description

Given a mode for a K-tensor, this returns the K-1 tensor resulting from summing across that particular mode.

Usage

```r
modeSum(tnsr, m, drop)
```

## S4 method for signature 'Tensor'

```r
modeSum(tnsr, m = NULL, drop = FALSE)
```

Arguments

- `tnsr`: the Tensor instance
- `m`: the index of the mode to sum across
- `drop`: whether or not mode `m` should be dropped

Details

```r
modeSum(tnsr, m = NULL, drop = FALSE)
```

Value

A K-1 or K tensor, where \( K = x @ \text{num\_modes} \)

See Also

- `modeMean`

Examples

```r
tnsr <- rand_tensor()
modeSum(tnsr, 3, drop = TRUE)
```
**Description**

This is basically the Tucker decomposition of a K-Tensor, `tucker`, with one of the modes uncompressed. If K = 3, then this is also known as the Generalized Low Rank Approximation of Matrices (GLRAM). This implementation assumes that the last mode is the measurement mode and hence uncompressed. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below `tol`, or the `max_iter` number of iterations has been reached. For more details on the MPCA of tensors, consult Lu et al. (2008).

**Usage**

```r
cmpa(tnsr, ranks = NULL, max_iter = 25, tol = 1e-05)
```

**Arguments**

- `tnsr` Tensor with K modes
- `ranks` a vector of the compressed modes of the output core Tensor, this has length K-1
- `max_iter` maximum number of iterations if error stays above `tol`
- `tol` relative Frobenius norm error tolerance

**Details**

Uses the Alternating Least Squares (ALS) estimation procedure. A progress bar is included to help monitor operations on large tensors.

**Value**

a list containing the following:

- `z_ext` the extended core tensor, with the first K-1 modes given by `ranks`
- `U` a list of K-1 orthogonal factor matrices - one for each compressed mode, with the number of columns of the matrices given by `ranks`
- `conv` whether or not `resid < tol` by the last iteration
- `est` estimate of `tnsr` after compression
- `norm_percent` the percent of Frobenius norm explained by the approximation
- `fnorm_resid` the Frobenius norm of the error `fnorm(est-tnsr)`
- `all_resids` vector containing the Frobenius norm of error for all the iterations

**Note**

The length of `ranks` must match `tnsr@num_modes-1`. 
References


See Also

tucker, hosvd

Examples

subject <- faces_tnsr[, , 21]
mpcaD <- mpca(subject, ranks = c(10, 10))
mpcaD$conv
mpcaD$norm_percent
plot(mpcaD$all_resids)

Ops-methods

Conformable elementwise operators for Tensor

Description

Overloads elementwise operators for tensors, arrays, and vectors that are conformable (have the same modes).

Usage

## S4 method for signature 'Tensor,Tensor'
Ops(e1, e2)

Arguments

e1 left-hand object
e2 right-hand object

Examples

tnsr <- rand_tensor(c(3, 4, 5))
ntsr2 <- rand_tensor(c(3, 4, 5))
tnsrsum <- ntsr + ntsr2
tnsrdiff <- ntsr - ntsr2
tnsrelemprod <- ntsr * ntsr2
tnsrelemquot <- ntsr / ntsr2
for (i in 1:3L){
  for (j in 1:4L){
    for (k in 1:5L){
      stopifnot(tnsrsum@data[i, j, k] == ntsr@data[i, j, k] + ntsr2@data[i, j, k])
      stopifnot(tnsrdiff@data[i, j, k] == (ntsrs@data[i, j, k] - ntsr2@data[i, j, k]))
      stopifnot(tnsrelemprod@data[i, j, k] == ntsr@data[i, j, k] * ntsr2@data[i, j, k])
    }
  }
}
**plot_orl**

Function to plot the ORL Database of Faces

**Description**

A wrapper function to image() to allow easy visualization of faces_tnsr, the ORL Face Dataset.

**Usage**

```r
plot_orl(subject = 1, condition = 1)
```

**Arguments**

- **subject** which subject to plot (1-40)
- **condition** which lighting condition (1-10)

**References**


**See Also**

- `faces_tnsr`

**Examples**

```r
plot_orl(subject=5,condition=4)
plot_orl(subject=2,condition=7)
```
Description

Extend print for Tensor

Usage

## S4 method for signature 'Tensor'
print(x, ...)

Arguments

x the Tensor instance
...
additional parameters to be passed into print()

Details

print(x,...)

