Package 'arrayhelpers'

February 19, 2015

Type Package

Title Convenience functions for arrays

Description Some convenient functions to work with arrays

Maintainer C. Beleites <Claudia.Beleites@ipht-jena.de>

Author C. Beleites <Claudia.Beleites@ipht-jena.de>

Version 0.76-20120816

Date 2012-08-16

License GPL

LazyLoad yes

LazyData yes

Depends methods

Suggests svUnit

URL http://arrayhelpers.r-forge.r-project.org/

Collate 'arrayhelpers.R' 'apply.R' 'array2df.R' 'array2vec.R' 'colSums.R' 'countrows.R' 'delold.R' 'dropdimnames.R' 'ensuredim.R' 'groupsum.R' 'init.R' 'makeNd.R' 'ndim.R' 'numericindices.R' 'restoredim.R' 'rowsum.R' 'slice.R' 'stack.R' 'ta.R' 'validate.R'

Repository CRAN

Date/Publication 2012-08-17 14:32:11

NeedsCompilation no

R topics documented:

`arrayhelpers-package` *Little helper functions to work with arrays*

Description

Little helper functions to work with arrays

`array2df` *array2df*

Description

array2df: Convert multidimensional array into matrix or data.frame The "wide-format" array is converted into a "long-format" `matrix` or `data.frame`.

Usage

```
array2df(x, levels, matrix = FALSE,
  label.x = deparse(substitute(x)), na.rm = FALSE)
```

Arguments

`x`	`array`
`levels`	`list` with the levels for the dimensions of `x`. If `levels[[i]]` is `NULL` no column is produced for this factor. If `levels[[i]]` is `NA`, the result column is a numeric with range from `1` to `dim (x)[i]` If `levels[[i]]` is `TRUE`, the levels are taken from the dimnames. `names(levels)` yield the resulting column names.
`matrix`	If `TRUE`, a numeric `matrix` rather than a `data.frame` is returned.
`label.x`	Name for the column containing the `x` values.
`na.rm`	should rows where the value of `x` is `NA` be removed?

Details

If the resulting `data.frame` is too large to fit in memory, a `matrix` might help.

The main benefit of this function is that it uses matrices as long as possible. This can give large advantages in terms of memory consumption.

Value

A data.frame or matrix with `prod (dim (x))` rows and `length (dim (x)) + 1` columns.

Author(s)

Claudia Beleites

See Also

`stack`

Examples

```
a <- arrayhelpers:::a
a
array2df (a)
array2df (a, matrix = TRUE)

array2df (a, levels = list(NULL, x = NA, c = NULL), label.x = "value")

array2df (a, levels = list(NULL, x = TRUE, c = c ("foo", "bar")), label.x = "value")

summary (array2df (a,
                   levels = list(NULL, x = NA, c = c ("foo", "bar")),
                   label.x = "value"))

summary (array2df (a,
                   levels = list(NULL, x = NA, c = c ("foo", "bar")),
                   label.x = "value",
                   matrix = TRUE))
```

---

array2vec *Converting array and vector Indices Calculate the vector index from array indices, and vice versa.*

Description

`arrays` are `numerics` with a `dim` attribute and are stored with the first index moving fastest (i.e. by column). They can be indexed both ways.

Usage

```
array2vec(iarr, dim)

vec2array(ivec, dim)
```

Arguments

iarr	vector with the indices into the array dimensions
dim	vector with the array dimensions, as returned by `dim (x)`
ivec	scalar with the index into the vector

Value

`array2vec` returns a scalar, `vec2array` a `matrix`.

Author(s)

C. Beleites

See Also

see `Extract` on the difference of indexing an `array` with a vector or a `matrix`.

Examples

```
arr <- array (rnorm (24), dim = 2 : 4)
arr

v <- matrix(c(2, 2, 2), nrow = 1)
i <- array2vec (v, dim = dim (arr))
i
arr[v]
arr[i]

arr[c(2, 2, 2)] ## indexing with a vector
arr[2]
i <- 14
v <- vec2array (i, dim = dim (arr))
v
arr [v]
arr [i]
```

arrayhelpers.unittest *Run the unit tests*

Description

Run the unit tests attached to the functions via svUnit

Usage

```
arrayhelpers.unittest()
```

Value

invisibly TRUE if the tests pass, NA if svUnit is not available. Stops if errors are encountered.

Author(s)

Claudia Beleites

See Also

svUnit

colSums *Row and column sums and means for numeric arrays.*

Description

These functions extend the respective base functions by (optionally) preserving the shape of the array (i.e. the summed dimensions have length 1).

