# Package ‘rTensor’

May 9, 2026

**Type** Package

**Title** Tools for Tensor Analysis and Decomposition

**Version** 1.5.0

**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, Tucker, INDSCAL, RESCAL, DEDICOM, PARAFAC2, and 2DLDA. 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** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 2.10.0)

**Imports** methods

**Suggests** testthat

**Date** 2026-04-12

**URL** https://github.com/rikenbit/rTensor

**Config/testthat/edition** 2

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** James Li [aut],
Jacob Bien [aut],
Martin Wells [aut],
Koki Tsuyuzaki [cre, ctb]

**Maintainer** Koki Tsuyuzaki <k.t.the-answer@hotmail.co.jp>

**Repository** CRAN

**Date/Publication** 2026-05-09 14:41:26 UTC

# Contents

---

`rTensor-package` *Tools for tensor analysis and decomposition*

---

## 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.

## Author(s)

James Li <jamesyili@gmail.com>, Jacob Bien, and Martin T. Wells

---

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/PARAFRAC. 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_compoents *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

```
cp(tnsr, num_components = NULL, max_iter = 25, 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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### See Also

tucker

### Examples

```
### 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,]
cpD <- cp(subject, num_components=3)
cpD$conv
cpD$norm_percent
plot(cpD$all_resids)
```

---

## cs_fold *Column Space Folding of Matrix*

---

### Description

DEPRECATED. Please see unmatvec

### Usage

```
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 *Tensor Column Space Unfolding*

---

### Description

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

### Usage

```
cs_unfold(tnsr, m)

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

### Arguments

| | |
|---|---|
| tnsr | Tensor instance |
| m | mode to be unfolded on |

### Details

cs_unfold(tnsr,m=NULL)

---

dedicom *DEDICOM Decomposition*

---

### Description

Decomposition into Directional Components (DEDICOM) of a 3-Tensor. Decomposes a 3-Tensor into a shared factor matrix A, a shared asymmetric relation matrix R, and slice-specific diagonal weight matrices D_k such that each frontal slice X_k = A %*% D_k %*% R %*% D_k %*% t(A). Uses the Alternating Least Squares (ALS) estimation procedure. For more details on DEDICOM, consult Bader et al. (2007).

### Usage

```
dedicom(tnsr, num_components = NULL, max_iter = 25, tol = 1e-05)
```

### Arguments

| | |
|---|---|
| tnsr | 3-Tensor with square frontal slices |
| num_components | the number of components for the decomposition |
| max_iter | maximum number of iterations if error stays above tol |
| tol | relative Frobenius norm error tolerance |

### Details

A progress bar is included to help monitor operations on large tensors. The input tensor must be 3-dimensional with square frontal slices (i.e. the first two modes must be equal).

### Value

a list containing the following:

A the shared factor matrix with num_components columns

R the asymmetric relation matrix of size num_components by num_components

D a list of diagonal weight matrices, one for each frontal slice

conv whether or not resid < tol by the last iteration

est estimate of tnsr after decomposition

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 first two modes of `tnsr` must be equal (square frontal slices).

### References

B. Bader, R. Harshman, T. Kolda, "Temporal analysis of semantic graphs using ASALSAN". Proceedings of the 7th IEEE International Conference on Data Mining 2007.

### See Also

`rescal`, `indscal`

### Examples

```
tnsr <- rand_tensor(c(5,5,3))
dedicomD <- dedicom(tnsr, num_components=2)
dedicomD$conv
dedicomD$norm_percent
plot(dedicomD$all_resids)
```

---

## `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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### 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-methods *Head for Tensor*

---

## Description

Extend head for Tensor

## Usage

```
## 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

```
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 speficied 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

L. Lathauwer, B.Moor, J. Vanderwalle "A multilinear singular value decomposition". Journal of Matrix Analysis and Applications 2000.

## 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)
```

---

`indscal` *INDSCAL Decomposition*

---

## Description

Individual Differences Scaling (INDSCAL) decomposition of a 3-Tensor. Decomposes a symmetric 3-Tensor into a shared factor matrix `A` and slice-specific diagonal weight matrices `D_k` such that each frontal slice `X_k = A %*% D_k %*% t(A)`. Uses the Alternating Least Squares (ALS) estimation procedure. For more details on INDSCAL, consult Carroll and Chang (1970).

