Package ‘rTensor’
October 14, 2022

Type Package
Title Tools for Tensor Analysis and Decomposition
Version 1.4.8
Author James Li and Jacob Bien and Martin Wells
Maintainer Koki Tsuyuzaki <k.t.the-answer@hotmail.co.jp>
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 Hadamard product for a list of matrices.
License GPL (>= 2)
Depends R (>= 2.10.0)
Imports methods
Date 2021-05-14
URL https://github.com/rikenbit/rTensor
RoxygenNote 6.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2021-05-15 06:20:10 UTC

R topics documented:

rTensor-package ................................................................. 3
R topics documented:

<table>
<thead>
<tr>
<th>R Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.tensor</td>
<td>4</td>
</tr>
<tr>
<td>cp</td>
<td>4</td>
</tr>
<tr>
<td>cs_fold</td>
<td>6</td>
</tr>
<tr>
<td>cs_unfold-methods</td>
<td>6</td>
</tr>
<tr>
<td>dim-methods</td>
<td>7</td>
</tr>
<tr>
<td>fnorm-methods</td>
<td>7</td>
</tr>
<tr>
<td>fold</td>
<td>8</td>
</tr>
<tr>
<td>hadamard_list</td>
<td>9</td>
</tr>
<tr>
<td>head-methods</td>
<td>10</td>
</tr>
<tr>
<td>hosvd</td>
<td>10</td>
</tr>
<tr>
<td>initialize-methods</td>
<td>12</td>
</tr>
<tr>
<td>innerProd-methods</td>
<td>12</td>
</tr>
<tr>
<td>khatri_rao</td>
<td>13</td>
</tr>
<tr>
<td>khatri_rao_list</td>
<td>14</td>
</tr>
<tr>
<td>kronecker_list</td>
<td>14</td>
</tr>
<tr>
<td>k_fold</td>
<td>15</td>
</tr>
<tr>
<td>k_unfold-methods</td>
<td>16</td>
</tr>
<tr>
<td>load_orl</td>
<td>17</td>
</tr>
<tr>
<td>matvec-methods</td>
<td>18</td>
</tr>
<tr>
<td>modeMean-methods</td>
<td>19</td>
</tr>
<tr>
<td>modeSum-methods</td>
<td>20</td>
</tr>
<tr>
<td>m pca</td>
<td>21</td>
</tr>
<tr>
<td>Ops-methods</td>
<td>22</td>
</tr>
<tr>
<td>plot_orl</td>
<td>23</td>
</tr>
<tr>
<td>print-methods</td>
<td>23</td>
</tr>
<tr>
<td>pvd</td>
<td>24</td>
</tr>
<tr>
<td>rand_tensor</td>
<td>25</td>
</tr>
<tr>
<td>rs_fold</td>
<td>26</td>
</tr>
<tr>
<td>rs_unfold-methods</td>
<td>26</td>
</tr>
<tr>
<td>show-methods</td>
<td>27</td>
</tr>
<tr>
<td>t-methods</td>
<td>27</td>
</tr>
<tr>
<td>tail-methods</td>
<td>28</td>
</tr>
<tr>
<td>Tensor-class</td>
<td>29</td>
</tr>
<tr>
<td>tperm-methods</td>
<td>31</td>
</tr>
<tr>
<td>ttl</td>
<td>32</td>
</tr>
<tr>
<td>ttm</td>
<td>33</td>
</tr>
<tr>
<td>tucker</td>
<td>34</td>
</tr>
<tr>
<td>t_mult</td>
<td>35</td>
</tr>
<tr>
<td>t_svd</td>
<td>36</td>
</tr>
<tr>
<td>t_svd_reconstruct</td>
<td>37</td>
</tr>
<tr>
<td>unfold-methods</td>
<td>38</td>
</tr>
<tr>
<td>unmatvec</td>
<td>39</td>
</tr>
<tr>
<td>vec-methods</td>
<td>40</td>
</tr>
<tr>
<td>[-methods</td>
<td>41</td>
</tr>
</tbody>
</table>