See Also

show

Examples

tnsr <- rand_tensor()
print(tnsr)

---

pvd

Population Value Decomposition

Description

The default Population Value Decomposition (PVD) of a series of 2D images. Constructs population-level matrices P, V, and D to account for variances within as well as across the images. Structurally similar to Tucker (tucker) and GLRAM (mpca), but retains crucial differences. Requires $2 \times n_3 + 2$ parameters to specified the final ranks of P, V, and D, where $n_3$ is the third mode (how many images are in the set). Consult Crainiceanu et al. (2013) for the construction and rationale behind the PVD model.

Usage

pvd(tnsr, uranks = NULL, wranks = NULL, a = NULL, b = NULL)
**Arguments**

- **tnsr**: 3-Tensor with the third mode being the measurement mode
- **uranks**: ranks of the U matrices
- **wranks**: ranks of the W matrices
- **a**: rank of $P = U^*t(U)$
- **b**: rank of $D = W^*t(W)$

**Details**

The PVD is not an iterative method, but instead relies on $n \times 2$ separate PCA decompositions. The third mode is for how many images are in the set.

**Value**

A list containing the following:

- $P$: population-level matrix $P = U^*t(U)$, where $U$ is constructed by stacking the truncated left eigenvectors of slicewise PCA along the third mode
- $V$: a list of image-level core matrices
- $D$: population-level matrix $D = W^*t(W)$, where $W$ is constructed by stacking the truncated right eigenvectors of slicewise PCA along the third mode
- **est**: estimate of tnsr after compression
- **norm_percent**: the percent of Frobenius norm explained by the approximation
- **fnorm_resid**: the Frobenius norm of the error $\text{fnorm}(\text{est} - \text{tnsr})$

**References**


**Examples**

```r
subject <- faces_tnsr[,8,]
pvdD<-pvd(subject,uranks=rep(46,10),wranks=rep(56,10),a=46,b=56)
```
### rand_tensor

**Tensor with Random Entries**

**Description**

Generate a Tensor with specified modes with iid normal(0,1) entries.

**Usage**

```r
rand_tensor(modes = c(3, 4, 5), drop = FALSE)
```

**Arguments**

- `modes`: the modes of the output Tensor
- `drop`: whether or not modes equal to 1 should be dropped

**Value**

a Tensor object with modes given by `modes`

**Note**

Default `rand_tensor()` generates a 3-Tensor with modes `c(3, 4, 5)`.

**Examples**

```r
rand_tensor()
rand_tensor(c(4,4,4))
rand_tensor(c(10,2,1),TRUE)
```

### rs_fold

**Row Space Folding of Matrix**

**Description**

DEPRECATED. Please see `k_fold`.

**Usage**

```r
rs_fold(mat, m = NULL, modes = NULL)
```

**Arguments**

- `mat`: matrix to be folded
- `m`: the mode corresponding to `rs_unfold`
- `modes`: the original modes of the tensor
Description

DEPRECATED. Please see k_unfold-methods and unfold-methods.

Usage

```r
rs_unfold(tnsr, m)
```

## S4 method for signature 'Tensor'

```r
rs_unfold(tnsr, m = NULL)
```

Arguments

- `tnsr` : Tensor instance
- `m` : mode to be unfolded on

Details

```r
rs_unfold(tnsr, m = NULL)
```

#### show-methods

**Show for Tensor**

Description

Extend show for Tensor

Usage

```r
## S4 method for signature 'Tensor'
show(object)
```

Arguments

- `object` : the Tensor instance
- `...` : additional parameters to be passed into show()

Details

```r
show(object)
```
See Also

print

Examples

tnsr <- rand_tensor()

------------------

<table>
<thead>
<tr>
<th>t-methods</th>
<th>Tensor Transpose</th>
</tr>
</thead>
</table>

Description

Implements the tensor transpose based on block circulant matrices (Kilmer et al. 2013) for 3-tensors.