## Usage

```
indscal(tnsr, num_components = NULL, max_iter = 25, tol = 1e-05)
```

## Arguments

| | |
|---|---|
| `tnsr` | 3-Tensor with square frontal slices |
| `num_components` | the number of components for the decomposition |
| `max_iter` | maximum number of iterations if error stays above `tol` |
| `tol` | relative Frobenius norm error tolerance |

## Details

A progress bar is included to help monitor operations on large tensors. The input tensor must be 3-dimensional with square frontal slices (i.e. the first two modes must be equal).

## Value

a list containing the following:

`A` the shared factor matrix with `num_components` columns

`D` a list of diagonal weight matrices, one for each frontal slice

`conv` whether or not `resid` < `tol` by the last iteration

`est` estimate of `tnsr` after decomposition

`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 first two modes of `tnsr` must be equal (square frontal slices).

### References

J. Carroll, J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition". Psychometrika 1970.

### See Also

`cp`, `rescal`

### Examples

```
tnsr <- rand_tensor(c(4,4,3))
# make symmetric frontal slices
for(k in 1:3) tnsr[,,k] <- (tnsr[,,k] + t(tnsr[,,k]@data)) / 2
indscalD <- indscal(tnsr, num_components=2)
indscalD$conv
indscalD$norm_percent
plot(indscalD$all_resids)
```

---

## `initialize-methods` *Initializes a Tensor instance*

---

### Description

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

### Usage

```
## 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)
```

### 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

```
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

```
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))
```

---

## 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

```
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

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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

## See Also

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

## Examples

```
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)
```

## k_unfold-methods *Tensor k-mode Unfolding*

### 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

```
k_unfold(tnsr, m)

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

### Arguments

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

### Details

```
k_unfold(tnsr,m=NULL)
```

### Value

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

### References

T. Kolda and B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### See Also

matvec-methods and unfold-methods

### Examples

```
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

### Usage

```
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 correpsonds to the lighting.

### Source

https://www.kaggle.com/kasikrit/att-database-of-faces

### References

AT&T Laboratories Cambridge. https://www.kaggle.com/kasikrit/att-database-of-faces

F. Samaria, A. Harter, "Parameterisation of a Stochastic Model for Human Face Identification". IEEE Workshop on Applications of Computer Vision 1994.

### 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

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

`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)
```

---

## modeSum-methods *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

```
modeSum(tnsr, m, drop)

## 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

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

### Value

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

### See Also

modeMean

## Examples

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

---

mpca *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

```
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 orthgonal 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

H. Lu, K. Plataniotis, A. Venetsanopoulos, "Mpca: Multilinear principal component analysis of tensor objects". IEEE Trans. Neural networks, 2008.

## See Also

tucker, hosvd

## Examples

```
### How to retrieve faces_tnsr from figshare
# 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

```
## 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))
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){
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(tnsrelemprod@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])
}
}
}
```

---

# parafac2 *PARAFAC2 Decomposition*

---

## Description

PARAFAC2 decomposition of a 3-Tensor. Decomposes a 3-Tensor into slice-specific orthogonal matrices `H_k`, a shared profile matrix `B`, a mode-3 factor matrix `C`, and slice-specific diagonal weight matrices `D_k` such that each frontal slice `X_k = H_k %*% B %*% D_k %*% t(C)`. The key PARAFAC2 constraint is that `t(H_k) %*% H_k` is constant across all slices. Uses the ALS estimation procedure of Kiers et al. (1999). For more details on PARAFAC2, consult Harshman (1972).

## Usage

