Package ‘DelayedTensor’

July 9, 2025

Type Package

Title R package for sparse and out-of-core arithmetic and decomposition of Tensor

Version 1.14.0

Depends R (>= 4.1.0)

Imports methods, utils, S4Arrays, SparseArray, DelayedArray (>= 0.31.8), HDF5Array, BiocSingular, rTensor, DelayedRandomArray (>= 1.13.1), irlba, Matrix, einsum,

Suggests markdown, rmarkdown, BiocStyle, knitr, testthat, magrittr, dplyr, reticulate

Description DelayedTensor operates Tensor arithmetic directly on DelayedArray object. DelayedTensor provides some generic function related to Tensor arithmetic/decompotision and dispatches it on the DelayedArray class. DelayedTensor also suppors Tensor contraction by einsum function, which is inspired by numpy einsum.

License Artistic-2.0

biocViews Software, Infrastructure, DataRepresentation, DimensionReduction

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

BugReports https://github.com/rikenbit/DelayedTensor/issues

git_url https://git.bioconductor.org/packages/DelayedTensor

git_branch RELEASE_3_21

git_last_commit 85b592f

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-07-09

Author Koki Tsuyuzaki [aut, cre]

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

Contents

DelayedTensor-package — *R package for sparse and out-of-core arithmetic and decomposition of Tensor*

Description

DelayedTensor operates Tensor arithmetic directly on DelayedArray object. DelayedTensor provides some generic function related to Tensor arithmetic/decompotision and dispatches it on the DelayedArray class. DelayedTensor also suppors Tensor contraction by einsum function, which is inspired by numpy einsum.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Koki Tsuyuzaki [aut, cre]

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

See Also

Unfold operations `unfold`, `k_unfold`, `matvec`, `rs_unfold`, `cs_unfold`, `ttl` # Fold operations `fold`, `k_fold`, `unmatvec`, `rs_fold`, `cs_fold`, `ttm` # Vectorization `vec` # Norm operations `fnorm`, `innerProd` # Diagonal operations / Diagonal Tensor `diag`, `DelayedDiagonalArray` # Mode-wise operations `modeSum`, `modeMean` # Tensor product operations `hadamard`, `hadamard_list`, `kronecker`, `kronecker_list`, `khatri_rao`, `khatri_rao_list` # Utilities `list_rep`, `modebind_list`, `rbind_list`, `cbind_list` # Decomposition operations `hosvd`, `cp`, `tucker`, `mpca`, `pvd` # Einsum operation `einsum`

Examples

```
ls("package:DelayedTensor")
```

cbind_list — *Mode-binding against list*

Description

Returns the binded DelayedArray in column space.

Usage

```
cbind_list(L)
```

Arguments

L list of 2D DelayedArray

Details

This is a wrapper function to modebind_list, when the DelayedArrays are 2D.

Value

2D DelayedArray object

Note

The dimensions of column in each DelayedArray must match.

See Also

modebind_list

Examples

```
library("DelayedRandomArray")
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,3)),
    'darr2' = RandomUnifArray(c(2,3)))
cbind_list(dlizt)
```

---

cp-methods *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 *darr@num_modes vectors in the output, stored in darr@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(darr, num_components=NULL, max_iter=25, tol=1e-05)

## S4 method for signature 'DelayedArray'
cp(darr, num_components, max_iter, tol)
```

Arguments

darr	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

This function is an extension of the cp by DelayedArray.

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 darr after compression

fnorm_resid the Frobenius norm of the error fnorm(est-darr)

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

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(3,4,5))
cp(darr, num_components=2)
```

cs_fold-methods *Column Space Folding of 2D DelayedArray*

Description

The inverse operation to cs_unfold.

Usage

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

## S4 method for signature 'DelayedArray'
cs_fold(mat, m, modes)
```

Arguments

mat	DelayedArray object (only 2D)
m	the mode corresponding to cs_unfold
modes	the original modes of the DelayedArray

Details

This function is an extension of the cs_fold by DelayedArray.

This is a wrapper function to fold.

Value

DelayedArray (higher than 2D)

References

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

See Also

fold, cs_unfold

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT3 <- DelayedTensor::cs_unfold(darr, m=3)
identical(
    as.array(DelayedTensor::cs_fold(matT3, m=3, modes=c(2,3,4))),
    as.array(darr))
```

cs_unfold-methods — *Tensor Column Space Unfolding of DelayedArray*

Description

Please see `matvec` and `unfold`.