Usage

## S4 method for signature 'Tensor'

t(x)

Arguments

x       a 3-tensor

Details

t(x)

Value

tensor transpose of x

References


Examples

tnsr <- rand_tensor()

identical(t(tnsr)[,1], t(tnsr)[,1])

identical(t(tnsr)[,2], t(tnsr)[,5])

identical(t(t(tnsr)), tnsr)
Description

Extend tail for Tensor

Usage

```r
## S4 method for signature 'Tensor'
tail(x, ...)
```

Arguments

- `x`: the Tensor instance
- `...`: additional parameters to be passed into `tail()`

Details

```r
tail(x, ...)
```

See Also

- `head-methods`

Examples

```r
tnsr <- rand_tensor()
tail(tnsr)
```

Description

An S4 class for a tensor with arbitrary number of modes. The Tensor class extends the base `array` class to include additional tensor manipulation (folding, unfolding, reshaping, subsetting) as well as a formal class definition that enables more explicit tensor algebra.
Tensor-class

Details

This can be seen as a wrapper class to the base array class. While it is possible to create an instance using `new`, it is also possible to do so by passing the data into `as.tensor`.

Each slot of a Tensor instance can be obtained using `@`. The following methods are overloaded for the Tensor class: `dim-methods`, `head-methods`, `tail-methods`, `print-methods`, `show-methods`, element-wise array operations, array subsetting (extract via `['']`), array subset replacing (replace via `['<-']`), and `tperm-methods`, which is a wrapper around the base `aperm` method.

To sum across any one mode of a tensor, use the function `modeSum-methods`. To compute the mean across any one mode, use `modeMean-methods`.

You can always unfold any Tensor into a matrix, and the `unfold-methods`, `k_unfold-methods`, and `matvec-methods` methods are for that purpose. The output can be kept as a Tensor with 2 modes or a matrix object. The vectorization function is also provided as `vec`. See the attached vignette for a visualization of the different unfoldings.

Conversion from `array/matrix` to Tensor is facilitated via `as.tensor`. To convert from a Tensor instance, simply invoke `@data`.

The Frobenius norm of the Tensor is given by `fnorm-methods`, while the inner product between two Tensors (of equal modes) is given by `innerProd-methods`. You can also sum through any one mode to obtain the K-1 Tensor sum using `modeSum-methods`. `modeMean-methods` provides similar functionality to obtain the K-1 Tensor mean. These are primarily meant to be used internally but may be useful in doing statistics with Tensors.

For Tensors with 3 modes, we also overloaded `t` (transpose) defined by Kilmer et.al (2013). See `t-methods`.

To create a Tensor with i.i.d. random normal(0, 1) entries, see `rand_tensor`.

Slots

- `num_modes` number of modes (integer)
- `modes` vector of modes (integer), aka sizes/extents/dimensions
- `data` actual data of the tensor, which can be `array` or `vector`

Methods

- `[ signature(tnsr = "Tensor"): ...
- `[<- signature(tnsr = "Tensor"): ...
- `matvec` signature(tnsr = "Tensor"): ...
- `dim` signature(tnsr = "Tensor"): ...
- `fnorm` signature(tnsr = "Tensor"): ...
- `head` signature(tnsr = "Tensor"): ...
- `initialize` signature(.Object = "Tensor"): ...
- `innerProd` signature(tnsr1 = "Tensor", tnsr2 = "Tensor"): ...
- `modeMean` signature(tnsr = "Tensor"): ...
modeSum signature(tnsr = "Tensor"): ...

Ops signature(e1 = "array", e2 = "Tensor"): ...

Ops signature(e1 = "numeric", e2 = "Tensor"): ...

Ops signature(e1 = "Tensor", e2 = "array"): ...

Ops signature(e1 = "Tensor", e2 = "numeric"): ...

Ops signature(e1 = "Tensor", e2 = "Tensor"): ...

print signature(tnsr = "Tensor"): ...

k_unfold signature(tnsr = "Tensor"): ...

show signature(tnsr = "Tensor"): ...

t signature(tnsr = "Tensor"): ...

tail signature(tnsr = "Tensor"): ...

unfold signature(tnsr = "Tensor"): ...

tperm signature(tnsr = "Tensor"): ...

image signature(tnsr = "Tensor"): ...

