Package 'tensorA'

December 13, 2023

Version 0.36.2.1

Date 2020-11-13

Title Advanced Tensor Arithmetic with Named Indices

Author K. Gerald van den Boogaart <boogaart@uni-greifswald.de>

Maintainer K. Gerald van den Boogaart <boogaart@math.tu-freiberg.de>

Depends R (>= 2.2.0), stats

Description Provides convenience functions for advanced
linear algebra with tensors and computation with data sets of
tensors on a higher level abstraction. It includes Einstein and
Riemann summing conventions, dragging, co- and contravariate
indices, parallel computations on sequences of tensors.

License GPL (>= 2)

URL http://www.stat.boogaart.de/tensorA/

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-12-13 17:28:05 UTC

R topics documented:

tensorA-package 2
add.tensor 5
as.tensor 6
bind.tensor 7
chol.tensor 8
delta.tensor 10
diag.tensor 11
diagmul.tensor 12
drag.tensor 13
einstein.tensor 15
ftable.tensor 16
inv.tensor 17
is.tensor 18

level.tensor . . . 19
margin.tensor . . . 20
mark.tensor . . . 21
mean.tensor . . . 22
mul.tensor . . . 24
names.tensor . . . 25
norm.tensor . . . 26
one.tensor . . . 28
pos.tensor . . . 29
power.tensor . . . 30
reorder.tensor . . . 31
reptensor . . . 32
riemann.tensor . . . 33
sequencing . . . 35
slice.tensor . . . 36
solve.tensor . . . 37
svd.tensor . . . 39
to.matrix.tensor . . . 40
to.tensor . . . 41
toPos.tensor . . . 43
trace.tensor . . . 44
tripledelta.tensor . . . 45
undrop.tensor . . . 46
untensor . . . 46

Index **48**

`tensorA-package` *The tensorA package for tensor arithmetic*

Description

tensorA stands for "tensor arithmetic". A tensor is a mathematical generalization of vector and matrix with many applications in physics, geometry and in the statistics of vectors valued data. However the package is also useful in any case, where computations on sequences of matrices, vectors or even tensors is involved.

Details

Package:	tensorA
Type:	Package
Version:	0.1
Date:	2006-06-08
License:	GPL Version 2 or newer

The tensorA package is made to allow programming for tensors in R on the same level of abstraction as we know from matrices. It provides many of the mathematical operations common in tensor arithmetics including the whole tensor calculus of covariate and contravariate indices, naming of indices, sequence of indices, decompositions of tensors, Einstein and Riemann summing conventions and vectorized computations on datasets of tensors just like the well vectorization of numbers in R. It provides tools to write tensor formulae very close to there paper form and to handle tensors of arbitrary level with simple programs.
The whole documentation of the package is best read in pdf or dvi format since it contains complicated mathematical formulae with multi-indices.

Simply speaking a tensor (see `to.tensor`) is just a multidimensional array `A[,,]`. The number of indices (i.e. `length(dim(A))` is called the level of the tensor (see `level.tensor`). A tensor is mathematically it is denoted by a core symbol (e.g. A) with multiple indices:e.g.

$$A_{ijk}$$

The indices i, j, k can be seen as names for the dimensions and as integer numbers giving the respective index into the array. However the tensor is an algebraical object with many algebraical operations defined on it, which are also of relevancy for programming, e.g. in the parallel treatment of multiple linear equation systems.

To understand the package we need to understand tensors including their mathematical origin, the corresponding calculus, notation and basic operations.
One mathematical interpretation of a tensor, which is most relevant for physics, that of a multilinear form of $level(A)$ vectors, i.e. a function of $level(A)$ many vectors to the real or complex numbers, which is linear with respect to each of its arguments. E.g. the two vectors "plane face direction" and "force direction" are mapped to the actual force by the stress tensor.
Row vectors are a special case of that and likewise column vectors as linear forms for row vectors. Matrices are bilinear forms of a row vector and a column vector. Thus Vectors and Matrices are examples of tensors of level 1 and 2.

Another interpretation of a tensor is the that of a linear mapping, quite like a matrix, but from a tensor space (e.g. the space of matrices or vectors seen as tensor) to another tensor space (e.g. again a space of matrices). An example for that is the Hook elasticity tensor mapping the strain tensor (i.e. a matrix describing the local deformation) to the stress tensor (i.e. a matrix describing the local forces). The Hook tensor is a tensor of level 4. Statistically relevant tensors of level 4 are e.g. covariances of matrices mapping two linear forms (i.e. 2 level 2 tensors) on observed matrices to there covariance. The mapping is performed with the tensor product, which is not unlike a matrix product, however more general. Let denote A a matrix and v a vector, we would write $r = Ab$ for the matrix product and `r <- A%*%b` in R, which is defined as:

$$r_i = \sum_{j=1}^{j_{\max}} A_{ij} b_j$$

We know that we have to use the \(j\)-dimension in the summing, since the matrix multiplication rule says "row times column". Since a tensor can have more than two indices there is no row or column specified and we need to specify the inner product differently. To do this in the Einstein-Notation writing the tensor always with indices $r_i = A_{ij} b_j$ and according to the Einstein summing rule the entries of \(r_i\) are given by an implicit sum over all indices which show up twice in this

notation:

$$r_i = \sum_{j=1}^{j_{\max}} A_{ij} b_j$$

This notation allows for a multitude of other products: $A_{ij}b_i = t(A)b$, $A_{ij}b_k = outer(A,b)$, $A_{ii}b_j = trace(A)b$ with equal simplicity and without any additional functions. More complicated products involving more than tensors of level two can not even be formulated in pure matrix algebra without re-dimensioning of arrays e.g. $b_i b_j b_k$, $A_{ijk}b_j$. The Einstein summing rule is implemented in `einstein.tensor` and supported by the index sequencing functions `$.tensor` and `|.tensor`. A general multiplication allowing to identify and sum over any two indices is implemented in `trace.tensor`, when the indices are in the same tensor and in `mul.tensor`, when the indices to sum over are in different tensors.
Tensors with the same level and dimensions (identified by name and dimension) can also be added like matrices according to the rule that the values with the same combination of index values are added (see `add.tensor`). The implementation takes care of the sequence of the indices and rearranges them accordingly to match dimensions with the same name. E.g. the tensor addition

$$E_i jk = A_{ijk} + B_{kji}$$

has the effect, which is expressed by the same formula read in entries, which is also true for the more surprising

$$E_i jk = A_{ij} + B_{kj}$$

Like a matrix a tensor can also be seen as a mapping from one tensor space to another:

$$A_{i_1 \ldots i_d j_1 \ldots j_e} x_{j_1 \ldots j_e} = b_{i_1 \ldots i_d}$$

In this reading all the standard matrix computations and decompositions get a tensorial interpretation and generalization. The package provides some of these (see `svd.tensor`).
Another interpretation of tensors is as a sequence of tensors of lower level. E.g. a data matrix is seen as a sequence of vectors in multivariate dataset. The tensorA library provides means to do computation on these in parallel on these sequences of tensors like we can do parallel computation on sequences of numbers. This is typically done by the `by=` argument present in most functions and giving the index enumerating the elements of the sequence.
E.g. If we have sequence V_{ijd} of variance matrices V_{ij} of some sequence v_{id} of vectors and we would like to transform the vectors with some Matrix $M_{i'i}$ we would get the sequence of transformed variances by $V_{ijd}M_{i'i}M_{j'j}$. However if the M_{ki} are different for each of the elements in sequence we would have stored them in a tensor M_{kid} and would have to replace M_{kid} with $M_{kidd'} = M_{kid}$ if $d = d'$ and zero otherwise. We can than get our result by

$$V_{ijd}M_{i'id'd}M_{j'jd'd''}$$

and we would have a by dimension of `by="d"`. These operations are not strictly mathematical tensor operation, but generalizations of the vectorization approach of R. This is also closely related to `diagmul.tensor` or `diag.tensor`.
To complicate things the Einstein rule is only valid in case of tensors represented with respect to a orthogonal basis. Otherwise tensors get lower and upper indices like

$$A_{i\cdot k}^{\cdot j \cdot}$$

for representation in the covariate and contravariate form of the basis. In this case the Riemann summing rule applies which only sums over pairs of the same index, where one is in the lower and one is in the upper position. The contravariate form is represented with indices prefixed by ^.
The state of being covariate or contravariate can be changed by the dragging rule, which allows to switch between both state through the multiplication with the geometry tensors g_i^j. This can be done through drag.tensor.