Usage

```
cs_unfold(darr, m)

## S4 method for signature 'DelayedArray'
cs_unfold(darr, m)
```

Arguments

`darr`	DelayedArray object
`m`	mode to be unfolded on

Details

This function is an extension of the `cs_unfold` by DelayedArray.

This is a wrapper function to `unfold`.

Value

DelayedArray (2D)

See Also

`unfold`, `cs_fold`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
DelayedTensor::cs_unfold(darr, m=3)
```

DelayedDiagonalArray *Diagonal DelayedArray*

Description

Constructor of the diagonal of a DelayedArray.

Usage

```
DelayedDiagonalArray(shape, value)
```

Arguments

shape	Shape of DelayedArray (mode of Tensor)
value	either a single value or a vector. This argument is optional. If nothing is specified, 1s are filled with each diagonal element.

Details

See also diag or diag.

Value

DelayedArray object

References

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

See Also

diag, diag

Examples

```
darr <- DelayedDiagonalArray(2:4, 5)
DelayedTensor::diag(darr)
```

`diag-methods` *DelayedArray Diagonals*

Description

Extract or replace the diagonal of a DelayedArray, or substitute the elements to the diagonal DelayedArray.

Usage

```
diag(darr)
diag(darr) <- value

## S4 method for signature 'DelayedArray'
diag(darr)
## S4 replacement method for signature 'DelayedArray'
diag(darr) <- value
```

Arguments

`darr`	DelayedArray object
`value`	either a single value or a vector of length equal to that of the current diagonal. Should be of a mode which can be coerced to that of `darr`.

Details

See also `DelayedDiagonalArray` or `diag`.

Value

1D DelayedArray (vector) with length `min(dim(darr))`

References

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

See Also

`DelayedDiagonalArray`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
DelayedTensor::diag(darr)
DelayedTensor::diag(darr)[1] <- 11111
DelayedTensor::diag(darr)[2] <- 22222
DelayedTensor::diag(darr)
```

einsum *Einstein Summation of DelayedArray*

Description

Einstein summation is a convenient and concise notation for operations on n-dimensional arrays.

NOTE: Sparse mode of einsum is not available for now.

Usage

```
einsum(subscripts, ...)
```

Arguments

`subscripts`	a string in Einstein notation where arrays are separated by ',' and the result is separated by '->'. For example `"ij,jk->ik"` corresponds to a standard matrix multiplication. Whitespace inside the `subscripts` is ignored. Unlike the equivalent functions in Python, `einsum` only supports the explicit mode. This means that the `subscripts` must contain '->'.
`...`	the DelayedArrays that are combined.

Details

This function is an extension of the `einsum` by DelayedArray.

Value

The `einsum` function returns an array with one dimension for each index in the result of the `subscripts`. For example `"ij,jk->ik"` produces a 2-dimensional array, `"abc,cd,de->abe"` produces a 3-dimensional array.

Examples

```
library("DelayedArray")
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(4,8))
darr2 <- RandomUnifArray(c(8,3))

# Matrix Multiply
darr1 %*% darr2
DelayedTensor::einsum("ij,jk -> ik", darr1, darr2)

# Diag
mat_sq <- RandomUnifArray(c(4,4))
DelayedTensor::diag(mat_sq)
einsum("ii->i", mat_sq)

# Trace
```

```
sum(DelayedTensor::diag(mat_sq))
einsum("ii->", mat_sq)

# Scalar product
darr3 <- RandomUnifArray(c(4,8))
darr3 * darr1
einsum("ij,ij->ij", darr3, darr1)

# Transpose
t(darr1)
einsum("ij->ji", darr1)

# Batched L2 norm
arr1 <- as.array(darr1)
arr3 <- as.array(darr3)
darr4 <- DelayedArray(array(c(arr1, arr3), dim = c(dim(arr1), 2)))

c(sum(darr1^2), sum(darr3^2))
einsum("ijb,ijb->b", darr4, darr4)
```

`fnorm-methods` *Tensor Frobenius Norm of DelayedArray*

Description

Returns the Frobenius norm of the Tensor instance.