Note

All of the decompositions and regression models in this package require a Tensor input.

Author(s)

James Li <jamesyili@gmail.com>

References


See Also

as.tensor

Examples

tnsr <- rand_tensor()
class(tnsr)
tnsr
print(tnsr)
dim(tnsr)
tnsr@num_modes
tnsr@data
**tperm-methods**

*Mode Permutation for Tensor*

**Description**

Overloads aperm for Tensor class for convenience.

**Usage**

```r
tperm(tnsr, perm, ...)
```

```r
## S4 method for signature 'Tensor'
tperm(tnsr, perm, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnsr</td>
<td>the Tensor instance</td>
</tr>
<tr>
<td>perm</td>
<td>the new permutation of the current modes</td>
</tr>
<tr>
<td>...</td>
<td>additional parameters to be passed into aperm</td>
</tr>
</tbody>
</table>

**Details**

```r
tperm(tnsr, perm=NULL, ...)
```

**Examples**

```r
tnsr <- rand_tensor(c(3, 4, 5))
dim(tperm(tnsr, perm=c(2, 1, 3)))
dim(tperm(tnsr, perm=c(1, 3, 2)))
```

---

**ttl**

*Tensor Times List*

**Description**

Contracted (m-Mode) product between a Tensor of arbitrary number of modes and a list of matrices. The result is folded back into Tensor.

**Usage**

```r
ttl(tnsr, list_mat, ms = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnsr</td>
<td>Tensor object with K modes</td>
</tr>
<tr>
<td>list_mat</td>
<td>a list of matrices</td>
</tr>
<tr>
<td>ms</td>
<td>a vector of modes to contract on (order should match the order of list_mat)</td>
</tr>
</tbody>
</table>
Details

Performs `ttm` repeated for a single Tensor and a list of matrices on multiple modes. For instance, suppose we want to do multiply a Tensor object `tnsr` with three matrices `mat1`, `mat2`, `mat3` on modes 1, 2, and 3. We could do `ttm(ttm(ttm(tnsr, mat1, 1), mat2, 2), 3)`, or we could do `ttl(tnsr, list(mat1, mat2, mat3), c(1, 2, 3))`. The order of the matrices in the list should obviously match the order of the modes. This is a common operation for various Tensor decompositions such as CP and Tucker. For the math on the m-Mode Product, see Kolda and Bader (2009).

Value

Tensor object with K modes

Note

The returned Tensor does not drop any modes equal to 1.

References


See Also

`ttm`

Examples

```r
library(tensr)

# Create a tensor
tnsr <- new("tensor", 3, c(3L, 4L, 5L), data = runif(60))

# Matrices for multiplication
list <- list('mat1' = matrix(runif(30), ncol = 3),
             'mat2' = matrix(runif(40), ncol = 4),
             'mat3' = matrix(runif(50), ncol = 5))

# Perform tensor times matrix
ttl(tnsr, list, c(1, 2, 3))
```

Description

Contracted (m-Mode) product between a Tensor of arbitrary number of modes and a matrix. The result is folded back into Tensor.

Usage

`ttm(tnsr, mat, m = NULL)`
**Arguments**

- `tnsr` Tensor object with K modes
- `mat` input matrix with same number columns as the mth mode of `tnsr`
- `m` the mode to contract on

**Details**

By definition, \( \text{rs\_unfold}(\text{ttm}(\text{tnsr}, \text{mat}), m) = \text{mat} \times \text{rs\_unfold}(\text{tnsr}, m) \), so the number of columns in \( \text{mat} \) must match the mth mode of \( \text{tnsr} \). For the math on the m-Mode Product, see Kolda and Bader (2009).

**Value**

a Tensor object with K modes

**Note**

The mth mode of \( \text{tnsr} \) must match the number of columns in \( \text{mat} \). By default, the returned Tensor does not drop any modes equal to 1.