Author(s)

K.Gerald van den Boogaart <boogaart@uni-greifswald.de

See Also

to.tensor, mul.tensor , einstein.tensor, add.tensor, [[.tensor, |.tensor

Examples

```
A <- to.tensor( 1:20, c(a=2,b=2,c=5) )
A
ftable(A)
B <- to.tensor( c(0,1,1,0) , c(a=2,"a'"=2))
A %e% B
drag.tensor( A , B, c("a","b"))
A %e% one.tensor(c(c=5))/5     # a mean of matrices
reorder.tensor(A,c("c","b","a"))
A -  reorder.tensor(A,c("c","b","a"))  # =0 since sequence is irrelevant
inv.tensor(A,"a",by="c")
```

---

add.tensor *Element-wise arithmetic operations +,-,*,/ with tensors*

Description

Adds/subs/multiplies/devides tensors element by element . The luxury difference to a simple + is that we do not need to consider the correct permutation of indices or rules on implicit replication, since all of this is handled automatically.

Usage

```
add.tensor(X,Y,op="+",only=NULL)
## Methods for class tensor
# x + y
# x - y
# x * y
# x - y
```

Arguments

X	a tensor
Y	a tensor
op	a binary function used to perform the "addition"
only	a list of dimnames that may be considered as equal. This parameter is here to allow parallelization of tensors with only partially known structure.

Details

The tensors are properly reordered such that dimensions of the same name are identified. If dimensions are missing in one of the tensors it is correspondingly repeated.

Value

A tensor giving the element-wise operation X,Y. If some of the indices are missing in one of the tensors they are added by repetition.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor

Examples

```
A <- to.tensor(1:20,c(U=2,V=2,W=5))
add.tensor(A,A)/2 -A
(A+A)/2
A/A
A * 1/A
norm.tensor(reorder.tensor(A,c(2,3,1)) - A)
```

---

as.tensor *Coercion to a tensor*

Description

Coerces a array to a tensor keeping dimension and names.

Usage

```
as.tensor(X,...)
## Default S3 method:
as.tensor(X,...,dims=NULL)
## S3 method for class 'tensor'
as.tensor(X,...)
```

Arguments

X	a multidimensional array
...	further generic arguments
dims	the new dim attribute to be used

Details

The main idea is that a multiway array like a vector or a matrix is nothing else than a tensor for R, but it still needs the tensor class be used with the tensorA library. However this is more a convenience function for migraters than a proper way construct a tensor, which is done by `to.tensor`.

Value

a tensor containing the same data as X

Note

You should typically use the `to.tensor` to generate a tensor, when you want to write vectorizable functions for tensors.

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`

Examples

```
A <- diag(5)
as.tensor(A)
```

---

bind.tensor *A cbind/rbind for tensors*

Description

Tensors can be put side by side in one dimension if they are of equal size in all other dimensions.

Usage

```
bind.tensor(A,dA=NULL,B,dB=dA)
```

Arguments

A	the first tensor
dA	the dimension of A to be used for binding the tensors
B	the second tensor
dB	the dimension of B to be used for binding the tensors

Details

This function works like a cbind or rbind function for tensors.

Value

a tensor with the tensors combined to one

Note

binding does not preserve the sequence of the dimensions.

Author(s)

K.Gerald van den Boogaart

See Also

base{cbind}

Examples

```
  A <- to.tensor(1:6,c(a=2,b=3))
bind.tensor(A,"a",A)
bind.tensor(A,"b",A)
```

---

chol.tensor *Cholesky decomposition of a tensor*

Description

A tensor can be seen as a linear mapping of a tensor to a tensor. This function computes its Cholesky decomposition.

Usage

```
chol.tensor(X,i,j,...,name="lambda")
```

Arguments

X	The tensor to be decomposed
i	The image dimensions of the linear mapping
j	The coimage dimensions of the linear mapping
name	The name of the eigenspace dimension. This is the dimension created by the decompositions, in which the eigenvectors are e_i
...	for generic use only

Details

A tensor can be seen as a linear mapping of a tensor to a tensor. Let denote R_i the space of real tensors with dimensions $i_1 ... i_d$.

chol.tensor Computes for a tensor $a_{i_1 ... i_d j_1 ... j_d}$ representing a positive definit mapping form R_j to R_i with equal dimension structure in i and j its "Cholesky" decomposition $L_{i_1 ... i_d \lambda}$ such that

$$a_{i_1 ... i_d j_1 ... j_d} = \sum_{\lambda} L_{i_1 ... i_d \lambda} L_{j_1 ... j_d \lambda}$$

Value

a tensor

Note

A by argument is not necessary, since both processing dimensions have to be given.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor, svd.tensor

Examples

```
A <- to.tensor(rnorm(15),c(a=3,b=5))
AAt <- einstein.tensor(A,mark(A,i="a"))
ch <- chol.tensor(AAt,"a","a'",name="lambda")
#names(ch)[1]<-"lambda"
einstein.tensor(ch,mark(ch,i="a")) # AAt

A <- to.tensor(rnorm(30),c(a=3,b=5,c=2))
AAt <- einstein.tensor(A,mark(A,i="a"),by="c")
ch <- chol.tensor(AAt,"a","a'",name="lambda")
einstein.tensor(ch,mark(ch,i="a"),by="c") #AAt
```

`delta.tensor` *Creates a Kronecker delta tensor*

Description

The delta tensor is the tensor equivalent of the identity.

Usage

```
delta.tensor(d,mark="'",dn=NULL,by=NULL)
```

Arguments

`d`	the row dimensions
`mark`	a character to be concatenated to the names of the row dimensions to get the column dimension names
`dn`	dimnames for the result
`by`	the dimensions which should not be duplicated

Details

$$E_{i_1 \ldots i_n j_1 \ldots j_n} = \delta_{i_1 j_1} \ldots \delta_{i_n j_n}$$

Value

a tensor with dimension `c(d,mark(d,mark))`

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`

Examples

```
delta.tensor(c(a=2,b=3))
```

`diag.tensor` *Creates a "diagonal" tensor*

Description

The diagonal tensor is the tensor equivalent of the diagonal matrix.

Usage

```
diag.tensor(X,mark="'",dn=NULL,by=NULL)
```

Arguments

`X`	a tensor containing the diagonal entries.
`mark`	a character to be concatenated to the names of the row dimensions to get the column dimension names
`dn`	dimnames which are used twice
`by`	The diagonal tensor is created for each level of the indices in `by`.

Details

$$E_{i_1 \ldots i_n j_1 \ldots j_n} = \delta_{i_1 j_1} \ldots \delta_{i_n j_n}$$

Value

a tensor with dimension `c(dim(X),mark(dim(X),mark))`

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`

Examples

```
A <- to.tensor(1:4,c(a=2,b=2))
diag.tensor(A)
diag.tensor(A,by="b")
```

`diagmul.tensor` *Multiplication of a tensor with a tensor given by its diagonal*

Description

This is a convenience function for scaling elements of a tensor with different numbers based on their position in the tensor.

Usage

```
diagmul.tensor(X,i=names(D),D,j=i,by=NULL)
```

Arguments

`X`	The tensor to be scaled
`D`	A tensor containing scaling constants
`i`	numeric of character vector giving the dimensions of X to be used for the product.
`j`	numeric of character vector giving the dimensions of D to be used for the product.
`by`	Every operation is parallel for all levels of by in X and/or D.

Details

Let

$$X_{i_1 \ldots i_d k_1 \ldots k_d}$$

and

$$D_{j_1 \ldots j_d}$$

than the result is:

$$E_{i_1 \ldots i_d k_1 \ldots k_d} = X_{i_1 \ldots i_d k_1 \ldots k_d} D_{j_1 \ldots j_d}$$

Value

A tensor with the shape and dimensions as X with entries X_{ik} scaled by $D_i m$, where i and k can represent multi-indices.