Usage

```
fnorm(darr)

## S4 method for signature 'DelayedArray'
fnorm(darr)
```

Arguments

`darr`	DelayedArray object

Details

This function is an extension of the `fnorm` by DelayedArray.

Value

numeric Frobenius norm of `darr`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
fnorm(darr)
```

fold-methods *Tensor folding of 2D DelayedArray*

Description

General folding of a 2D DelayedArray into a higher-order DelayedArray(Tensor). This is designed to be the inverse function to unfold, 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)

## S4 method for signature 'DelayedArray'
fold(mat, row_idx, col_idx, modes)
```

Arguments

mat	DelayedArray object (only 2D)
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 DelayedArray

Details

This function is an extension of the fold by DelayedArray.

Value

DelayedArray object with modes given by modes

References

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

See Also

unfold, k_fold, unmatvec, rs_fold, cs_fold

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT3 <- DelayedTensor::unfold(darr, row_idx=2, col_idx=c(3,1))
identical(
    as.array(DelayedTensor::fold(matT3, row_idx=2,col_idx=c(3,1),
        modes=c(2,3,4))),
    as.array(darr))
```

---

getSparse — *Getter of the intermediate/output DelayedArray object in DelayedTensor*

Description

Whether the intermediate and output DelayedArray used in DelayedTensor is used as sparse tensor or not.

NOTE: Sparse mode is experimental! Whether it contributes to higher speed and lower memory is quite dependent on the sparsity of the DelayedArray, and the current implementation does not recognize the block size, which may cause Out-of-Memory errors.

Usage

```
getSparse()
```

Value

TRUE or FALSE (Default: FALSE)

Examples

```
getSparse()
```

---

getVerbose — *Getter function to control the verbose messages from DelayedTensor*

Description

Returns the verbose setting of DelayedTensor functions.

Usage

```
getVerbose()
```

Value

TRUE or FALSE (Default: FALSE)

Examples

```
getVerbose()
```

---

hadamard-methods *Hadamard Product of DelayedArray*

Description

Returns the Hadamard product of two Tensors. Commonly used for n-mode products and various Tensor decompositions.

Usage

```
hadamard(darr1, darr2)

## S4 method for signature 'DelayedArray,DelayedArray'
hadamard(darr1, darr2)
```

Arguments

darr1	first DelayedArray object
darr2	second DelayedArray object

Value

matrix that is the Hadamard product

Note

The modes/dimensions of each element of two Tensors must match.

See Also

khatri_rao, khatri_rao_list, kronecker, kronecker_list, hadamard_list

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,4))
darr2 <- RandomUnifArray(c(2,4))
hadamard(darr1, darr1)
```

hadamard_list — *Hadamard Product against list*

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 DelayedArray

Details

This function is an extension of the hadamard_list by DelayedArray.

Value

matrix that is the Hadamard product

Note

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

See Also

khatri_rao, khatri_rao_list, kronecker, kronecker_list, hadamard

Examples

```
library("DelayedRandomArray")
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,3,4)),
    'darr2' = RandomUnifArray(c(2,3,4)))
hadamard_list(dlizt)
```

hosvd-methods *(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(darr, ranks=NULL)

## S4 method for signature 'DelayedArray'
hosvd(darr, ranks)
```

Arguments

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

Details

This function is an extension of the `hosvd` by DelayedArray.

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 `darr` after compression

`fnorm_resid` the Frobenius norm of the error `fnorm(est-darr)` - if there was no truncation, then this is on the order of mach_eps * fnorm.

Note

The length of `ranks` must match `darr@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

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(3,4,5))
hosvd(darr, ranks=c(2,1,3))
```

`human_mid_brain` *Matrix object of human mid brain data*

Description

A matrix with 500 rows (genes) * 1977 columns (cells).

Usage

```
data(human_mid_brain)
```

Details

The data matrix is downloaded from GEO Series GSE76381 (https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE76381&
For the details, see inst/script/make-data.R.