**References**


**See Also**

ttl, rs_unfold-methods

**Examples**

```r
tnsr <- new("tensor", 3L, c(3L, 4L, 5L), data=runif(60))
mat <- matrix(runif(50), ncol=5)
rtm(tnsr, mat, m=3)
```

---

**Description**

The Tucker decomposition of a tensor. Approximates a K-Tensor using a n-mode product of a core tensor (with modes specified by ranks) with orthogonal factor matrices. If there is no truncation in one of the modes, then this is the same as the MPCA, m pca. If there is no truncation in all the modes (i.e. ranks = tnsr@modes), then this is the same as the HOSVD, hosvd. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below tol, or the max_iter number of iterations has been reached. For more details on the Tucker decomposition, consult Kolda and Bader (2009).
**tucker**

**Usage**

`tucker(tnsr, ranks = NULL, max_iter = 25, tol = 1e-05)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnsr</td>
<td>Tensor with K modes</td>
</tr>
<tr>
<td>ranks</td>
<td>a vector of the modes of the output core Tensor</td>
</tr>
<tr>
<td>max_iter</td>
<td>maximum number of iterations if error stays above tol</td>
</tr>
<tr>
<td>tol</td>
<td>relative Frobenius norm error tolerance</td>
</tr>
</tbody>
</table>

**Details**

Uses the Alternating Least Squares (ALS) estimation procedure also known as Higher-Order Orthogonal Iteration (HOOI). Initialized using a (Truncated-)HOSVD. A progress bar is included to help monitor operations on large tensors.

**Value**

A list containing the following:

- `Z` the core tensor, with modes specified by `ranks`
- `U` a list of orthogonal factor matrices - one for each mode, with the number of columns of the matrices given by `ranks`
- `conv` whether or not `resid < tol` by the last iteration
- `est` estimate of `tnsr` after compression
- `norm_percent` the percent of Frobenius norm explained by the approximation
- `fnorm_resid` the Frobenius norm of the error `fnorm(est-tnsr)`
- `all_resids` vector containing the Frobenius norm of error for all the iterations

**Note**

The length of `ranks` must match `tnsr@num_modes`.

**References**


**See Also**

`hosvd, m pca`

**Examples**

```r
  tnsr <- rand_tensor(c(4,4,4,4))
  tuckerD <- tucker(tnsr, ranks=c(2,2,2,2))
  tuckerD$conv
tuckerD$norm_percent
plot(tuckerD$all_resids)
```
Description

Implements T-MULT based on block circulant matrices (Kilmer et al. 2013) for 3-tensors.

Usage

t_mult(x, y)

Arguments

x  a 3-tensor  
y  another 3-tensor

Details

Uses the Fast Fourier Transform (FFT) speed up suggested by Kilmer et al. 2013 instead of explicitly constructing the block circulant matrix. For the mathematical details of T-MULT, see Kilmer et al. (2013).

Value

tensor product between x and y

Note

This only works (so far) between 3-Tensors.

References


Examples

```  
tnsr <- new("Tensor",3L,c(3L,4L,5L),data=runif(60))  
tnsr2 <- new("Tensor",3L,c(4L,3L,5L),data=runif(60))  
t_mult(tnsr, tnsr2)  
```
t_svd

Tensor Singular Value Decomposition

Description

TSVD for a 3-Tensor. Constructs 3-Tensors \( U, S, V \) such that \( tnsr = t\_mult(t\_mult(U,S),t(V)) \). \( U \) and \( V \) are orthogonal 3-Tensors with orthogonality defined in Kilmer et al. (2013), and \( S \) is a 3-

Usage

\[ t\_svd(tnsr) \]

Arguments

- **tnsr**: 3-Tensor to decompose via TSVD

Value

A list containing the following:

- \( U \): the left orthogonal 3-Tensor
- \( V \): the right orthogonal 3-Tensor
- \( S \): the middle 3-Tensor consisting of face-wise diagonal matrices

Note

Computation involves complex values, but if the inputs are real, then the outputs are also real. Some
loss of precision occurs in the truncation of the imaginary components during the FFT and inverse
FFT.

References

M. Kilmer, K. Braman, N. Hao, and R. Hoover, "Third-order tensors as operators on matrices: a
theoretical and computational framework with applications in imaging". SIAM Journal on Matrix
Analysis and Applications 2013.

