Package 'abind'

July 22, 2025

Version 1.4-8

Date 2024-09-08

Title Combine Multidimensional Arrays

Maintainer Tony Plate <`tplate@acm.org`>

Description Combine multidimensional arrays into a single array. This is a generalization of 'cbind' and 'rbind'. Works with vectors, matrices, and higher-dimensional arrays (aka tensors). Also provides functions 'adrop', 'asub', and 'afill' for manipulating, extracting and replacing data in arrays.

Depends R (>= 1.5.0)

Imports methods, utils

License MIT + file LICENSE

NeedsCompilation no

Author Tony Plate [aut, cre],
Richard Heiberger [aut]

Repository CRAN

Date/Publication 2024-09-12 17:40:54 UTC

Contents

`abind` *Combine multi-dimensional arrays*

Description

Combine multi-dimensional arrays. This is a generalization of cbind and rbind. Takes a sequence of vectors, matrices, or arrays and produces a single array of the same or higher dimension.

Usage

```
abind(..., along=N, rev.along=NULL, new.names=NULL, force.array=TRUE,
      make.names=use.anon.names, use.anon.names=FALSE,
      use.first.dimnames=FALSE, hier.names=FALSE, use.dnns=FALSE)
```

Arguments

`...`	Any number of vectors, matrices, arrays, or data frames. The dimensions of all the arrays must match, except on one dimension (specified by `along=`). If these arguments are named, the name will be used for the name of the dimension along which the arrays are joined. Vectors are treated as having a dim attribute of length one. Alternatively, there can be one (and only one) list argument supplied, whose components are the objects to be bound together. Names of the list components are treated in the same way as argument names.
`along`	(optional) The dimension along which to bind the arrays. The default is the last dimension, i.e., the maximum length of the dim attribute of the supplied arrays. `along=` can take any non-negative value up to the minimum length of the dim attribute of supplied arrays plus one. When `along=` has a fractional value, a value less than 1, or a value greater than N (N is the maximum of the lengths of the dim attribute of the objects to be bound together), a new dimension is created in the result. In these cases, the dimensions of all arguments must be identical.
`rev.along`	(optional) Alternate way to specify the dimension along which to bind the arrays: `along = N + 1 - rev.along`. This is provided mainly to allow easy specification of `along = N + 1` (by supplying `rev.along=0`). If both `along` and `rev.along` are supplied, the supplied value of `along` is ignored.
`new.names`	(optional) If new.names is a list, it is the first choice for the dimnames attribute of the result. It should have the same structure as a dimnames attribute. If the names for a particular dimension are `NULL`, names for this dimension are constructed in other ways. If `new.names` is a character vector, it is used for dimension names in the same way as argument names are used. Zero length ("") names are ignored.
`force.array`	(optional) If `FALSE`, rbind or cbind are called when possible, i.e., when the arguments are all vectors, and along is not 1, or when the arguments are vectors or matrices or data frames and along is 1 or 2. If rbind or cbind are used, they will preserve the data.frame classes (or any other class that r/cbind preserve). Otherwise, abind will convert objects to class array. Thus, to guarantee that an array

	object is returned, supply the argument `force.array=TRUE`. Note that the use of rbind or cbind introduces some subtle changes in the way default dimension names are constructed: see the examples below.
`make.names`	(optional) If `TRUE`, the last resort for dimnames for the along dimension will be the deparsed versions of anonymous arguments. This can result in cumbersome names when arguments are expressions. <p>The default is `FALSE`.
`use.anon.names`	(optional) `use.anon.names` is a deprecated synonym for `make.names`.
`use.first.dimnames`	(optional) When dimension names are present on more than one argument, should dimension names for the result be take from the first available (the default is to take them from the last available, which is the same behavior as `rbind` and `cbind`.)
`hier.names`	(optional) If `TRUE`, dimension names on the concatenated dimension will be composed of the argument name and the dimension names of the objects being bound. If a single list argument is supplied, then the names of the components serve as the argument names. `hier.names` can also have values `"before"` or `"after"`; these determine the order in which the argument name and the dimension name are put together (`TRUE` has the same effect as `"before"`).
`use.dnns`	(default `FALSE`) Use names on dimensions, e.g., so that `names(dimnames(x))` is non-empty. When there are multiple possible sources for names of dimnames, the value of `use.first.dimnames` determines the result.

Details

The dimensions of the supplied vectors or arrays do not need to be identical, e.g., arguments can be a mixture of vectors and matrices. `abind` coerces arguments by the addition of one dimension in order to make them consistent with other arguments and `along=`. The extra dimension is added in the place specified by `along=`.