References

Y-h. Taguchi and T. Turki (2019) Tensor Decomposition-Based Unsupervised Feature Extraction Applied to Single-Cell Gene Expression Analysis. *Frontiers in Genetics*, **10(864)**: 10:3389/fgene.2019.00864

See Also

`mouse_mid_brain`

Examples

```
data(human_mid_brain)
```

innerProd-methods *Tensors Inner Product of DelayedArray*

Description

Returns the inner product between two Tensors

Usage

```
innerProd(darr1, darr2)

## S4 method for signature 'DelayedArray,DelayedArray'
innerProd(darr1, darr2)
```

Arguments

darr1	first DelayedArray object
darr2	second DelayedArray object

Details

This function is an extension of the innerProd by DelayedArray.

Value

inner product between darr1 and darr2

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,3,4))
darr2 <- RandomUnifArray(c(2,3,4))
innerProd(darr1, darr2)
```

khatri_rao-methods *Khatri-Rao Product of DelayedArray*

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(darr1, darr2)

## S4 method for signature 'DelayedArray,DelayedArray'
khatri_rao(darr1, darr2)
```

Arguments

darr1	first DelayedArray object
darr2	second DelayedArray object

Details

This function is an extension of the khatri_rao by DelayedArray.

Value

matrix that is the Khatri-Rao product

Note

The number of columns must match in the two inputs.

See Also

hadamard, hadamard_list, kronecker, kronecker_list, khatri_rao_list

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,4))
darr2 <- RandomUnifArray(c(3,4))
khatri_rao(darr1, darr2)
```

---

khatri_rao_list *Khatri-Rao Product against list*

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 DelayedArray
reverse	whether or not to reverse the order

Details

This function is an extension of the khatri_rao_list by DelayedArray.

Value

matrix that is the Khatri-Rao product

Note

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

See Also

hadamard, hadamard_list, kronecker, kronecker_list, khatri_rao

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,3))
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,4)),
    'darr2' = RandomUnifArray(c(3,4)))
khatri_rao_list(dlizt)
```

---

kronecker-methods *Kronecker Product of DelayedArray*

Description

Returns the Kronecker product of two Tensors. Commonly used for n-mode products and various Tensor decompositions.

Usage

```
kronecker(darr1, darr2)

## S4 method for signature 'DelayedArray,DelayedArray'
kronecker(darr1, darr2)
```

Arguments

darr1	first DelayedArray object
darr2	second DelayedArray object

Value

matrix that is the Kronecker product

See Also

khatri_rao, khatri_rao_list, hadamard, hadamard_list, kronecker_list

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,3))
darr2 <- RandomUnifArray(c(4,5))
kronecker(darr1, darr2)
```

`kronecker_list` *Kronecker Product against list*

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 DelayedArray

Details

This function is an extension of the `kronecker_list` by DelayedArray.

Value

matrix that is the Kronecker product

See Also

`khatri_rao`, `khatri_rao_list`, `hadamard`, `hadamard_list`, `kronecker`

Examples

```
library("DelayedRandomArray")
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,3,4)),
    'darr2' = RandomUnifArray(c(2,3,4)))
kronecker_list(dlizt)
```

k_fold-methods *k-mode Folding of 2D DelayedArray*

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(darr, m), m, dim(darr))` will result in the original Tensor.

Usage

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

## S4 method for signature 'DelayedArray'
k_fold(mat, m, modes)
```

Arguments

`mat`	DelayedArray object (only 2D)
`m`	the index of the mode that is mapped onto the row indices
`modes`	the modes of the output DelayedArray

Details

This function is an extension of the `k_fold` by DelayedArray.

This is a wrapper function to `fold`.

Value

DelayedArray object with modes given by `modes`

References

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

See Also

`fold`, `k_unfold`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT2 <- k_unfold(darr, m=2)
identical(
    as.array(k_fold(matT2, m=2, modes=c(2,3,4))),
    as.array(darr))
```

k_unfold-methods *Tensor k-mode Unfolding of DelayedArray*

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(darr, m)

## S4 method for signature 'DelayedArray'
k_unfold(darr, m)
```

Arguments

darr	DelayedArray object
m	the index of the mode to unfold on

Details

This function is an extension of the `k_unfold` by DelayedArray.

This is a wrapper function to `unfold`.