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`

Examples

```
(A <- matrix(rep(1:3,each=3),nrow=3))
(b <- to.tensor(c(1,1/2,1/3)))
diagmul.tensor(as.tensor(A),2,as.tensor(c(1,1/2,1/3)),1)
diagmul.tensor(as.tensor(A),1,as.tensor(c(1,1/2,1/3)),1)
A %*% diag(b)
diag(b) %*% A
```

`drag.tensor` *Managing covariate and contravariate indices*

Description

Each index of a tensor can be covariate or contravariate. The `is.*` routines check the state of the individual indices based on the tensor, its dimension or its index names. `drag.tensor` can change the state for the tensor and contraname for the names of the tensor.

Usage

```
drag.tensor(x,g,d)
contraname(x)
is.covariate(x,...)
## S3 method for class 'tensor'
is.covariate(x,...)
## S3 method for class 'numeric'
is.covariate(x,...)
## S3 method for class 'character'
is.covariate(x,...)
as.covariate(x,...)
## S3 method for class 'character'
as.covariate(x,...)
is.contravariate(x,...)
## S3 method for class 'numeric'
is.contravariate(x,...)
## S3 method for class 'character'
is.contravariate(x,...)
as.contravariate(x,...)
## S3 method for class 'character'
as.contravariate(x,...)
```

Arguments

x	the tensor, its dimension (for `*.numeric`) or its index-names (for `*.character` and `contraname`)
g	The geometry tensor g_{ij} giving the transformation between covariate and contravariate. It needs to have either covariate and or contravariate indices.

d	a vector (or list) of indices that should be dragged, i.e. multiplied with $g_i^{\ j}$ in the right way such that it changes from covariate to contravariate or vice versa. The name of the index is kept, only its state changes. The index is thus dragged from one state to the other. Indices can given in covariate or contravariate form.
...	only for generic use

Details

The covariate and contravariate state of a dimension corresponds to column and row vectors. The transformation between these type is done by a linear mapping give by the geometry tensor g, which is the identity matrix if the enclosing the geometry is represented by the orthonormal basis and ordinary scalar product.

Value

drag.tensor	returns a tensor like x but with the dimension
is.covariate	returns a boolean vector giving true for every covariate index
is.contravariate	returns a boolean vector giving true for every contravariate index
as.*	changes the state of the indices
contraname	returns the names with opposite the opposite covariate and contravariate state

Author(s)

K. Gerald van den Boogaart

See Also

riemann.tensor, to.tensor, Tensor

Examples

```
g <- to.tensor(c(1,2,0,1),c(i=2,j=2))
A <- to.tensor(rnorm(8),c(a=2,b=2,c=2))
A2 <- drag.tensor(A,g,c("b","c"))
A2
names(A2)
as.covariate(names(A2))
as.contravariate(names(A2))
is.covariate(A2)
is.contravariate(A2)
riemann.tensor(A2,g)
```

`einstein.tensor` *Tensor multiplication with Einstein's convention, by summing over all equally named indices.*

Description

Multiplies tensors by multiplying over all duplicate names according to Einsteins summing convention by doing an implicit inner product over all dimensions with the same name.

Usage

```
einstein.tensor(...,only=NULL,by=NULL)
## Methods for class tensor
# x %e% y
## Default method
# x %e% y
```

Arguments

`...`	some tensors, or a renaming code
`only`	optional list, if given only names in this list are automatically processed
`x`	a tensor
`y`	a tensor
`by`	the parallel dimensions

Details

see `mul.tensor` on details on tensor multiplication. In `einstein.tensor` complex operations can be performed by command and renaming code: The arguments are processed from left to right and multiplied. Unnamed attributes are regarded as tensors or scalars and multiplied with the current result by the Einstein summing convention, which means an inner product over all dimensions with the same name. Named attributes can either have the name diag, which performs a diagmul according to the same-name convention or be of the form `A="B"` or `"A"="B"`, for which we have two cases. If both names are present in the current result, an inner multiplication (trace) of on these two dimensions is performed. If only the first is a name up to this point, the specific dimension is renamed to the second name. This renaming might be visible in the result or inducing a multiplication according to the Einstein convention later.

Value

the tensor product of all the tensors along all duplicate dimensions.

Author(s)

K. Gerald van den Boogaart

See Also

mul.tensor, to.tensor, riemann.tensor

Examples

```
A <- to.tensor(1:20,c(U=2,V=2,W=5))
B <- to.tensor(1:30,list(U=c("a","b","c"),V=c("B1","B2"),W=1:5))
einstein.tensor(A,U="U'",B)
einstein.tensor(A,U="U'",mark(B,"k"))
einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk")
einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk",1/10)
einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk",diag=to.tensor(c(1,1/10,1/100),c(Uk=3)))

ftable(einstein.tensor(A,U="U'",B))
ftable(einstein.tensor(A,U="U'",mark(B,"k")))
ftable(einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk"))
ftable(einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk",1/10))
ftable(einstein.tensor(A,U="U'",mark(B,"k"),V="Vk",W="Wk",diag=to.tensor(c(1,1/10,1/100),c(Uk=3))))

dim(A[[U=~M]])
A[[U=~M]]
A[[U=~M,V=~"L"]]
```

---

ftable.tensor *Pretty printing of tensors*

Description

Returns the tensor as (flat) ftable, providing a pretty output.

Usage

```
## S3 method for class 'tensor'
ftable(x,...)
```

Arguments

x	the tensor
...	additional arguments to ftable

Details

This function is called for a pretty output of a tensor, just try it.

Value

an ftable containing the same data as the tensor

Author(s)

K. Gerald van den Boogaart

See Also

ftable

Examples

```
A <- to.tensor(1:20,c(U=2,V=2,W=5))
A
dim(A)
names(A)
dimnames(A)

ftable(to.tensor(A))
ftable(to.tensor(c(A),dim(A)))
```

---

inv.tensor *Inversion of a tensor as linear mapping from tensors to tensors*

Description

A tensor can be seen as a linear mapping of a tensor to a tensor. This function computes its (generalized-Moore-Penrose) inverse.

Usage

```
inv.tensor(X,i,...,allowSingular=FALSE,eps=1E-10,by=NULL)
```

Arguments

X	The tensor to be decomposed
i	The image dimensions of the linear mapping
allowSingular	A boolean, indicating that a Moore-Penrose-Inverse should be computed rather than an error generated in case of a numerically singular mapping.
...	further arguments for generic use
eps	The limit for condition-number, to select an generalized inverse.
by	the operation is done in parallel for these dimensions

Details

A tensor can be seen as a linear mapping of a tensor to a tensor.

inv.tensor Computes the inverse of the mapping

Value

a tensor containing the inverse mapping. If allowSingular is given and the condition number of the matrix is bellow eps a generalized inverse is returned.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor, solve.tensor, svd.tensor

Examples

```
# SVD
# inv.tensor
R1  <- matrix(rnorm(9),nrow=3)
R1i <- solve(R1)
R2 <- to.tensor(R1,c(a=3,b=3),what=1:2)
R2i <- to.tensor(R1i,c(b=3,a=3),what=1:2)

inv.tensor(R2,"a","b") - R2i
inv.tensor(R2,"a","b",allowSingular=TRUE) - R2i

inv.tensor(rep(R2,4,1,"K"),"a","b",by="K") - rep(R2i,4,1,"K")
inv.tensor(rep(R2,4,1,"K"),"a","b",by="K",allowSingular=TRUE) - rep(R2i,4,3,"K")
```

---

is.tensor *Checking for being a tensor*

Description

Checks whether the object has a tensor attribute.

Usage

```
is.tensor(X)
```

Arguments

X	the objected to be checked

Details

This is a simple convenience function to check for the property of being a tensor.