The default action of abind is to concatenate on the last dimension, rather than increase the number of dimensions. For example, the result of calling abind with vectors is a longer vector (see first example below). This differs from the action of `rbind` and cbind which is to return a matrix when called with vectors. abind can be made to behave like cbind on vectors by specifying `along=2`, and like rbind by specifying `along=0`.

The dimnames of the returned object are pieced together from the dimnames of the arguments, and the names of the arguments. Names for each dimension are searched for in the following order: new.names, argument name, dimnames (or names) attribute of last argument, dimnames (or names) attribute of second last argument, etc. (Supplying the argument `use.first.dimnames=TRUE` changes this to cause `abind` to use dimnames or names from the first argument first. The default behavior is the same as for `rbind` and `cbind`: use dimnames from later arguments.) If some names are supplied for the along dimension (either as argument names or dimnames in arguments), names are constructed for anonymous arguments unless `use.anon.names=FALSE`.

Value

An array with a dim attribute calculated as follows.

Let `rMin=min(sapply(list(...), function(x) length(dim(x))))` and `rMax=max(sapply(list(...), function(x) length(dim(x))))` (where the length of the dimensions of a vector are taken to be 1). Then `rMax` should be equal to or one greater than `rMin`.

If `along` refers to an existing dimension, then the length of the dim attribute of the result is `rMax`. If `along` does not refer to an existing dimension, then `rMax` should equal `rMin` and the length of the dim attribute of the result will be `rMax+1`.

`rbind` or `cbind` are called to compute the result if (a) `force.array=FALSE`; and (b) the result will be a two-dimensional object.

Note

It would be nice to make `abind()` an S3 generic, but S3 generics cannot dispatch off anonymous arguments.

The ability of `abind()` to accept a single list argument removes much of the need for constructs like `do.call("abind", list.of.arrays)`. Instead, just do `abind(list.of.arrays)`. The direct construct is preferred because `do.call()` construct can sometimes consume more memory during evaluation.

Author(s)

Tony Plate <`tplate@acm.org`> and Richard Heiberger

Examples

```
# Five different ways of binding together two matrices
x <- matrix(1:12,3,4)
y <- x+100
dim(abind(x,y,along=0))     # binds on new dimension before first
dim(abind(x,y,along=1))     # binds on first dimension
dim(abind(x,y,along=1.5))
dim(abind(x,y,along=2))
dim(abind(x,y,along=3))
dim(abind(x,y,rev.along=1)) # binds on last dimension
dim(abind(x,y,rev.along=0)) # binds on new dimension after last

# Unlike cbind or rbind in that the default is to bind
# along the last dimension of the inputs, which for vectors
# means the result is a vector (because a vector is
# treated as an array with length(dim(x))==1).
abind(x=1:4,y=5:8)
# Like cbind
abind(x=1:4,y=5:8,along=2)
abind(x=1:4,matrix(5:20,nrow=4),along=2)
abind(1:4,matrix(5:20,nrow=4),along=2)
# Like rbind
abind(x=1:4,matrix(5:20,nrow=4),along=1)
abind(1:4,matrix(5:20,nrow=4),along=1)
# Create a 3-d array out of two matrices
abind(x=matrix(1:16,nrow=4),y=matrix(17:32,nrow=4),along=3)
# Use of hier.names
```

```
abind(x=cbind(a=1:3,b=4:6), y=cbind(a=7:9,b=10:12), hier.names=TRUE)
# Use a list argument
abind(list(x=x, y=x), along=3)
# Use lapply(..., get) to get the objects
an <- c('x','y')
names(an) <- an
abind(lapply(an, get), along=3)
```

`acorn` *Return a corner of an array object (like head)*

Description

Return a small corner of an array object, like `head()` or `tail()` but taking only a few slices on each dimension.

Usage

```
acorn(x, n=6, m=5, r=1, ...)
```

Arguments

`x`	An array (including a matrix or a data frame)
`n, m, r`	Numbers of items on each dimension. A negative number is interpreted as this many items at the end (like tail).
`...`	Further arguments specifying numbers of slices to return on each dimension.

Details

Like `head()` for multidimensional arrays, with two differences: (1) returns just a few items on each dimension, and (2) negative numbers are treated like `tail()`.

Value

An object like `x` with fewer elements on each dimension.

Author(s)

Tony Plate `<tplate@acm.org>`

Examples

```
x <- array(1:24,dim=c(4,3,2),dimnames=rev(list(letters[1:2],LETTERS[1:3],letters[23:26])))
acorn(x)
acorn(x, 3)
acorn(x, -3)
acorn(x, 3, -2)
```

`adrop` *Drop dimensions of an array object*

Description

Drop degenerate dimensions of an array object. Offers less automaticity and more control than the base `drop()` function. `adrop()` is a S3 generic, with one method, `adrop.default`, supplied in the `abind` package.