See also `k_unfold(darr, m=NULL)`

Value

matrix with `dim(darr)[m]` rows and `prod(dim(darr)[-m])` columns

References

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

See Also

`unfold`, `k_fold`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
rs_unfold(darr, m=2)
```

list_rep *Replicate of arbitrary object*

Description

Returns the replicates of base obejct x.

Usage

```
list_rep(x, n=NULL)
```

Arguments

x	Any object
n	Number of replicate

Value

List

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
list_rep(darr, 3)
```

matvec-methods *Tensor Matvec Unfolding of DelayedArray*

Description

For 3-tensors only. Stacks the slices along the third mode.

Usage

```
matvec(darr)

## S4 method for signature 'DelayedArray'
matvec(darr)
```

Arguments

darr	DelayedArray object

Details

This function is an extension of the `matvec` by DelayedArray.

This is a wrapper function to `unfold`.

Value

matrix with `prod(dim(darr)[-m])` rows and `dim(darr)[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

`unfold`, `unmatvec`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matvec(darr)
```

`modebind_list` *Mode-binding against list*

Description

Returns the binded DelayedArray in mode-`m`.

Usage

```
modebind_list(L, m=NULL)
```

Arguments

`L`	list of DelayedArray
`m`	list of DelayedArray

Value

DelayedArray object

Note

The dimensions of mode `m` must match.

See Also

rbind_list, cbind_list

Examples

```
library("DelayedRandomArray")
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,3,4)),
    'darr2' = RandomUnifArray(c(2,3,4)))
modebind_list(dlizt, m=1)
modebind_list(dlizt, m=2)
modebind_list(dlizt, m=3)
```

---

modeMean-methods *Tensor Mean Across Single Mode of DelayedArray*

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(darr, m = NULL, drop = FALSE)

## S4 method for signature 'DelayedArray'
modeMean(darr, m, drop)
```

Arguments

darr	DelayedArray object
m	the index of the mode to average across
drop	whether or not mode m should be dropped

Details

This function is an extension of the modeMean by DelayedArray.

NOTE: Sparse mode of modeMean is not available for now.

`modeMean(darr, m=NULL, drop=FALSE)`

Value

K-1 or K Tensor, where K = `length(dim(darr))`

See Also

modeSum

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(1,2,3))
modeMean(darr, 1, drop=FALSE)
modeMean(darr, 1, drop=TRUE)
modeMean(darr, 2)
modeMean(darr, 3)
```

---

modeSum-methods *Tensor Sum Across Single Mode of DelayedArray*

Description

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

Usage

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

## S4 method for signature 'DelayedArray'
modeSum(darr, m, drop)
```

Arguments

`darr`	DelayedArray object
`m`	the index of the mode to sum across
`drop`	whether or not mode m should be dropped

Details

This function is an extension of the `modeSum` by DelayedArray.

NOTE: Sparse mode of modeSum is not available for now.

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

Value

K-1 or K tensor, where `K = length(dim(darr))`

See Also

`modeMean`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(1,2,3))
modeSum(darr, 1, drop=FALSE)
modeSum(darr, 1, drop=TRUE)
modeSum(darr, 2)
modeSum(darr, 3)
```

---

mouse_mid_brain   *Matrix object of mouse mid brain data*

Description

A matrix with 500 rows (genes) * 1907 columns (cells).

Usage

```
data(mouse_mid_brain)
```

Details

The data matrix is downloaded from GEO Series GSE76381 (https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE76381&
For the details, see inst/script/make-data.R.

References

Y-h. Taguchi and T. Turki (2019) Tensor Decomposition-Based Unsupervised Feature Extraction Applied to Single-Cell Gene Expression Analysis. *Frontiers in Genetics*, **10(864)**: 10:3389/fgene.2019.00864

See Also

mouse_mid_brain

Examples

```
data(mouse_mid_brain)
```

mpca-methods *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(darr, ranks=NULL, max_iter=25, tol=1e-05)

## S4 method for signature 'DelayedArray'
mpca(darr, ranks, max_iter, tol)
```

Arguments

`darr`	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

This function is an extension of the `mpca` by DelayedArray.