Index 42
Description

This package is centered around the 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 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:

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

rTensor also provides a set functions for tensors multiplication:

- **ttm**  Tensor times matrix, aka m-mode product
- **ttl**  Tensor times list (of matrices)
- **t_mult**  Tensor product based on block circulant unfolding; only implemented for a pair of 3-Tensors

...as well as for matrices:

- **hadamard_list**  Computes the Hadamard (element-wise) product of a list of matrices
- **kronecker_list**  Computes the Kronecker product of a list of matrices
- **khatri_rao**  Computes the Khatri-Rao product of two matrices
- **khatri_rao_list**  Computes the Khatri-Rao product of a list of matrices
- **fold**  General folding of a matrix into a tensor
- **k_fold**  Inverse operation for k_unfold
- **unmatvec**  Inverse operation for matvec

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

#From vector
vec <- runif(3); vecT <- as.tensor(vec); vecT

#From matrix
mat <- matrix(runif(2*3), nrow=2, ncol=3)
matT <- as.tensor(mat); matT

#From array
indices <- c(2,3,4)
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 numComponents * tnsr@numModes vectors in the output, stored in tnsr@numModes matrices, each with numComponents columns. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below tol, or the maxIter number of iterations has been reached. For more details on CP decomposition, consult Kolda and Bader (2009).
Usage

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

Arguments

\texttt{tnsr} \quad \text{Tensor with K modes}
\texttt{num\_components} \quad \text{the number of rank-1 K-Tensors to use in approximation}
\texttt{max\_iter} \quad \text{maximum number of iterations if error stays above tol}
\texttt{tol} \quad \text{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

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

References


See Also

tucker

Examples

```r
### How to retrieve faces_tnsr from figshare
# faces_tnsr <- load_orl()
# subject <- faces_tnsr[,14,]
dummy_faces_tnsr <- rand_tensor(c(92,112,40,10))
subject <- dummy_faces_tnsr[,14,]
D <- cp(subject, num\_components=3)
D$conv
D$norm\_percent
plot(D$all\_resids)
```
cs_fold

*Column Space Folding of Matrix*

**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` (DEPRECATED)
- `modes`: the original modes of the tensor (DEPRECATED)

---

cs_unfold-methods

*Tensor Column Space Unfolding*

**Description**

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

**Usage**

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

```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

## S4 method for signature 'Tensor'
dim(x)

Arguments

x the Tensor instance

Details

dim(x)

Value

an integer vector of the modes associated with x

Examples

tnsr <- rand_tensor()
dim(tnsr)

fnorm-methods

Tensor Frobenius Norm

Description

Returns the Frobenius norm of the Tensor instance.

Usage

fnorm(tnsr)

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

Arguments

tnsr the Tensor instance
Details

fnorm(tnsr)

Value

numeric Frobenius norm of x

Examples

tnsr <- rand_tensor()
fnorm(tnsr)

fold General Folding of Matrix

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

hadamard_list

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

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

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 for Tensor

## Usage

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

## Arguments

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

## Details

`head(x, ...)`

## See Also

`tail-methods`

## Examples

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

# (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