Value

boolean

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor

Examples

```
A <- matrix(1:9,nrow=3)
is.tensor(A) # no
A <- to.tensor(A)
is.tensor(A) # yes
```

---

level.tensor *The level (number of indices) of a tensor*

Description

The level of a tensor is the number of dimensions or subscripts used.

Usage

```
level.tensor(X,...)
```

Arguments

X	the tensor to be used
...	not used

Details

The level of the tensor is the length of its dim attribute. Objects without a dim attribute get level 1 if they are of length > 1 and are marked as scalars by 0 level otherwise.

Value

the number of levels

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor

Examples

```
A <- to.tensor(1:24,c(a=1,b=2,c=3,d=4))
level.tensor(A)
level.tensor(matrix(1))
level.tensor(1:10)
level.tensor(1)
```

`margin.tensor` *Marginalization of tensors*

Description

The function removes dimensions from a tensor by summing all entries which only differ in these dimensions.

Usage

```
margin.tensor(X,i=NULL,by=NULL)
```

Arguments

X	the tensor
i	the dimensions to be removed
by	instead of i the dimensions to be kept

Details

This is a tensor multiplication with the 1_i tensor.

Value

The tensor with all elements only differing only in the dimensions specified added up and only the other dimensions left over.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor

Examples

```
A <- diag(1:5)
A
margin.tensor(A,1)

A <- to.tensor(1:30,dim=c(i=3,j=5,k=2))
ftable(A)
margin.tensor(A,"j")
```

`mark.tensor` *Marks the names of a tensor with a mark*

Description

This modifies the names of the dimensions in a simple and reversible way by adding a mark.

Usage

```
mark(X,mark,...)
## S3 method for class 'tensor'
mark(X,mark="'",i=1:level.tensor(X),...,by=NULL)
## S3 method for class 'numeric'
mark(X,mark="'",i=1:length(X),...,by=NULL)
## S3 method for class 'character'
mark(X,mark="'",i=1:length(X),...,by=NULL)
```

Arguments

`X`	A tensor or dimension to be marked
`mark`	a character giving the mark
`i`	the dimensions to be marked
`...`	generic arguments
`by`	Dimensions not to be marked. Wins in case of conflicts.

Details

The concept is very important in tensor algebra since it allows to keep dimensions connected without but still distinguishable. Eventually later a function for the Riemann summing rule will make use of marks to distinguish covariate and contravariate dimensions.

Value

A object similar to X but with marked dimensions.

Author(s)

K. Gerald van den Boogaart

See Also

delta.tensor, diag.tensor

Examples

```
# The outer product
A <- to.tensor(1:4,c(a=2,b=2))
A
```

---

mean.tensor *Mean and variance of tensors*

Description

Mean and variance of tensors again tensors.

Usage

```
 ## S3 method for class 'tensor'
mean(x,along,...,na.rm=FALSE)
 ## S3 method for class 'tensor'
var(x,y=NULL,...,along,by=NULL,na.rm=FALSE,mark="'")
```

Arguments

x	(set of) dataset(s) of tensors represented by a tensor
y	a second dataset of connected tensors represented by a tensor
along	the indices indexing the datasets
...	here for generic compatibility with the compositions package
by	the indices indexing the set of datasets
na.rm	a boolean, if FALSE and missings are in the dataset a error is given. If TRUE pairwise exclusion is used.
mark	the to mark the second instance of indices in var(x,...)

Details

Let denote a the along dimension, $i_1, \ldots, i_k$ and $j_1, \ldots, j_l$ the data dimension, and b the by dimension, then the mean is given by:

$$M^x_{bi_1,\ldots,i_k} = \frac{1}{n}\sum_a x_{abi_1,\ldots,i_k}$$

the covariance by

$$C_{abi_1,\ldots,i_k j_1,\ldots,j_l} = \frac{1}{n-1}\sum_a (x_{abi_1,\ldots,i_k} - M^x_{bi_1,\ldots,i_k})(y_{abj_1,\ldots,j_l} - M^y_{bj_1,\ldots,j_l})$$

and the variance by

$$V_{abi_1,\ldots,i_k i'_1,\ldots,i'_l} = \frac{1}{n-1}\sum_a (x_{abi_1,\ldots,i_k} - M^x_{bi_1,\ldots,i_k})(x_{abi'_1,\ldots,i'_k} - M^x_{bi'_1,\ldots,i'_l})$$

Value

`mean`	gives a tensor like x without the along dimensions representing the a mean over all tensors in the dataset. It is not necessary to have a by dimension since everything not in along is automatically treated parallel
`var(x,...)`	Gives the covariate tensor representing the covariance of x and y. The data tensor indices of x any y should be different, since otherwise duplicated names exist in the result.
`var(x,...)`	Gives the covariate representation of the variance of x. All data indices (i.e. all indices neither in by nor in along are duplicated. One with and one without the given mark.

Author(s)

K.Gerald van den Boogaart

See Also

tensorA

Examples

```
d1 <- c(a=2,b=2)
d2 <- c("a'"=2,"b'"=2)
# a mean tensor:
m <- to.tensor(1:4,d1)
# a positive definite variance tensor:
V <- delta.tensor(d1)+one.tensor(c(d1,d2))
V
# Simulate Normally distributed tensors with these moments:
X <- (power.tensor(V,c("a","b"),c("a'","b'"),p=1/2)  %e%
     to.tensor(rnorm(1000*2*2),c(i=1000,d2))) + m
# The mean
```

```
mean.tensor(X,along="i")
# Full tensorial covariance:
var.tensor(X,along="i")
# Variance of the slices  X[[b=1]] and X[[b=2]] :
var.tensor(X,along="i",by="b")
# Covariance of the slices X[[b=1]] and X[[b=2]] :
var.tensor(X[[b=1]],X[[a=~"a'",b=2]],along="i")
```

`mul.tensor` *Tensor multiplication for the tensor class*

Description

Performs a tensor multiplication like tensor(), but with named indices, keeping dimnames, and vectorized.

Usage

```
mul.tensor(X,i=c(),Y,j=i,by=NULL)
```

Arguments

`X`	a tensor to be multiplied
`i`	numeric or character vector specifying the dimension to be used in the multiplication for X
`Y`	a tensor to be multiplied
`j`	numeric or character vector specifying the dimension to be used in the multiplication for Y
`by`	the by dimensions if present and not mentioned in i or j are used as sequence dimensions. tensors in these dimensions are processed in parallel. So in this dimension the product is neither inner nor outer but parallel like `a*b`, rather than `a%*%b` or `a%o%b`. Unmentioned dimensions get an outer product. Mentioned dimensions an inner.

Details

Say

$$X_{i_1 \ldots i_n h_1 \ldots h_l}$$

and

$$Y_{j_1 \ldots j_n k_1 \ldots k_m}$$

the the result is:

$$E_{h_1 \ldots h_l k_1 \ldots k_m} = \sum_{i_1,\ldots,i_n} X_{i_1 \ldots i_n h_1 \ldots h_l} Y_{j_1 \ldots j_n k_1 \ldots k_m}$$

This is an full outer product with i,j not given and a full inner product product of i=dim(X)

Value

The tensor product of X and Y with respect to the regarding dimensions.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor, %e%, %r%, diagmul.tensor, einstein.tensor, riemann.tensor, solve.tensor

Examples

```
A <- to.tensor(1:20,c(A=2,B=2,C=5))
B <- to.tensor(1:20,c(D=2,B=2,E=5))
mul.tensor(A,"A",A,"B")
```

---

names.tensor *Getting and setting index and dimensionnames of a tensor*

Description

The names of a tensor are the names of its dimension

Usage

```
## S3 method for class 'tensor'
names(x)
## S3 replacement method for class 'tensor'
names(x) <- value
## S3 method for class 'tensor'
dimnames(x)
## S3 replacement method for class 'tensor'
dimnames(x) <- value
## S3 replacement method for class 'tensor'
dim(x) <- value
```

Arguments

x	a tensor object
value	The new value. If this is a named list it replaces the names of the dimensions. If its an unnamed list it gets the names of the dimensions.