Usage

```
adrop(x, drop = TRUE, named.vector = TRUE, one.d.array = FALSE, ...)
```

Arguments

`x`	An array (including a matrix)
`drop`	A logical or numeric vector describing exactly which dimensions to drop. It is intended that this argument be supplied always. The default is very rarely useful (`drop=TRUE` means drop the first dimension of a 1-d array).
`named.vector`	Optional, defaults to `TRUE`. Controls whether a vector result has names derived from the `dimnames` of `x`.
`one.d.array`	Optional, defaults to `FALSE`. If `TRUE`, a one-dimensional array result will be an object with a `dim` attribute of length 1, and possibly a `dimnames` attribute. If `FALSE`, a one-dimensional result will be a vector object (named if `named.vector==TRUE`).
`...`	There are no additional arguments allowed for `adrop.default` but other methods may use them.

Details

Dimensions can only be dropped if their extent is one, i.e., dimension `i` of array `x` can be dropped only if `dim(x)[i]==1`. It is an error to request `adrop` to drop a dimension whose extent is not 1.

A 1-d array can be converted to a named vector by supplying `drop=NULL` (which means drop no dimensions, and return a 1-d array result as a named vector).

Value

If `x` is an object with a `dim` attribute (e.g., a matrix or `array`), then `adrop` returns an object like `x`, but with the requested extents of length one removed. Any accompanying `dimnames` attribute is adjusted and returned with `x`.

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

`abind`

Examples

```
x <- array(1:24,dim=c(2,3,4),dimnames=list(letters[1:2],LETTERS[1:3],letters[23:26]))
adrop(x[1,,,drop=FALSE],drop=1)
adrop(x[,1,,drop=FALSE],drop=2)
adrop(x[,,1,drop=FALSE],drop=3)
adrop(x[1,1,1,drop=FALSE],drop=1)
adrop(x[1,1,1,drop=FALSE],drop=2)
adrop(x[1,1,1,drop=FALSE],drop=3)
adrop(x[1,1,1,drop=FALSE],drop=1:2)
adrop(x[1,1,1,drop=FALSE],drop=1:2,one.d=TRUE)
adrop(x[1,1,1,drop=FALSE],drop=1:2,named=FALSE)
dim(adrop(x[1,1,1,drop=FALSE],drop=1:2,one.d=TRUE))
dimnames(adrop(x[1,1,1,drop=FALSE],drop=1:2,one.d=TRUE))
names(adrop(x[1,1,1,drop=FALSE],drop=1:2,one.d=TRUE))
dim(adrop(x[1,1,1,drop=FALSE],drop=1:2))
dimnames(adrop(x[1,1,1,drop=FALSE],drop=1:2))
names(adrop(x[1,1,1,drop=FALSE],drop=1:2))
```

`afill` *Fill an array with subarrays*

Description

Fill an array with subarrays. `afill` uses the dimension names in the value in determining how to fill the LHS, unlike standard array assignment, which ignores dimension names in the value. `afill()` is a S3 generic, with one method, `afill.default`, supplied in the `abind` package.

Usage

```
afill(x, ..., excess.ok = FALSE, local = TRUE) <- value
```

Arguments

`x`	An array to be changed
`...`	Arguments that specify indices for `x`. If `length(dim(value))` < `length(dim(x))`, then exactly `length(dim(x))` anonymous arguments must be supplied, with empty ones corresponding to dimensions of `x` that are supplied in `value`.
`excess.ok`	If there are elements of the dimensions of `value` that are not found in the corresponding dimensions of x, they will be discarded if `excess.ok=TRUE`.
`local`	Should the assignment be done in on a copy of x, and the result returned (normal behavior). If `local=FALSE` the assignment will be done directly on the actual argument supplied as `x`, which can be more space efficient.
`value`	A vector or array, with dimension names that match some dimensions of `x`

Details

The simplest use of `afill` is to fill a sub-matrix. Here is an example of this usage:

```
> (x <- matrix(0, ncol=3, nrow=4, dimnames=list(letters[1:4], LETTERS[24:26])))
  X Y Z
a 0 0 0
b 0 0 0
c 0 0 0
d 0 0 0
> (y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26])))
  Y Z
b 1 3
c 2 4
> afill(x) <- y
> x
  X Y Z
a 0 0 0
b 0 1 3
c 0 2 4
d 0 0 0
>
```

The above usage is equivalent (when x and y have appropriately matching dimnames) to

```
> x[match(rownames(y), rownames(x)), match(colnames(y), colnames(x))] <- y
```