```r
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

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

```r
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)

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

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

Examples

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

kroncker_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

kroncker_list(L)

Examples

smalllist <- list('mat1' = matrix(runif(12),ncol=4),
'mat2' = matrix(runif(12),ncol=4),
'mat3' = matrix(runif(12),ncol=4))

**k_fold**

**Arguments**

- `L` list of matrices or vectors

**Value**

matrix that is the Kronecker product

**See Also**

hadamard_list, khatri_rao_list, kronecker

**Examples**

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

---

**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**

`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

\texttt{k_unfold-methods, fold, unmatvec}

Examples

\begin{verbatim}
 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)
\end{verbatim}

---

\textbf{k_unfold-methods} \hspace{1cm} \textit{Tensor k-mode Unfolding}

Description

Unfolding of a tensor by mapping the kth mode (specified through parameter \texttt{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

\begin{verbatim}
 k_unfold(tnsr, m)
\end{verbatim}

## S4 method for signature 'Tensor'
 k_unfold(tnsr, m = NULL)

Arguments

- \texttt{tnsr} the Tensor instance
- \texttt{m} the index of the mode to unfold on

Details

\begin{verbatim}
 k_unfold(tnsr,m=NULL)
\end{verbatim}

Value

matrix with \texttt{x@modes[m]} rows and \texttt{prod(x@modes[-m])} columns

References


See Also

\texttt{matvec-methods} and \texttt{unfold-methods}
**load_orl**

**Examples**

```r
tnsr <- rand_tensor()
matT2<-rs_unfold(tnsr,m=2)
```

---

**load_orl**

**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. The data is now stored in figshare [https://ndownloader.figshare.com/files/28005669](https://ndownloader.figshare.com/files/28005669)

**Usage**

```r
load_orl()
```

**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**


**References**


**See Also**

`plot_orl`
matvec-methods

Tensor Matvec Unfolding

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

tnsr <- rand_tensor()
modeMean(tnsr,1,drop=TRUE)
**Tensor Sum Across Single Mode**

**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)
```

```r
# S4 method for signature 'Tensor'
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**

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

**See Also**

`modeMean`

**Examples**

```r
tnsr <- rand_tensor()
modeSum(tnsr,3,drop=TRUE)
```
Multilinear Principal Components Analysis

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
mpca(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

### How to retrieve faces_tnsr from figshare

```r
# faces_tnsr <- load_orl()
# subject <- faces_tnsr[,,21,]
dummy_faces_tnsr <- rand_tensor(c(92,112,40,10))
subject <- dummy_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**

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

**Arguments**

- `e1` left-hand object
- `e2` right-hand object

**Examples**

```r
tnsr <- rand_tensor(c(3,4,5))
tnsr2 <- rand_tensor(c(3,4,5))
tnsrsum <- tnsr + tnsr2
tnsrdiff <- tnsr - tnsr2
tnsrelemprod <- tnsr * tnsr2
tnsrelemquot <- tnsr / tnsr2
for (i in 1:3L){
  for (j in 1:4L){
    i
  }
}
for (k in 1:5L){
    stopifnot(tnsrsum@data[i,j,k]==tnsr@data[i,j,k]+tnsr2@data[i,j,k])
    stopifnot(tnsrdiff@data[i,j,k]==(tnsr@data[i,j,k]-tnsr2@data[i,j,k]))
    stopifnot(tnsrelempod@data[i,j,k]==tnsr@data[i,j,k]*tnsr2@data[i,j,k])
    stopifnot(tnsrelemquot@data[i,j,k]==tnsr@data[i,j,k]/tnsr2@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. The data is now stored in figshare [https://ndownloader.figshare.com/files/28005669](https://ndownloader.figshare.com/files/28005669)

Usage

plot_orl(subject = 1, condition = 1)

Arguments

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

References


---

print-methods

Print for Tensor

Description

Extend print for Tensor

Usage

```r
## 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 \(2*n_3 + 2\) 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 \(n_3 + 2\) 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 \( P = U \times t(U) \), 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 \( D = W \times t(W) \), where \( W \) is constructed by stacking the truncated right eigenvectors of slicewise PCA along the third mode

\( \text{est} \) estimate of \( \text{tnsr} \) after compression

\( \text{norm\_percent} \) the percent of Frobenius norm explained by the approximation

\( \text{fnorm\_resid} \) the Frobenius norm of the error \( \text{fnorm}(\text{est}-\text{tnsr}) \)