Usage

```
  ## S4 method for signature 'matrix'
colSums(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  colSums.AsIs(x, ...)

  ## S4 method for signature 'array'
colSums(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  ## S4 method for signature 'matrix'
colMeans(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)
```

```
  colMeans.AsIs(x, ...)

  ## S4 method for signature 'array'
colMeans(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  ## S4 method for signature 'matrix'
rowSums(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  rowSums.AsIs(x, ...)

  ## S4 method for signature 'array'
rowSums(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  ## S4 method for signature 'matrix'
rowMeans(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)

  rowMeans.AsIs(x, ...)

  ## S4 method for signature 'array'
rowMeans(x, na.rm = FALSE, dims = 1L,
    drop = TRUE)
```

Arguments

x	an array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame.
na.rm	logical indicating treatment of missing values
dims	integer: Which dimensions are regarded as 'rows' or 'columns' to sum over. For row*, the sum or mean is over dimensions dims + 1, ...; for col* it is over dimensions 1 : dims.
...	the signature = "AsIs" methods hand on all parameters
drop	If FALSE, the number of dimensions is retained: the length of the dimensions that are summed or averaged is set to 1. TRUE yield the same behaviour as colSums

Value

like colSums if drop = TRUE, otherwise an array where the summed dimensions have length 1.

Author(s)

Claudia Beleites

See Also

colSums

Examples

```
a <- array (1 : 24, 4 : 2)
a

rowSums (a)
rowSums (a, drop = FALSE)

colSums (a)
colSums (a, drop = FALSE)

colSums (a, dim = 2)
colSums (a, dim = 2, drop = FALSE)
```

---

countRows *Count equal rows*

Description

matrices are converted to data.frame.

Usage

```
countRows(x)
```

Arguments

x	the matrix or data.frame

Value

data frame with unique rows, their counts and indices into the original data.frame

Note

this function is subject to changes in the future.

Author(s)

Claudia Beleites

delold *Strip the attributes keeping track of the former shape*

Description

Convenient for printing

Usage

```
delold(a)
```

Arguments

a	the array

Value

a stripped of the old.* attributes.

Author(s)

Claudia Beleites

Examples

```
a <- arrayhelpers:::a
makeNd (a, 2)
delold (makeNd (a, 2))
```

dropdimnames *Drop dimnames if all elements are* NULL

Description

Drop dimnames if all elements are NULL

Usage

```
dropdimnames(x)

lon(l)
```

Arguments

x	object
l	list

Value

object without empty dimnames

`lon`: `NULL` if all elements of `dn` are `NULL`, otherwise `dn`

Author(s)

Claudia Beleites

`ensuredim` *Enforce array and convert to vector if appropriate*

Description

`ensuredim` turns vectors into 1d-arrays, and leaves arrays unchanged. `drop1d` is the inverse: it converts 1d arrays into vectors.

Usage

```
ensuredim(x)

drop1d(x, drop = TRUE)
```

Arguments

`x`	vector (or array)
`drop`	if `TRUE`, 1d arrays are converted into vectors

Value

`esuredim` array of at least one dimension

`drop1d` vector, if `x` had only 1 dimension

Author(s)

Claudia Beleites

Claudia Beleites

groupsum *Extension of* rowsum

Description

groupsum extends rowsum: it allows group to be an array of the same shape as x.

Usage

```
groupsum(x, group = NULL, dim = 1L, reorder = TRUE,
  na.rm = FALSE, ..., drop = !is.array(x))
```

Arguments

x	array to be rowsummed
group	grouping variable (integer or factor) indicating groups of samples.
dim	along which dimension should the group sums be taken? (default: rows)
reorder	should the groups be ordered? see rowsum
na.rm	shoud NAs be removed?
...	ignored
drop	should 1d arrays drop to vectors?

Value

like rowsum, but further dimensions of the array are preserved.

Author(s)

Claudia Beleites

See Also

rowsum rowsum

makeNd *Ensure/collapse an array into* n *dimensions and restore the old dimensions*

Description

nameNd ensures a given number of dimensions: If a has less than N dimensions, new dimensions of length 1 are appended. If a has more than N dimensions, the supernumerary dimensions are collapsed onto the last dimension.

Attributes old.dim and old.dimnames are used by default. restoredim is the inverse of makeNd.