Details

The names of the dimensions of the tensor are very relevant in any tensor arithmetic since they are the principle way to specify the dimensions to be involved in an operation. The dimnames function is here only for convenice to guarantee that the names of the dimnames are always the same as the names of the dimensions and to ensure that always at least a list with the right length and names.

Value

the names of the dimensions the tensor

Author(s)

K. Gerald van den Boogaart

See Also

mul.tensor

Examples

```
A <- to.tensor(1:20,c(U=2,V=2,W=5))
A
dim(A)
names(A)
names(A) <- c("A","B","C")
A
dim(A)
names(A)
```

---

norm.tensor *Calculate the Euclidean norm or Euclidean operator norm of a tensor or its subtensors*

Description

Calculates the Euclidean norm of a tensor or its subtensors.

Usage

```
norm(X,...)
## S3 method for class 'tensor'
norm(X,i=NULL,...,by=NULL)
opnorm(X,...)
## S3 method for class 'tensor'
opnorm(X,i=NULL,...,by=NULL)
```

Arguments

X	The tensor
i	For norm the dimensions to of the subtensors to be used. If missing the norm of the whole tensor is computed. For opnorm the dimensions of the image.
...	unused
by	the list dimension, if i is not specified the norm is calculated for each of these in parallel.

Details

norm The function computes the Euclidean norm, which is the square root over the sum of all entries and not the operator norm.

opnorm The function computes the Euclidean operator norm, which is largest factor in changing the Euclidean norm, when mapped with the linear mapping corresponding to the tensor.

Value

norm	either a single number giving the norm of the tensor or a tensors with the dimensions i removed containing the individual norms in each entry.
opnorm	a tensor of dimension dim(X)[by] giving the Euclidean operator norm of the tensor (i.e. its largest singular value)

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor

Examples

```
C <- to.tensor(1:20,c(A=4,B=5))
norm(C,"A")
norm(C,2)
norm(C,c("A","B"))
opnorm(C,"A")
```

`one.tensor` *Creates a tensor with all entries 1*

Description

Creates a tensor with all entries one.

Usage

```
one.tensor(d=NULL,dn=NULL)
```

Arguments

`d`	the dimensions of the new tensor
`dn`	the dimnames of the new tensor

Details

$$E_{i_1 \ldots i_n} = 1$$

Value

A tensor with dim d and all elements one

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`

Examples

```
one.tensor(c(a=3,b=3,c=3))
```

pos.tensor *enumeration of index combinations*

Description

This gives all combinations of indices of a tensor with dimension d in the order of the numbers in the memory.

Usage

```
pos.tensor(d)
```

Arguments

d	a dim attribute of a tensor

Details

tensors are stored according to the R-convention that the leftmost index varies fastest.

Value

a matrix with the same number of rows as the tensor has entries an the same number of columns as the tensor has dimensions. Each row represents the index combination of a the corresponding element.

Author(s)

K.Gerald van den Boogaart

See Also

reorder.tensor

Examples

```
(A <- to.tensor(1:20,dim=c(A=2,B=2,C=5)))
pos.tensor(dim(A))
```

power.tensor *Compute the power of a symmetric tensor*

Description

A tensor can be seen as a linear mapping of a tensor to a tensor. If domain and image are the same and the tensor is definite, we can define powers.

Usage

```
power.tensor(X,i,j,p=0.5,by=NULL)
```

Arguments

X	The tensor to be decomposed
i	The image dimensions of the linear mapping
j	The domain dimensions of the linear mapping
p	the power of the tensor to be computed
by	the operation is done in parallel for these dimensions

Details

A tensor can be seen as a linear mapping of a tensor to a tensor. Let denote R_i the space of real tensors with dimensions $i_1 ... i_d$.
To compute a power `dim(X)[i]` and `dim(X)[j]` need to be equal and the tensor symmetric between these dimension. Some exponents are only valid with positive definite mappings. None of these conditions is checked.

Value

a tensor

Note

symmetry of the matrix is assumed but not checked.

Author(s)

K. Gerald van den Boogaart

See Also

svd.tensor,

Examples

```
A <- to.tensor(rnorm(120),c(a=2,b=2,c=5,d=3,e=2))
AAt <- A %e% mark(A,"'",c("a","b"))
AAt

power.tensor(AAt,c("a","b"),c("a'","b'"),-1)

inv.tensor(AAt,c("a","b"))

power.tensor(AAt,c("a","b"),c("a'","b'"),2)
mul.tensor(AAt,c("a","b"),AAt,c("a'","b'"))

power.tensor(power.tensor(AAt,c("a","b"),c("a'","b'"),1/pi),
                     c("a","b"),c("a'","b'"),pi)


AAt <- einstein.tensor(A , mark(A,"'",c("a","b")),by="e")

power.tensor(AAt,c("a","b"),c("a'","b'"),-1,by="e")

inv.tensor(AAt,c("a","b"),by="e")

power.tensor(AAt,c("a","b"),c("a'","b'"),2,by="e")
mul.tensor(AAt,c("a","b"),AAt,c("a'","b'"),by="e")

power.tensor(power.tensor(AAt,c("a","b"),c("a'","b'"),1/pi,by="e"),
c("a","b"),c("a'","b'"),pi,by="e")
```

`reorder.tensor` *Permutation of indices and storage sequence of a tensor*

Description

This permutes tensor dimensions like aperm. However the interface is more flexible since not all dimensions have to given and names can be used instead of numbers.

Usage

```
## S3 method for class 'tensor'
reorder(x,i=NULL,...,by=NULL)
```

Arguments

x	the tensor
i	numeric or character giving dimensions intended to come first
...	further arguments to other instances of the generic function
by	the complement of i, if i is not given

Details

the remaining dimensions keep their relative sequence and follow at the end of the dimension attribute.

Value

`reorder.tensor` returns a tensor equal to x but stored with a different sequence of dimensions.

Author(s)

K.Gerald v.d. Boogaart

See Also

`to.tensor`

Examples

```
A <- to.tensor(1:20,c(A=2,B=2,C=5))
A
reorder(A,"C")
reorder(A,"B")
```

---

reptensor *Repeats a tensor*

Description

The tensor is repeated like a number is repeated by rep and an additional dimension is added to select the different tensors.

Usage

```
## S3 method for class 'tensor'
rep(x,times,pos=1,name="i",...)
```

Arguments

`x`	the tensor to be repeated
`times`	the number of copies that should be created. If `times` is a vector, `x` is seen as a sequence of tensors in dimension `pos` and each of the tensors is repeated according to the corresponding entry of times.
`name`	the name of the additional dimension. if NA no additional dimension is used.
`pos`	the position where the extra dimension should be added
`...`	not used, only here for generic consistency

Details

This function is modeled as much as possible to mimic rep, by repeating tensors rather than numbers. The each argument is not necessary, since sequence of the dimensions can more precisely be controlled by pos. Another problem is the a ambiguity between `rep(x,3)` and `rep(x,c(3))` as a special case of `rep(x,c(3,2))`. If the second is wanted it can be forced by `rep(x,c(3),NA)` through setting the name argument to NA.

Value

A tensor with one additional dimensions of length times.

Author(s)

K. Gerald van den Boogaart

See Also

rep

Examples

```
A <- to.tensor(1:4,c(A=2,B=2))
rep(A,3)
rep(A,3,3,"u")
rep(A,c(2,3))
A <- to.tensor(1:4,c(A=1,B=4))
rep(A,5,pos="A",name=NA)
```

---

riemann.tensor *Tensor multiplication with Riemann's convention*

Description

Multiplies tensors by multiplying over all pairs with one covariate and one contravariate variable with the same name according to Riemann's summing convention.

Usage

```
riemann.tensor(...,only=NULL,by=NULL)
## Methods for class tensor
# x %r% y
## Default method
# x %r% y
```

Arguments

...	some tensors, or a renaming code
only	an optional list of the dimension names to be recognized for duplication to allow parallel processing on lists of tensors
x	a tensor
y	a tensor
by	Riemannian summing is done in parallel in these dimensions.