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 `darr` after compression

`norm_percent` the percent of Frobenius norm explained by the approximation

`fnorm_resid` the Frobenius norm of the error `fnorm(est-darr)`

`all_resids` vector containing the Frobenius norm of error for all the iterations

Note

The length of `ranks` must match `darr@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

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(3,4,5))
mpca(darr, ranks=c(1,2))
```

---

outerProd-methods *Tensors Outer Product of DelayedArray*

Description

Returns the outer product between two Tensors

Usage

```
outerProd(darr1, darr2)

## S4 method for signature 'DelayedArray,DelayedArray'
outerProd(darr1, darr2)
```

Arguments

`darr1`	first DelayedArray object
`darr2`	second DelayedArray object

Details

NOTE: Sparse mode of outerProd is not available for now.

Value

outer product between `darr1` and `darr2`

Examples

```
library("DelayedRandomArray")
darr1 <- RandomUnifArray(c(2,3))
darr2 <- RandomUnifArray(c(4,5))
outerProd(darr1, darr2)
```

`pvd-methods` *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(darr, uranks=NULL, wranks=NULL, a=NULL, b=NULL)

## S4 method for signature 'DelayedArray'
pvd(darr, uranks, wranks, a, b)
```

Arguments

`darr`	3D DelayedArray (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

This function is an extension of the `pvd` by DelayedArray.

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 `darr` after compression

`norm_percent` the percent of Frobenius norm explained by the approximation

`fnorm_resid` the Frobenius norm of the error `fnorm(est-darr)`

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

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(3,4,5))
pvd(darr, uranks=rep(2,5), wranks=rep(3,5), a=2, b=3)
```

`rbind_list` *Mode-binding against list*

Description

Returns the binded DelayedArray in row space.

Usage

```
rbind_list(L)
```

Arguments

`L`	list of 2D DelayedArray

Details

This is a wrapper function to `modebind_list`, when the DelayedArrays are 2D.

Value

2D DelayedArray object

Note

The dimensions of row in each DelayedArray must match.

See Also

`modebind_list`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
dlizt <- list(
    'darr1' = RandomUnifArray(c(2,3)),
    'darr2' = RandomUnifArray(c(2,3)))
rbind_list(dlizt)
```

---

rs_fold-methods *Row Space Folding of 2D DelayedArray*

Description

The inverse operation to rs_unfold.

Usage

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

## S4 method for signature 'DelayedArray'
rs_fold(mat, m, modes)
```

Arguments

mat	DelayedArray object (only 2D)
m	the mode corresponding to rs_unfold
modes	the original modes of the DelayedArray

Details

This function is an extension of the rs_fold by DelayedArray.

This is a wrapper function to fold.

Value

DelayedArray (higher than 2D)

References

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

See Also

fold, rs_unfold

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT2 <- rs_unfold(darr, m=2)
identical(
    as.array(rs_fold(matT2, m=2, modes=c(2,3,4))),
    as.array(darr))
```

---

rs_unfold-methods *Tensor Row Space Unfolding of DelayedArray*

Description

Please see k_unfold and unfold.

Usage

```
rs_unfold(darr, m)

## S4 method for signature 'DelayedArray'
rs_unfold(darr, m)
```

Arguments

darr	DelayedArray object
m	mode to be unfolded on

Details

This function is an extension of the rs_unfold by DelayedArray.

This is a wrapper function to unfold.

See also rs_unfold(darr, m=NULL)

Value

DelayedArray (2D)

See Also

unfold, rs_fold

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT2 <- rs_unfold(darr, m=2)
```

setSparse — *Setter to set the intermediate DelayedArray object in DelayedTensor*

Description

Set whether the intermediate and output DelayedArray used in DelayedTensor is used as sparse tensor or not.

NOTE: Sparse mode is experimental! Whether it contributes to higher speed and lower memory is quite dependent on the sparsity of the DelayedArray, and the current implementation does not recognize the block size, which may cause Out-of-Memory errors.

Usage

```
setSparse(as.sparse=FALSE)
```

Arguments

`as.sparse`	TRUE or FALSE (Default: FALSE)

Value

Nothing

Examples

```
setSparse(TRUE)
setSparse(FALSE)
```

setVerbose — *Setter to set the verbose mode of DelayedTensor*

Description

Set the verbose message to monitor the block-processing procedure.

Usage

```
setVerbose(as.verbose=FALSE)
```

Arguments

`as.verbose`	TRUE or FALSE (Default: FALSE)

Value

Nothing

Examples

```
setVerbose(TRUE)
setVerbose(FALSE)
```

`ttl` *DelayedArray 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(darr, list_mat, ms=NULL)
```

Arguments

`darr`	DelayedArray object with K modes
`list_mat`	a list of 2D DelayedArray objects
`ms`	a vector of modes to contract on (order should match the order of `list_mat`)

Details

This function is an extension of the `ttl` by DelayedArray.