References


Examples

```r
### How to retrieve faces_tnsr from figshare
# faces_tnsr <- load_orl()
# subject <- faces_tnsr[,8,]
dummy_faces_tnsr <- rand_tensor(c(92,112,40,10))
subject <- dummy_faces_tnsr[,8,]
pvdD <- pvd(subject, uranks=rep(46,10), wranks=rep(56,10), a=46, b=56)
plot(pvdD$fnorm_resid)
```

---

rand_tensor

**Tensor with Random Entries**

Description

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

Usage

`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

---

**rs_unfold-methods**

*Tensor Row Space Unfolding*

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

Details

```
show(object)
```

See Also

`print`

Examples

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

t-methods

Tensor Transpose

Description

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

Usage

```r
## 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@Data[,1],t(tnsr@Data[,1]))
identical(t(tnsr@Data[,2],t(tnsr@Data[,5]))
identical(t(t(tnsr)),tnsr)

Description

Extend tail for Tensor

Usage

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

Arguments

x the Tensor instance

... additional parameters to be passed into tail()

Details

tail(x,...)

See Also

head-methods

Examples

tnsr <- rand_tensor()
tail(tnsr)
**Tensor-class**

Tensor-class

S4 Class for a Tensor

### 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.

### 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
tperm-methods

Examples

```r
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, ...)
```

### Arguments

- `tnsr`  
  the Tensor instance
- `perm`  
  the new permutation of the current modes
- `...`  
  additional parameters to be passed into `aperm`

### 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**

- `tnsr`: Tensor object with K modes
- `list_mat`: a list of matrices
- `ms`: a vector of modes to contract on (order should match the order of `list_mat`)

**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

tnsr <- new("Tensor",3L,c(3L,4L,5L),data=runif(60))
lizt <- list('mat1' = matrix(runif(30),ncol=3),
              'mat2' = matrix(runif(40),ncol=4),
              'mat3' = matrix(runif(50),ncol=5))
ttl(tnsr,lizt,ms=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, rs_unfold(ttm(tnsr,mat),m) = mat%*%rs_unfold(tnsr,m), so the number of columns in mat must match the mth mode of 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 tnsr must match the number of columns in mat. By default, the returned Tensor does not drop any modes equal to 1.

References


See Also

ttl, rs_unfold-methods
Examples

tnsr <- new("Tensor",3L,c(3L,4L,5L),data=runif(60))
mat <- matrix(runif(50),ncol=5)
ttm(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 \texttt{ranks}) with orthogonal factor matrices. If there is no truncation
in one of the modes, then this is the same as the MPCA, \texttt{mpca}. If there is no truncation in all the
modes (i.e. \texttt{ranks = tnsr@modes}), then this is the same as the HOSVD, \texttt{hosvd}. This is an iterative
algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten
below \texttt{tol}, or the \texttt{max_iter} number of iterations has been reached. For more details on the Tucker
decomposition, consult Kolda and Bader (2009).

Usage

\texttt{tucker(tnsr, ranks = NULL, max_iter = 25, tol = 1e-05)}

Arguments

\begin{itemize}
  \item \texttt{tnsr} \hspace{1cm} Tensor with K modes
  \item \texttt{ranks} \hspace{1cm} a vector of the modes of the output core Tensor
  \item \texttt{max_iter} \hspace{1cm} maximum number of iterations if error stays above \texttt{tol}
  \item \texttt{tol} \hspace{1cm} relative Frobenius norm error tolerance
\end{itemize}

Details

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

Value

a list containing the following:

\begin{itemize}
  \item \texttt{Z} \hspace{1cm} the core tensor, with modes specified by \texttt{ranks}
  \item \texttt{U} \hspace{1cm} a list of orthogonal factor matrices - one for each mode, with the number of columns of the matrices
given by \texttt{ranks}
  \item \texttt{conv} \hspace{1cm} whether or not \texttt{resid < tol} by the last iteration
  \item \texttt{est} \hspace{1cm} estimate of \texttt{tnsr} after compression
  \item \texttt{norm_percent} \hspace{1cm} the percent of Frobenius norm explained by the approximation
  \item \texttt{fnorm_resid} \hspace{1cm} the Frobenius norm of the error \texttt{fnorm(est-tnsr)}
  \item \texttt{all_resids} \hspace{1cm} vector containing the Frobenius norm of error for all the iterations
\end{itemize}
**t_mult**

**Note**

The length of ranks must match tnsr@num_modes.

**References**


**See Also**

hosvd, mpca

**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)
```