```
parafac2(tnsr, num_components = NULL, max_iter = 25, tol = 1e-05)
```

## Arguments

| | |
|---|---|
| `tnsr` | 3-Tensor to decompose |
| `num_components` | the number of components for the decomposition |
| `max_iter` | maximum number of iterations if error stays above `tol` |
| `tol` | relative Frobenius norm error tolerance |

## Details

A progress bar is included to help monitor operations on large tensors. The input tensor must be 3-dimensional. Unlike `cp`, PARAFAC2 allows different row spaces per slice while constraining their cross-products to be equal.

## Value

a list containing the following:

- H a list of orthogonal matrices (one per frontal slice), each of size mode1 by num_components
- B the shared profile matrix of size num_components by num_components
- C the mode-3 factor matrix of size mode2 by num_components
- D a list of diagonal weight matrices, one for each frontal slice
- conv whether or not resid < tol by the last iteration
- est estimate of tnsr after decomposition
- 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 input tensor must be 3-dimensional.

## References

R. Harshman, "PARAFAC2: Mathematical and technical notes". UCLA Working Papers in Phonetics 1972.

H. Kiers, J. ten Berge, R. Bro, "PARAFAC2 - Part I. A direct fitting algorithm for the PARAFAC2 model". Journal of Chemometrics 1999.

## See Also

cp, tucker

## Examples

```
tnsr <- rand_tensor(c(6,5,4))
pf2D <- parafac2(tnsr, num_components=2)
pf2D$conv
pf2D$norm_percent
plot(pf2D$all_resids)
```

## 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

### Usage

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

### Arguments

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

### References

AT&T Laboratories Cambridge. https://www.kaggle.com/kasikrit/att-database-of-faces

F. Samaria, A. Harter, "Parameterisation of a Stochastic Model for Human Face Identification". IEEE Workshop on Applications of Computer Vision 1994.

## print-methods — *Print for Tensor*

### 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*n3 + 2` parameters to specified the final ranks of P, V, and D, where n3 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 `n3 + 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-leve 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 `fnorm(est-tnsr)`

## References

C. Crainiceanu, B. Caffo, S. Luo, V. Zipunnikov, N. Punjabi, "Population value decomposition: a framework for the analysis of image populations". Journal of the American Statistical Association, 2013.

## Examples

```
### 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

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

rescal *RESCAL Decomposition*

## Description

RESCAL decomposition of a 3-Tensor for relational data. Decomposes a 3-Tensor into a shared entity factor matrix `A` and slice-specific core matrices `R_k` such that each frontal slice `X_k = A %*% R_k %*% t(A)`. Uses the Alternating Least Squares (ALS) estimation procedure. For more details on RESCAL, consult Nickel et al. (2011).

## Usage

```
rescal(tnsr, num_components = NULL, max_iter = 25, tol = 1e-05)
```

## Arguments

| | |
|---|---|
| `tnsr` | 3-Tensor with square frontal slices |
| `num_components` | the number of components for the decomposition |
| `max_iter` | maximum number of iterations if error stays above `tol` |
| `tol` | relative Frobenius norm error tolerance |

## Details

A progress bar is included to help monitor operations on large tensors. The input tensor must be 3-dimensional with square frontal slices (i.e. the first two modes must be equal). Unlike `indscal`, the core matrices `R_k` are dense (not restricted to be diagonal), which allows modeling of asymmetric relations.

## Value

a list containing the following:

`A` the shared entity factor matrix with `num_components` columns

`R` a list of core matrices (one per frontal slice), each of size `num_components` by `num_components`

`conv` whether or not `resid` < `tol` by the last iteration

`est` estimate of `tnsr` after decomposition

`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 first two modes of `tnsr` must be equal (square frontal slices).

### References

M. Nickel, V. Tresp, H. Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data". Proceedings of the 28th International Conference on Machine Learning 2011.

### See Also

indscal, dedicom

### Examples

```
tnsr <- rand_tensor(c(5,5,3))
rescalD <- rescal(tnsr, num_components=2)
rescalD$conv
rescalD$norm_percent
plot(rescalD$all_resids)
```

---

## rs_fold — *Row Space Folding of Matrix*

---

### Description

DEPRECATED. Please see k_fold.