A more complex usage of `afill` is to fill a sub-matrix in a slice of a higher-dimensional array. In this case, indices for `x` must be supplied as arguments to `afill`, with the dimensions corresponding to those of `value` being empty, e.g.:

```
> x <- array(0, dim=c(2,4,3), dimnames=list(LETTERS[1:2], letters[1:4], LETTERS[24:26]))
> y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26]))
> afill(x, 1, , ) <- y
> x[1,,]
  X Y Z
a 0 0 0
b 0 1 3
c 0 2 4
d 0 0 0
> x[2,,]
  X Y Z
a 0 0 0
b 0 0 0
c 0 0 0
d 0 0 0
>
```

The most complex usage of `afill` is to fill a sub-matrix in multiple slice of a higher-dimensional array. Again, indices for `x` must be supplied as arguments to `afill`, with the dimensions corresponding to those of `value` being empty. Indices in which all slices should be filled can be supplied as `TRUE`. E.g.:

```
> x <- array(0, dim=c(2,4,3), dimnames=list(LETTERS[1:2], letters[1:4], LETTERS[24:26]))
> y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26]))
> afill(x, TRUE, , ) <- y
> x[1,,]
  X Y Z
a 0 0 0
b 0 1 3
c 0 2 4
d 0 0 0
> x[2,,]
  X Y Z
a 0 0 0
b 0 1 3
c 0 2 4
d 0 0 0
>
```

In the above usage, `afill` takes care of replicating `value` in the appropriate fashion (which is not straghtforward in some cases).

Value

The object `x` is changed. The return value of the assignment is the parts of the object `x` that are changed. This is similar to how regular subscript-replacement behaves, e.g., the expression `x[2:3] <- 1:2` returns the vector `1:2`, not the entire object `x`. However, note that there can be differences

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

`Extract`

Examples

```
# fill a submatrix defined by the dimnames on y
(x <- matrix(0, ncol=3, nrow=4, dimnames=list(letters[1:4], LETTERS[24:26])))
(y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26])))
afill(x) <- y
x
all.equal(asub(x, dimnames(y)), y) # TRUE
# fill a slice in a higher dimensional array
x <- array(0, dim=c(2,4,3), dimnames=list(LETTERS[1:2], letters[1:4], LETTERS[24:26]))
y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26]))
afill(x, 1, , ) <- y
x[1,,]
x[2,,]
```

```
all.equal(asub(x, c(1,dimnames(y))), y) # TRUE
# fill multiple slices
x <- array(0, dim=c(2,4,3), dimnames=list(LETTERS[1:2], letters[1:4], LETTERS[24:26]))
y <- matrix(1:4, ncol=2, nrow=2, dimnames=list(letters[2:3], LETTERS[25:26]))
afill(x, TRUE, , ) <- y
x[1,,]
x[2,,]
all.equal(asub(x, c(1,dimnames(y))), y) # TRUE
all.equal(asub(x, c(2,dimnames(y))), y) # TRUE
```

`asub` *Arbitrary subsetting of array-like objects at specified indices*

Description

Subset array-like objects at specified indices. `asub()` is a S3 generic, with one method, `asub.default`, supplied in the `abind` package.

Usage

```
asub(x, idx, dims = seq(length.out = max(length(dim(x)), 1)), drop = NULL, ...)
```

Arguments

`x`	The object to index
`idx`	A list of indices (e.g., a list of a mixture of integer, character, and logical vectors, but can actually be anything). Can be just a vector in the case that `length(dims)==1`. `NULL` entries in the list will be treated as empty indices.
`dims`	The dimensions on which to index (a numeric or integer vector). The default is all of the dimensions.
`drop`	The 'drop' argument to index with (the default is to not supply a 'drop' argument
`...`	There are no additional arguments allowed for `asub.default` but other methods may use them.

Details

Constructs and evaluates an expression to do the requested indexing. E.g., for `x` with `length(dim(x))==4` the call `asub(x, list(c("a","b"), 3:5), 2:3)` will construct and evaluate the expression `x[, c("a","b"), 3:5, ]`, and the call `asub(x, 1, 2, drop=FALSE)` will construct and evaluate the expression `x[, 1, , , drop=FALSE]`.

`asub` checks that the elements of `dims` are in the range 1 to `length(dim(x))` (in the case that `x` is a vector, `length(x)` is used for `dim(x)`). Other than that, no checks are made on the suitability of components of `idx` as indices for `x`. If the components of `idx` have any out-of-range values or unsuitable types, this will be left to the subsetting method for `x` to catch.

Value

A subset of `x`, as returned by `x[...]`.

Author(s)

Tony Plate <`tplate@acm.org`>

References

~put references to the literature/web site here ~

See Also

`Extract`

Examples

```
x <- array(1:24,dim=c(2,3,4),dimnames=list(letters[1:2],LETTERS[1:3],letters[23:26]))
asub(x, 1, 1, drop=FALSE)
asub(x, list(1:2,3:4), c(1,3))
```

Index