Details

see mul.tensor on details on tensor multiplication. In einstein.tensor complex operations can be performed by command and renaming code: The arguments are processed from left to right and multiplied. Unnamed attributes are regarded as tensors or scalars and multiplied with the current result by the Riemann summing convention, which means an inner product over all pairs of covariate and contravariate indices with the same name. Named attributes can either have the name diag, which performs a diagmul according to the same-name convention or be of the form A="B" or "A"="B", for which we have two cases. Typically both are given covariate. The first specifies the covariate to be used in the multiplication and the second the contravariate. If both names are present in the current result, an inner multiplication (trace) of on these two dimensions is performed. If only the covariate or the contravariate is present up to this point, the specific dimension is renamed to the second name, but keeps its type. This renaming might be visible in the result or inducing a multiplication according to the Riemann convention later if the other shows up.

Value

the tensor product of all the tensors along all duplicate dimensions.

Author(s)

K. Gerald van den Boogaart

See Also

mul.tensor, to.tensor, riemann.tensor

Examples

```
A <- to.tensor(1:20,c(U=2,"^V"=2,W=5))
B <- to.tensor(1:20,c("^U"=2,V=2,Q=5))
riemann.tensor(A,B)
A %r% B
```

sequencing *Working with index sequences*

Description

In typical tensor notation the indices are not identified by names but by positions. The operators allow to identify names and positions transparently during calculation.

Usage

```
## Methods for class tensor
# x $ y
# x ^ y
# x | y
renamefirst.tensor(x,y)
```

Arguments

x	A tensor
y	Typically a character vector specifying a sequence of names for the tensor. The names can be specified in various ways: The following specifications are equal and specify a sequence of the names i,j and k: xijk, xi.j.k, i.j.k., x"$ijk", x^"i.j.k", x^c("i","j","k"),x^c("i.j","k"), x^c("$i.j","k"),x^c("$ij","k"), x^c("$","ijk") In general names are separated by dots. All notations with \$ either as operator or as the first character of the first string allow to omit the dots assuming that all names are single character. If any dot is present all dots must be given. The difference of \$ and \^ is that the first accepts a name and the second an character valued expression. Multi letter indices like "alpha","beta","gamma" can only be given in the dot-free version of the notation making the following specifications equal: x$alpha.beta.gamma, alpha.beta.gamma., x^"$alpha.beta.gamma", x^"alpha.beta.gamma", x^c("alpha","beta","gam x^c("alpha.beta","gamma"), x^c("$alpha.beta","k"), x^c("$","alpha.beta.gammak") The specification for | is equal to that for ^.

Details

These functions are used to mimic the mathematical notation in tensor analysis. Formulae of the form (with Einstein convention):

$$E_{ijk} = A_{ihl} C_{hj} C_{lk}$$

with defined tensors A_{ijk} and C_{ij} can be given the simple form
E <- A$ihl %e% C$hj %e% C$lk |"$ijk"
or alternatively for multi letter names:
E <- A$i.h.l %e% C$h.j %e% C$l.k |"i.j.k"
or more flexible in computation with arguments I,J,K:

`E <- A^c(I,"h.l") %e% C^c("h",J) %e% C^c("l",K) | c(I,J,K)`
The `$` or `^` binds to the tensors with high precedence and renames the first elements. The `|` binds with very low precedence and reorders the tensor according to the assumed index sequence of the result afterwards.

Value

A tensor of the same shape as x but with reordered dimensions (for `|`) or renamed dimensions (for the others)

Author(s)

K. Gerald van den Boogaart

See Also

`reorder.tensor`, `names<-.tensor`, `[[.tensor`

Examples

```
A <- to.tensor(1:20,c(i=5,j=2,k=2))
C <- to.tensor(1:4,c(i=2,j=2))
E <- A$ihl %e% C$hj %e% C$lk |"$ijk"
E
# Same as:
E2 <- reorder.tensor(A[[j=~h,k=~l]] %e% C[[i=~h]] %e% C[[i=~l,j=~k]],c("i","j","k"))
E-E2
E <- A$i.h.l %e% C$h.j %e% C$l.k |"i.j.k"
E
E-E2
E <- A^"i.h.l" %e% C^"h.j" %e% C^"l.k" |"i.j.k"
E
E-E2
```

---

slice.tensor *Working with the indices of a tensor (accessing, slicing, renaming, ...)*

Description

Indexing of tensors allows beside the ordinary selection of ranges of indices the renaming of indices. The functions are mainly here to keep the the tensor property of the results.

Usage

```
slice.tensor(X,i,what,drop=FALSE)
## Methods for class tensor
# X[...,drop=TRUE]
# X[...,drop=TRUE] <- value
# X[[...,drop=TRUE]]
# X[[...,drop=TRUE]] <- value
```

Arguments

X	A tensor
i	an index given as number or character
what	levels of the index, a number or a character from dimnames
drop	a boolean, if true, indices with only a single level are removed
...	arguments of the form name=indices, and for the [[ ]] functions it also allowed to give names from the corresponding dimnames name=c("a","b") to select indices by names or name=~newname to rename dimensions, the first use makes a usual array access in the given dimension, where [[ ]] only supports a single index, while [ ] allows vectors. The other type changes the names.

Details

The functions allow to rename dimensions and to take select a part of the tensor.

Value

a new tensor with dimensions renamed or individual levels selected

Author(s)

K. Gerald van den Boogaart

See Also

einstein.tensor

Examples

```
A <- to.tensor(1:20,c(A=2,B=2,C=5))
A[C=1]
A[C=1:3]
A[[B=~b]]                  # renaming dimensions
A[[B=~b,A=~aaa]]
A[[B=~b,A=~aaa,aaa=1]]
A[[A=1,B=~gamma]][C=1:2]
A
```

---

solve.tensor *Solving linear equations with tensors*

Description

We can formulate linear equation systems with tensors. This functions solves these systems or gives a least squares fit of minimal norm.

Usage

```
 ## S3 method for class 'tensor'
solve(a,b,i,j=i,...,allowSingular=FALSE,eps=1E-10,by=NULL)
```

Arguments

a	The a of ax=b
b	The a of ax=b
i	The dimensions of the equation in a
j	The dimensions of the equation in b
allowSingular	A boolean, indicating the that a least squares fit should be generated with singular equations systems.
...	further arguments for generic use
eps	The limit for the smallest singular value in inversion
by	the operation is done in parallel for these dimensions

Details

A tensor can be seen as a linear mapping of a tensor to a tensor. Let denote R_i the space of real tensors with dimensions $i_1 ... i_d$.

solve.tensor Solves the equation for $a_{i_1 ... i_d k_1 ... k_p}$, $b_{j_1 ... j_d l_1 ... l_q}$ and $x_{k_1 ... k_p l_1 ... l_q}$ the equation

$$\sum_{k_1,...,k_p} a_{i_1 ... i_d k_1 ... k_p} x_{k_1 ... k_p l_1 ... l_q} = b_{j_1 ... j_d l_1 ... l_q}$$

.