### Usage

```
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

```
rs_unfold(tnsr, m)

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

## Arguments

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

## Details

`rs_unfold(tnsr,m=NULL)`

---

`show-methods` *Show for Tensor*

---

## Description

Extend show for Tensor

## Usage

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

## Arguments

| | |
|---|---|
| `object` | the Tensor instance |

## Details

`show(object)`

## See Also

`print`

## Examples

```
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

```
## S4 method for signature 'Tensor'
t(x)
```

### Arguments

| | |
|---|---|
| x | a 3-tensor |

### Details

t(x)

### Value

tensor transpose of x

### 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.

### 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)
```

## tail-methods *Tail for Tensor*

### 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 *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 tenor, 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

James Li, Jacob Bien, Martin T. Wells (2018). rTensor: An R Package for Multidimensional Array (Tensor) Unfolding, Multiplication, and Decomposition. Journal of Statistical Software, 87(10), 1-31. URL http://www.jstatsoft.org/v087/i10/.

## 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

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

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

### Arguments

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

### Details

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

### Examples

```
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

```
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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### 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))
```

---

## ttm *Tensor Times Matrix (m-Mode Product)*

---

### 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 `m`th 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 `m`th 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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### 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)
```

---

## tucker *Tucker Decomposition*

---

### 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, `mpca`. 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).

## Usage

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

## Arguments

| | |
|---|---|
| tnsr | Tensor with K modes |
| ranks | a vector of the modes of the output core Tensor |
| 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 also known as Higher-Order Orthogonal Iteration (HOOI). Intialized 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 orthgonal 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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

## See Also

hosvd, mpca

## Examples

```
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)
```

twodlda *Two-Dimensional Linear Discriminant Analysis*

## Description

Two-Dimensional Linear Discriminant Analysis (2DLDA) for a 3-Tensor of matrix observations with class labels. Finds left and right projection matrices L and R that maximize class separability in the projected space. Each frontal slice is treated as a matrix observation. This is an iterative algorithm. For more details on 2DLDA, consult Ye et al. (2005).

## Usage

```
twodlda(tnsr, labels = NULL, r_ranks = NULL, c_ranks = NULL,
  max_iter = 25, tol = 1e-05)
```

## Arguments

| | |
|---|---|
| tnsr | 3-Tensor where the third mode indexes observations |
| labels | a vector of class labels (length must equal the third mode) |
| r_ranks | number of left projection vectors (columns of L) |
| c_ranks | number of right projection vectors (columns of R) |
| max_iter | maximum number of iterations if error stays above tol |
| tol | relative Frobenius norm error tolerance |

## Details

A progress bar is included to help monitor operations on large tensors. The input tensor must be 3-dimensional, where the third mode indexes the observations. Unlike other decompositions in this package, 2DLDA is a supervised method that requires class labels.

## Value

a list containing the following:

L the left projection matrix of size mode1 by r_ranks

R the right projection matrix of size mode2 by c_ranks

Z a list of projected feature matrices (one per observation), each of size r_ranks by c_ranks

conv whether or not resid < tol by the last iteration

est estimate of tnsr after projection and back-projection

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 labels must match the third mode of tnsr.

### References

J. Ye, R. Janardan, Q. Li, "Two-Dimensional Linear Discriminant Analysis". Advances in Neural Information Processing Systems 2005.

### See Also

mpca, tucker

### Examples

```
tnsr <- rand_tensor(c(5,4,10))
labels <- rep(c(1,2), each=5)
twodldaD <- twodlda(tnsr, labels=labels, r_ranks=2, c_ranks=2)
twodldaD$conv
twodldaD$norm_percent
```

---

## t_mult *Tensor Multiplication (T-MULT)*

---

### 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

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.

### 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 orthgonal 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 orthgonal 3-Tensor
- `V` the right orthgonal 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

```
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)

## 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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

## 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

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

## 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)
```

---

`vec-methods` *Tensor Vec*

---

### Description

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

### Usage

```
vec(tnsr)

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

### Arguments

| | |
|---|---|
| `tnsr` | the Tensor instance |

### Details

```
vec(tnsr)
```

### Value

vector with length `prod(x@modes)`

### References

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

### 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

```
## 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