See Also

t_mult, t_svd_reconstruct

Examples

```
tnsr <- rand_tensor()
tsvdD <- t_svd(tnsr)
```
**t_svd_reconstruct**  
*Reconstruct Tensor From TSVD*

**Description**
Reconstruct the original 3-Tensor after it has been decomposed into $U$, $S$, $V$ via $t_svd$.

**Usage**
`t_svd_reconstruct(l)`

**Arguments**
- `l` list that is an output from $t_svd$

**Value**
a 3-Tensor

**See Also**
$t_svd$

**Examples**
```r
tnsr <- rand_tensor(c(10,10,10))
tsvdD <- t_svd(tnsr)
1 - fnorm(t_svd_reconstruct(tsvdD)-tnsr)/fnorm(tnsr)
```

---

**unfold-methods**  
*Tensor Unfolding*

**Description**
Unfolds the tensor into a matrix, with the modes in $rs$ onto the rows and modes in $cs$ onto the columns. Note that $c(rs, cs)$ must have the same elements (order doesn't matter) as $x@modes$. Within the rows and columns, the order of the unfolding is determined by the order of the modes. This convention is consistent with Kolda and Bader (2009).

**Usage**
`unfold(tnsr, row_idx, col_idx)`

---

```r
# S4 method for signature 'Tensor'
unfold(tnsr, row_idx = NULL, col_idx = NULL)
```
Arguments

- `tnsr` the Tensor instance
- `row_idx` the indices of the modes to map onto the row space
- `col_idx` the indices of the modes to map onto the column space

Details

For Row Space Unfolding or m-mode Unfolding, see `rs_unfold-methods`. For Column Space Unfolding or `matvec`, see `cs_unfold-methods`.

Kolda and Bader (2009) return the vectorization of the tensor.

Value

Matrix with `prod(row_idx)` rows and `prod(col_idx)` columns

References


See Also

- `k_unfold-methods` and `matvec-methods`

Examples

```r
tnsr <- rand_tensor()
matT3<-unfold(tnsr, row_idx=2, col_idx=c(3,1))
```

unmatvec

Unmatvec Folding of Matrix

Description

The inverse operation to `matvec-methods`, turning a matrix into a Tensor. For a full account of matrix folding/unfolding operations, consult Kolda and Bader (2009).

Usage

`unmatvec(mat, modes = NULL)`

Arguments

- `mat` matrix to be folded into a Tensor
- `modes` the modes of the output Tensor
**Value**

Tensor object with modes given by `modes`

**References**


**See Also**

`matvec-methods`, `fold`, `k_fold`

**Examples**

```r
tnsr <- new("tensor", 3L, c(3L, 4L, 5L), data=runif(60))
matT1 <- matvec(tnsr)
identical(unmatvec(matT1, modes=c(3, 4, 5)), tnsr)
```

---

**vec-methods**

**Tensor Vec**

**Description**

Turns the tensor into a single vector, following the convention that earlier indices vary slower than later indices.

**Usage**

```r
vec(tnsr)
```

```r
# S4 method for signature 'Tensor'
vec(tnsr)
```

**Arguments**

- `tnsr`: the Tensor instance

**Details**

```r
vec(tnsr)
```

**Value**

vector with length `prod(x@modes)`

**References**

Examples

tnsr <- rand_tensor(c(4,5,6,7))
vec(tnsr)

[-methods  

**Extract or Replace Subtensors**

Description

Extends ‘[’ and ‘[<-’ from the base array class for the Tensor class. Works exactly as it would for the base ‘array’ class.