---

**t_mult**

**Tensor Multiplication (T-MULT)**

**Description**

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

**Usage**

```r
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)

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-Tensor consists of facewise diagonal matrices. For more details on the TSVD, consult Kilmer et al. (2013).

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

**t_svd_reconstruct**

---

**See Also**

`t_mult`, `t_svd_reconstruct`

**Examples**

```r
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**

```r
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)
```
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)
```

## 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`. `vec-methods` returns the vectorization of the tensor.

```
unfold(tnsr,row_idx=NULL,col_idx=NULL)
```

Value

matrix with prod(row_idx) rows and prod(col_idx) columns

References


See Also

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

Examples

```
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

tnsr <- new("Tensor", 3L, c(3L, 4L, 5L), data=runif(60))
matT1<-matvec(tnsr)
identical(unmatvec(matT1, modes=c(3,4,5)), tnsr)
**Description**

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

**Usage**

```r
vec(tnsr)
```

## S4 method for signature 'Tensor'

```r
vec(tnsr)
```

**Arguments**

- `tnsr` the Tensor instance

**Details**

```r
vec(tnsr)
```

**Value**

vector with length `prod(x@modes)`

**References**


**Examples**

```r
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

## S4 method for signature 'Tensor'

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

## 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

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

Value

an object of class Tensor

Examples

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

* datasets
  *load_orl*, 17
  [,Tensor-method ([methods], 41
  [-methods], 41
  [<-,Tensor-method ([methods], 41
  as.tensor, 4, 29, 30
  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
  fnorm (fnorm-methods), 7
  fnorm, Tensor-method (fnorm-methods), 7
  fnorm-methods, 7
  fold, 3, 8, 15, 16, 39
  hadamard_list, 3, 9, 15
  head, Tensor-method (head-methods), 10
  head-methods, 10
  hosvd, 3, 10, 22, 34, 35
  initialize, Tensor-method
    (initialize-methods), 12
  initialize-methods, 12
  innerProd (innerProd-methods), 12
  innerProd, Tensor-method
    (innerProd-methods), 12
  innerProd-methods, 12
  k_fold, 3, 9, 15, 26, 39
  k_unfold, 3
  k_unfold (k_unfold-methods), 16
  k_unfold, Tensor-method
    (k_unfold-methods), 16
  k_unfold-methods, 16
  khatri_rao, 3, 13, 14
  khatri_rao_list, 3, 9, 13, 14, 15
  kronecker, 13, 15
  kronecker_list, 3, 9, 14
  load_orl, 17
  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
  mpga, 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, 17, 23
INDEX

print, 27
print, Tensor-method (print-methods), 23
print-methods, 23
pvd, 3, 24
rand_tensor, 25, 29
rs_fold, 26
rs_unfold (rs_unfold-methods), 26
rs_unfold, Tensor-method
   (rs_unfold-methods), 26
rs_unfold-methods, 26
rTensor (rTensor-package), 3
rTensor-package, 3
show, 24
show, Tensor-method (show-methods), 27
show-methods, 27
t, Tensor-method (t-methods), 27
t-methods, 27
t_mult, 3, 35, 37
t_svd, 3, 36, 37
t_svd_reconstruct, 3, 37, 37
tail, Tensor-method (tail-methods), 28
tail-methods, 28
Tensor (Tensor-class), 29
tensor-class, 29
tperm (tperm-methods), 31
tperm, Tensor-method (tperm-methods), 31
tperm-methods, 31
ttl, 3, 32, 33
ttm, 3, 32, 33
tucker, 3, 5, 11, 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