This is a wrapper function to `unfold`.

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 `darr` with three matrices `mat1`, `mat2`, `mat3` on modes 1, 2, and 3. We could do `ttm(ttm(ttm(darr,mat1,1),mat2,2),3)`, or we could do `ttl(darr,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

DelayedArray object with K modes (Tensor)

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

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(3,4,5))
dlizt <- list(
    'darr1' = RandomUnifArray(c(10,3)),
    'darr2' = RandomUnifArray(c(10,4)))
ttl(darr, dlizt, ms=c(1,2))
```

---

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

Description

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

Usage

```
ttm(darr, mat, m = NULL)

## S4 method for signature 'DelayedArray,DelayedArray'
ttm(darr, mat, m)
```

Arguments

darr	DelayedArray object
mat	input 2D DelayedArray with same number columns as the mth mode of darr
m	the mode to contract on

Details

This function is an extension of the ttm by DelayedArray.

By definition, rs_unfold(ttm(darr, mat), m) = mat%*%rs_unfold(darr, m), so the number of columns in mat must match the mth mode of darr. For the math on the m-Mode Product, see Kolda and Bader (2009).

Value

a DelayedArray object with K modes

Note

The mth mode of darr 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

`rs_unfold`, `ttl`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
mat <- RandomUnifArray(c(10,4))
ttm(darr, mat, m=3)
```

`tucker-methods` *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` = `darr@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(darr, ranks=NULL, max_iter=25, tol=1e-05)

## S4 method for signature 'DelayedArray'
tucker(darr, ranks, max_iter, tol)
```

Arguments

`darr`	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

This function is an extension of the `tucker` by DelayedArray.

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 `darr` after compression

`norm_percent` the percent of Frobenius norm explained by the approximation

`fnorm_resid` the Frobenius norm of the error `fnorm(est-darr)`

`all_resids` vector containing the Frobenius norm of error for all the iterations

Note

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

References

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

See Also

hosvd, mpca

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
tucker(darr, ranks=c(1,2,3))
```

`unfold-methods` *Tensor Unfolding of 2D DelayedArray*

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 `dim(darr)`. 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(darr, row_idx, col_idx)

## S4 method for signature 'DelayedArray'
unfold(darr, row_idx, col_idx)
```

Arguments

darr	DelayedArray object
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

This function is an extension of the unfold by DelayedArray.

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

vec returns the vectorization of the tensor.

Value

2D DelayedArray 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, matvec, rs_unfold, cs_unfold

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
unfold(darr, row_idx=2, col_idx=c(3,1))
```

---

unmatvec-methods *Unmatvec Folding of 2D DelayedArray*

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)

## S4 method for signature 'DelayedArray'
unmatvec(mat, modes)
```

Arguments

`mat`	DelayedArray object (only 2D)
`modes`	the modes of the output DelayedArray

Details

This function is an extension of the `unmatvec` by DelayedArray.

This is a wrapper function to `fold`.

Value

DelayedArray object with modes given by `modes`

References

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

See Also

`fold`, `matvec`

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
matT1 <- matvec(darr)
identical(
    as.array(unmatvec(matT1, modes=c(2,3,4))),
    as.array(darr))
```

---

vec-methods *Tensor Vectorization of DelayedArray*

Description

Change the dimension of DelayedArray from multi-dimension (e.g. array) to single-dimension (e.g. vector).

Usage

```
vec(darr)

## S4 method for signature 'DelayedArray'
vec(darr)
```

Arguments

`darr`	DelayedArray object

Details

This function is an extension of the `vec` by DelayedArray.

Value

1D DelayedArray (vector) with length `prod(dim(darr))`

References

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

Examples

```
library("DelayedRandomArray")
darr <- RandomUnifArray(c(2,3,4))
vec(darr)
```

Index