Value

a tensor such that ax=b as good as possible for each combination of by values.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor, svd.tensor, inv.tensor, chol.tensor, power.tensor

Examples

```
R1  <- matrix(rnorm(9),nrow=3)
R1i <- solve(R1)
R2 <- to.tensor(R1,c(a=3,b=3),what=1:2)
R2i <- to.tensor(R1i,c(b=3,a=3),what=1:2)

inv.tensor(R2,"a","b") - R2i
inv.tensor(R2,"a","b",allowSingular=TRUE) - R2i
```

```
inv.tensor(rep(R2,4,1,"K"),"a","b",by="K") - rep(R2i,4,1,"K")
inv.tensor(rep(R2,4,1,"K"),"a","b",by="K",allowSingular=TRUE) - rep(R2i,4,3,"K")

R3 <- to.tensor(rnorm(15),c(a=3,z=5))

mul.tensor(R2i,"b",mul.tensor(R2,"a",R3)) # R3

solve.tensor(R2i,R3[[z=1]],"a")
mul.tensor(R2,"a",R3[[z=1]])

solve.tensor(R2i,R3,"a")
mul.tensor(R2,"a",R3)

solve.tensor(R2i,R3[[z=1]],"a",allowSingular=TRUE)
mul.tensor(R2,"a",R3[[z=1]])

solve.tensor(R2i,R3,"a",allowSingular=TRUE)
mul.tensor(R2,"a",R3)

solve.tensor(rep(R2i,4,1,"K"),R3[[z=1]],"a",by="K")
rep(mul.tensor(R2,"a",R3[[z=1]]),4,1,"K")

solve.tensor(rep(R2i,4,1,"K"),rep(R3[[z=1]],4,1,"K"),"a",by="K")
rep(mul.tensor(R2,"a",R3[[z=1]]),4,1,"K")
```

`svd.tensor` *Singular value decomposition of tensors*

Description

A tensor can be seen as a linear mapping of a tensor to a tensor. This function computes the singular value decomposition of this mapping

Usage

```
svd.tensor(X,i,j=NULL,...,name="lambda",by=NULL)
```

Arguments

`X`	The tensor to be decomposed
`i`	The image dimensions of the linear mapping
`j`	The coimage dimensions of the linear mapping
`name`	The name of the eigenspace dimension. This is the dimension created by the decompositions, in which the eigenvectors are e_i
`...`	further arguments for generic use
`by`	the operation is done in parallel for these dimensions

Details

A tensor can be seen as a linear mapping of a tensor to a tensor. Let denote R_i the space of real tensors with dimensions $i_1 ... i_d$.

svd.tensor Computes a singular value decomposition $u_{i_1 ... i_d \lambda}$,d_λ, $v_{j_1 ... j_l} \lambda$ such that u and v correspond to orthogonal mappings from R_λ to R_i or R_j respectively.

Value

a tensor or in case of svd a list u,d,v, of tensors like in svd.

Author(s)

K. Gerald van den Boogaart

See Also

to.tensor, to.matrix.tensor, inv.tensor, solve.tensor

Examples

```
# SVD
A <- to.tensor(rnorm(120),c(a=2,b=2,c=5,d=3,e=2))

SVD <- svd.tensor(A,c("a","d"),c("b","c"),by="e")
dim(SVD$v)
# Property of decomposition
einstein.tensor(SVD$v,diag=SVD$d,SVD$u,by="e") # A
# Property of orthogonality
SVD$v %e% SVD$v[[lambda=~"lambda'"]]         # 2*delta.tensor(c(lambda=6))
SVD$u %e% SVD$u[[lambda=~"lambda'"]]         # 2*delta.tensor(c(lambda=6)))
SVD$u %e% mark(SVD$u,"'",c("a","d"))  # 2*delta.tensor(c(a=2,d=3)))
```

---

to.matrix.tensor *The matrix corresponding to a tensor seen as a linear mapping of tensors.*

Description

A tensor can be seen as a linear mapping of a tensor to a tensor. This function gives the corresponding matrix of the mapping.

Usage

```
to.matrix.tensor(X,i,j,by=NULL)
```

Arguments

X	The tensor
i	The image indices of the linear mapping
j	The domain indices of the linear mapping
by	the operation is done in parallel for these dimensions

Details

A tensor can be seen as a linear mapping of a tensor to a tensor. This function computes the corresponding matrix, mapping the entries of the domain tensor to the entries of the image tensor.

Value

if no by is given a matrix. Otherwise a tensor of level `2+length(dim(X))[by]` giving matrices for each specification of the by dimensions.

Author(s)

K. Gerald van den Boogaart

See Also

`to.tensor`, `solve.tensor`, `inv.tensor`, `svd.tensor`

Examples

```
A <- reorder.tensor(to.tensor(1:30,c(a=2,b=3,c=5)),c("c","a","b"))

to.matrix.tensor(A,"a",c("b","c"))              # matrix(1:30,nrow=2)

to.matrix.tensor(A,c("a","b"),c("c"))           # matrix(1:30,nrow=6)

to.matrix.tensor(A,c("a","b"),by=c("c")) # structure(1:30,dim=c(6,1,5)))
to.matrix.tensor(A,c("a"),by=c("c"))     # structure(1:30,dim=c(2,3,5)))
```

---

to.tensor *Creates a tensor object*

Description

Constructs a "tensor". A tensor is the generalization of vectors and matrices to multi-index arrays.

Usage

```
to.tensor(X,...)
## Default S3 method:
to.tensor(X,dims=NULL,ndimnames=NULL,what=1,addIndex=FALSE,...)
```

Arguments

X	the numeric data with the entries of the tensor. If the object is already a tensor only the subtensors given by the dimension what are converted
dims	These dimensions to be added for the new tensor. If the object is to big or addIndex an extra dimension is added.
ndimnames	The new dimnames to be used
what	a numeric or character vector specifying the dimensions to be removed.
addIndex	boolean or character, FALSE says no additional dimension, or string to give the name of the dimension
...	further arguments to other instances of the generic function

Details

This package provides a class "tensor" allowing easy computation regarding tensorial computation in the Einstein convention and allows an easier control of the computation than aperm and tensor. The package is made to work with things like matrices of matrices and linear mapping of matrices to matrices, etc.

A tensor is a multidimensional array, with specific mathematical meaning, generalizing vectors and matrices. Tensors can be added, subtracted and multiplied and used in linear equations. While two matrices A,B are commonly only multiplied in two ways A%*%B or B%*%A and have some more t(A)%*%B, B%*%t(A), sum(A*B), sum(A*t(B)),kronecker(A,B) the tensor calculus brings all of them into a organized system.
An important aspect for that is the name of its dimensions. Thus we are not bound to work with rows and columns, but can name the dimensions to be multiplied. This leads to much more organized computation of linear mappings of matrices or datasets of matrices or other genuine tensor arithmetic gets involved.
The package provides a full linear algebra support of tensors including tensor addition, tensor multiplication, norms, deltatensors, binding, inversion, normalization, Einstein summing convention, trace, , dimension renaming, smart display of tensors, renaming and reshaping, solving equation system and giving decompositions and parallelized data processing ,

Value

a tensor of the specified shape

Note

This constructor is not called tensor() according to the general convention of constructors to avoid conflicts with the tensor multiplication routine in the tensor package

Author(s)

K. Gerald van den Boogaart

See Also

tensorA, level.tensor, diag.tensor, norm.tensor drag.tensor, one.tensor, mul.tensor, %e%, %r%, , drag.tensor, , trace.tensor, solve.tensor, svd.tensor, mean.tensor

Examples

```
A <- to.tensor(1:20,c(U=2,V=2,W=5))
B <- to.tensor(1:20,c(U=2,VV=2,WW=5))
A %e% B
```

`toPos.tensor` *get the position of an index of tensor*

Description

Calculates the position of a tensor index, which specified in any possible way.

Usage

```
toPos.tensor(M,l=NULL,mnames=names(dim(M)),by=NULL,...,both=FALSE,missing.ok=FALSE)
```

Arguments

`M`	a tensor
`l`	a vector specifying the indices as positions or names
`mnames`	The names of the indices of the tensor. This can be specified instead of M.
`both`	Matches the index in its covariate and contravariate form.
`by`	the list dimension, all operations are done in parallel for all levels of these dimensions. Thus in the case of toPos all other dimensions are returned if they are not specified.
`...`	not used
`missing.ok`	If TRUE does give an error on missing dimension. Rather returns NA in that place.

Details

The function is only here to provide a consistent interface which provides the same functionality for positions and characters.

Value

a numeric vector giving the positions of the dimensions selected.

Author(s)

K. Gerald van den Boogaart

Examples

```
A <- to.tensor(1:30,c(a=2,b=3,c=5))
toPos.tensor(A,c("b","c"))
toPos.tensor(A,c(2,1))     # only returns the values
toPos.tensor(A,c("^a"),both=TRUE)
```

`trace.tensor` *Collapse a tensor*

Description

Collapses the tensor over dimensions i and j. This is like a trace for matrices or like an inner product of the dimensions i and j.

Usage

```
trace.tensor(X,i,j)
```

Arguments

`X`	the tensor
`i`	a numeric or character vector of dimensions of `X`, used for the inner product.
`j`	a numeric or character vector of dimensions of `X` with the same length but other elements than i.