Usage

```
makeNd(a, N)

restoredim(a, old = NULL, n = 1L, ..., usedim = TRUE,
  fromend = FALSE, drop = FALSE)
```

Arguments

a	an array (matrix, vector)
N	the desired number of dimensions, 0 to remove the dim and dimnames attributes (i.e. to create a vector).
old	list containing a list with (possibly) elements dim, dimnames, and names. The nth last element of this list is used.
n	how many makeNdim steps to go back?
...	ignored
usedim	use only the specified dimensions
fromend	if TRUE, numeric usedim are counted from the end, see details.
drop	should 1d arrays drop to vectors?

Details

Note that missing attributes as well as old.dim = NULL produce a (dimensionless) vector. This is also the case if a lost the old.* attributes during computations like as.numeric, c, etc..

fromend together with numeric usedim specifies dimensions counting from the end. E.g. fromend = TRUE and usedim = 1 : 3 for an array to be restored to 10d means restoring dimensions 8 : 10. fromend = TRUE and usedim = -(1 : 3) restores dimensions 1 to 7.

Value

N-dimensional array

an array

Author(s)

Claudia Beleites

Claudia Beleites

Examples

```
v <- arrayhelpers:::v
v
makeNd (v, 1)
dim (makeNd (v, 1))
dim (makeNd (v, 3))

m <- arrayhelpers:::m
m
makeNd (m, 1)
dim (makeNd (m, 1))
makeNd (m, 0)
dim (makeNd (m, 0))
makeNd (m, 3)

a <- arrayhelpers:::a
a
dim (makeNd (a, 1))
dim (makeNd (a, 0))
makeNd (a,  2)
makeNd (a, -2)
makeNd (a, -4)
makeNd (a, 3);
a <- array (1 : 24, 4 : 3)
a
restoredim (makeNd (a, 0))

x <- makeNd (a, 0)
attr (x, "old")
```

---

ndim *number of dimensions*

Description

number of dimensions

Usage

```
ndim(a)
```

Arguments

a	vector, matrix, or array
...	indexing instructions. The names of the arguments specify the dimension (i = 1st, j = 2nd, ...). The indexing expressions are the same as for [

Value

integer: length of dim attribute

Author(s)

Claudia Beleites

---

numericindex *Convert character or logical indices to numeric*

Description

Convert character or logical indices to numeric

Usage

```
numericindex(x, i, n = names(x))
```

Arguments

x	the object that is to be indexed
n	names of the object
i	the indices to be converted

Value

numeric indices

Author(s)

Claudia Beleites

peek *A little stack.*

Description

TODO: implement as reference class? Note: pop only removes elements. To retrieve them, use peek.

Usage

```
peek(x, an, n = 1L)

pop(x, an, n = 1L)

push (x, an) <- value
```

Arguments

x	the object
an	attribute holding the stack
n	numer of element to peek at and numer of elements to pop (delete), respectively
value	list of things to push on the stack.

Value

push and pop: the object with stack in list an pushed/popped by the n elements

peek: the nth stack element (without popping!)

Author(s)

Claudia Beleites

rowsum *rowsum for arrays*

Description

This function extends the base function rowsum.

Usage

```
  ## S4 method for signature 'array'
rowsum(x, group, reorder = TRUE,
    na.rm = FALSE, ...)
```

Arguments

x	array to be rowsummed
group	grouping variable (integer or factor) indicating groups of samples.
reorder	should the groups be ordered? see rowsum
na.rm	shoud NAs be removed?
...	ignored

Value

like rowsum, but further dimensions of the array are preserved.

Author(s)

Claudia Beleites

See Also

rowsum

---

slice *alternative to index an array*

Description

slice is an alternative interface to [(extract). Dimensions to index must be given by name, i for the first, j for the second and so on.

Usage

```
slice(a, ..., drop = TRUE)
```

Arguments

a	vector, matrix, or array
...	indexing instructions. The names of the arguments specify the dimension (i = 1st, j = 2nd, ...). The indexing expressions are the same as for [
drop	see [

Value

array

Author(s)

Claudia Beleites

Examples

```
slice (arrayhelpers:::a, j = 3 : 2)
```

`ta` *Transpose arrays*

Description

This function provides transposing of arrays or vectors as swapping their first two dimensions. `t (array)` can be enabled via `setMethod`, see the example.

Usage

```
ta(x)
```

Arguments

`x`	an array

Value

the array with the first two dimensions swapped.

Author(s)

Claudia Beleites

See Also

`t`

Examples

```
a <- array (1 : 24, 4:2)
a
ta (a)

setMethod ("t", "array", ta)
t (a)
removeMethod ("t", "array")
```

Index