Usage

```r
## S4 method for signature 'Tensor'
x[i, j, ..., drop = TRUE]
```

```r
## S4 replacement method for signature 'Tensor'
x[i, j, ...] <- value
```

Arguments

- `x`: Tensor to be subset
- `i, j, ...`: indices that specify the extents of the sub-tensor
- `drop`: whether or not to reduce the number of modes to exclude those that have ‘1’ as the mode
- `value`: either vector, matrix, or array that will replace the subtensor

Details

```r
x[i,j,...,drop=TRUE]
```

Value

an object of class Tensor

Examples

```r
tnsr <- rand_tensor()
tnsr[1,2,3]
tnsr[3,1,]
tnsr[,]5]
tnsr[,5,drop=FALSE]

tnsr[1,2,3] <- 3; tnsr[1,2,3]
tnsr[3,1,] <- rep(0,5); tnsr[3,1,]
tnsr[,2,] <- matrix(0,nrow=3,ncol=5); tnsr[,2,]
```
Index

+Topic datasets
  faces_tnsr, 7
  [,Tensor-method ([methods], 41
  [-methods], 41
  [<-,Tensor-method ([methods], 41
  as.tensor, 4, 30, 31
  cp, 3, 4
  cs_fold, 6
  cs_unfold (cs_unfold-methods), 6
  cs_unfold, Tensor-method
    (cs_unfold-methods), 6
  cs_unfold-methods, 6
  dim, Tensor-method (dim-methods), 7
  dim-methods, 7
  extract, Tensor-method ([methods], 41
  faces_tnsr, 7, 23
  fnorm (fnorm-methods), 8
  fnorm, Tensor-method (fnorm-methods), 8
  fnorm-methods, 8
  fold, 3, 9, 16, 40
  hadamard_list, 3, 10, 15
  head, Tensor-method (head-methods), 10
  head-methods, 10
  hosvd, 3, 11, 22, 34, 35
  initialize, Tensor-method
    (initialize-methods), 12
  initialize-methods, 12
  innerProd (innerProd-methods), 13
  innerProd, Tensor, Tensor-method
    (innerProd-methods), 13
  innerProd-methods, 13
  k_fold, 3, 9, 16, 26, 40
  k_unfold, 3
  k_unfold (k_unfold-methods), 17
  k_unfold, Tensor-method
    (k_unfold-methods), 17
  k_unfold-methods, 17
  khatri_rao, 3, 13, 14
  khatri_rao_list, 3, 10, 14, 14, 15
  kronecker, 14, 15
  kronecker_list, 3, 10, 15
  matvec, 3
  matvec (matvec-methods), 18
  matvec, Tensor-method (matvec-methods), 18
  matvec-methods, 18
  modeMean, 20
  modeMean (modeMean-methods), 19
  modeMean, Tensor-method
    (modeMean-methods), 19
  modeMean-methods, 19
  modeSum, 19
  modeSum (modeSum-methods), 20
  modeSum, Tensor-method
    (modeSum-methods), 20
  modeSum-methods, 20
  mpca, 3, 21, 24, 34, 35
  Ops, array, Tensor-method (Ops-methods), 22
  Ops, numeric, Tensor-method
    (Ops-methods), 22
  Ops, Tensor, array-method (Ops-methods), 22
  Ops, Tensor, numeric-method
    (Ops-methods), 22
  Ops, Tensor, Tensor-method (Ops-methods), 22
  Ops-methods, 22
  plot_orl, 8, 23
  print, 28
print, Tensor-method (print-methods), 24
print-methods, 24
pvd, 3, 24

rand_tensor, 26, 30
rs_fold, 26
rs_unfold (rs_unfold-methods), 27
rs_unfold, Tensor-method
  (rs_unfold-methods), 27
rs_unfold-methods, 27
rTensor (rTensor-package), 3
rTensor-package, 3

show, 24
show, Tensor-method (show-methods), 27
show-methods, 27

t, Tensor-method (t-methods), 28
t-methods, 28
t_mult, 3, 36, 37
t_svd, 3, 37, 38
t_svd_reconstruct, 3, 37, 38
tail, Tensor-method (tail-methods), 29
tail-methods, 29
Tensor (Tensor-class), 29
Tensor-class, 29
tperm (tperm-methods), 32
tperm, Tensor-method (tperm-methods), 32
tperm-methods, 32
ttl, 3, 32, 34
ttm, 3, 33, 33
tucker, 3, 5, 12, 21, 22, 24, 34

unfold (unfold-methods), 38
unfold, Tensor-method (unfold-methods), 38
unfold-methods, 38
unmatvec, 3, 6, 9, 16, 39

vec (vec-methods), 40
vec, Tensor-method (vec-methods), 40
vec-methods, 40