Details

Let be

$$X_{i_1 \ldots i_n j_1 \ldots j_n k_1 \ldots k_d}$$

the tensor. Then the result is given by

$$E_{k_1 \ldots k_d} = sum_{i_1 \ldots i_n} X_{i_1 \ldots i_n i_1 \ldots i_n k_1 \ldots k_d}$$

With the Einstein summing convention we would write:

$$E_{k_1 \ldots k_d} = X_{i_1 \ldots i_n j_1 \ldots j_n k_1 \ldots k_d} \delta_{i_1 j_1} \ldots \delta_{i_n j_n} E_{k_1 \ldots k_d} = X_{i_1 \ldots i_n j_1 \ldots j_n k_1 \ldots k_d} \delta_{i_1 j_1} \ldots \delta_{i_n j_n}$$

Value

A tensor like X with the i and j dimensions removed.

Author(s)

K. Gerald van den Boogaart

See Also

`mul.tensor`, `to.tensor`

Examples

```
A <- to.tensor(1:20,c(i=2,j=2,k=5))
A
trace.tensor(A,"i","j")
```

`tripledelta.tensor` *A tensor with entry 1 if and only if three indices are equal*

Description

The tensor mapping a tensor of dimension d to its corresponding diagonal tensor of dimension c(d',d*)

Usage

```
tripledelta.tensor(d,mark1="'",mark2="*",dn=NULL)
```

Arguments

`d`	the first of three dimension vectors
`mark1`	the mark for the second dimension vectors
`mark2`	the mark for the third dimension vectors
`dn`	list of character vectors, optional dimnames

Details

The tripledelta is the tensor mapping a tensor to a corresponding diagonal tensor.

Value

The tensor given by:

$$E_{i_1 \ldots i_n j_1 \ldots j_n k_1 \ldots k_n} = \delta_{i_1 j_1} \delta_{i_1 k_1} \ldots \delta_{i_n j_n} \delta_{i_n k_1}$$

Author(s)

K. Gerald van den Boogaart

See Also

`delta.tensor`, `diag.tensor`

Examples

```
tripledelta.tensor(3)
```

undrop.tensor *Adds a spurious dimension to a tensor*

Description

A dimension of length 1 is added a given position to a tensor

Usage

```
undrop.tensor(A,name,pos=1)
```

Arguments

A	the tensor
name	the name of the dimension to be added
pos	the position, where to insert the new dimension

Details

The function is a pure convenience function.

Value

A tensor with one extra dimension of length 1 with name name at position pos.

Author(s)

K. Gerald van den Boogaart

Examples

```
A <- to.tensor(1:4,c(a=2,b=2))
undrop.tensor(A,"i")
```

untensor *Removes indices/dimensions from a tensor*

Description

untensor is more or less the inverse of to.tensor. It flattens tensorial dimensions. However the result is still a tensor.

Usage

```
untensor(X,i=NULL,name=NULL,pos=1,by=NULL)
```

Arguments

X	the tensor
i	the names of the dimensions to be removed and combined to a single new one as a character vector or a named list of character vectors if the remove should be done in multiple chunks. pos and name is in this case ignored.
name	the name of the new dimension to replace the others
pos	where to insert the the new dimension
by	if i not given the dimensions to be kept

Details

The dimensions to be removed are gathered and

Value

a tensor with the dimensions i removed.

Author(s)

K.Gerald van den Boogaart

See Also

to.tensor

Examples

```
A <- to.tensor(1:64,c(a=2,b=2,c=2,d=2,e=2,f=2))
untensor(A,list(c(1,5),c(2,4)),name=c("i","j"))
untensor(A,by=c("c","f"))
untensor(A,c("a","d"))
```

Index

∗ **algebra**
names.tensor, 25
norm.tensor, 26
to.tensor, 41
∗ **arith**
add.tensor, 5
as.tensor, 6
bind.tensor, 7
chol.tensor, 8
delta.tensor, 10
diag.tensor, 11
diagmul.tensor, 12
drag.tensor, 13
einstein.tensor, 15
inv.tensor, 17
is.tensor, 18
level.tensor, 19
margin.tensor, 20
mark.tensor, 21
mul.tensor, 24
one.tensor, 28
power.tensor, 30
reptensor, 32
riemann.tensor, 33
sequencing, 35
slice.tensor, 36
solve.tensor, 37
svd.tensor, 39
to.matrix.tensor, 40
toPos.tensor, 43
trace.tensor, 44
tripledelta.tensor, 45
undrop.tensor, 46
untensor, 46
∗ **array**
pos.tensor, 29
reorder.tensor, 31
∗ **math**
ftable.tensor, 16
∗ **multivariate**
mean.tensor, 22
∗ **package**
tensorA-package, 2
*.tensor *(add.tensor)*, 5
+.tensor *(add.tensor)*, 5
-.tensor *(add.tensor)*, 5
/.tensor *(add.tensor)*, 5
[.tensor *(slice.tensor)*, 36
[<-.tensor *(slice.tensor)*, 36
[[.tensor, *5*, *36*
[[.tensor *(slice.tensor)*, 36
[[<-.tensor *(slice.tensor)*, 36
$.tensor, *4*
$.tensor *(sequencing)*, 35
%e% *(einstein.tensor)*, 15
%r% *(riemann.tensor)*, 33
%e%, *25*, *42*
%r%, *25*, *42*
^.tensor *(sequencing)*, 35

add.tensor, *4*, *5*, 5
as.contravariate *(drag.tensor)*, 13
as.covariate *(drag.tensor)*, 13
as.tensor, 6

base, *8*
bind.tensor, 7

chol.tensor, 8, *38*
contraname *(drag.tensor)*, 13

delta.tensor, 10, *22*, *45*
diag.tensor, *4*, 11, *22*, *42*, *45*
diagmul.tensor, *4*, 12, *25*
dim<-.tensor *(names.tensor)*, 25
dimnames.tensor *(names.tensor)*, 25
dimnames<-.tensor *(names.tensor)*, 25
drag.tensor, *5*, 13, *42*

einstein.tensor, *4*, *5*, 15, *25*, *37*

ftable, *17*
ftable.tensor, 16

inv.tensor, 17, *38*, *40, 41*
is.contravariate *(*drag.tensor*)*, 13
is.covariate *(*drag.tensor*)*, 13
is.tensor, 18

level.tensor, *3*, 19, *42*

margin.tensor, 20
mark *(*mark.tensor*)*, 21
mark.tensor, 21
mean.tensor, 22, *42*
mul.tensor, *4, 5*, *15, 16*, 24, *26*, *34*, *42*, *44*

names.tensor, 25
names<-.tensor *(*names.tensor*)*, 25
norm *(*norm.tensor*)*, 26
norm.tensor, 26, *42*

one.tensor, 28, *42*
opnorm *(*norm.tensor*)*, 26

pos.tensor, 29
power.tensor, 30, *38*

renamefirst.tensor *(*sequencing*)*, 35
reorder.tensor, *29*, 31, *36*
rep, *33*
rep.tensor *(*reptensor*)*, 32
reptensor, 32
riemann.tensor, *14*, *16*, *25*, 33, *34*

sequencing, 35
slice.tensor, 36
solve.tensor, *18*, *25*, 37, *40–42*
svd, *40*
svd.tensor, *4*, *9*, *18*, *30*, *38*, 39, *41, 42*

Tensor, *14*
Tensor *(*tensorA-package*)*, 2
tensor *(*tensorA-package*)*, 2
tensorA, *23*, *42*
tensorA *(*tensorA-package*)*, 2
tensorA-package, 2
to.matrix.tensor, *40*, 40
to.tensor, *3*, *5–7*, *9–12*, *14*, *16*, *18–20*, *25*, *27, 28*, *32*, *34*, *38*, *40, 41*, 41, *44*, *47*
toPos.tensor, 43

trace.tensor, *4*, *42*, 44
tripledelta.tensor, 45

undrop.tensor, 46
untensor, 46

var.tensor *(*mean.tensor*)*, 22