Package 'mefa4'

October 14, 2016

Type Package

Title Multivariate Data Handling with S4 Classes and Sparse Matrices

Version 0.3-4

Date 2016-10-12

Author Peter Solymos

Maintainer Peter Solymos <solymos@ualberta.ca>

Description An S4 update of the 'mefa' package
using sparse matrices for enhanced efficiency.
Sparse array-like objects are supported via
lists of sparse matrices.

Depends R (>= 2.14.0), methods, Matrix, pbapply

Suggests mefa

License GPL-2

URL https://github.com/psolymos/mefa4

BugReports https://github.com/psolymos/mefa4/issues

LazyLoad yes

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2016-10-14 08:16:21

R topics documented:

`mefa4-package` *Multivariate Data Handling with S4 Classes and Sparse Matrices*

Description

An S4 update of the 'mefa' package using sparse matrices for enhanced efficiency.

Details

An S4 update of the 'mefa' package using sparse matrices for enhanced efficiency. Sparse array-like objects are supported via lists of sparse matrices.

Main functions: `Xtab`, `Mefa`.

Accessor and replacement functions: `xtab`, `samp`, `taxa`.

Methods: `mbind`, `groupSums`, `groupMeans`.

Coercion methods and virtual classes defined for cross compatibility with the `mefa` package. S4 object classes are described in `Mefa` help page.

The vignette `vignette("mefa4")` gives an overview of the package, gives a comparison of S3 and S4 object classes, and presents a performance review.

Author(s)

Peter Solymos

Maintainer: Peter Solymos <solymos@ualberta.ca>

References

Solymos P. (2008) mefa: an R package for handling and reporting count data. *Community Ecology* **9**, 125–127.

Solymos P. (2009) Processing ecological data in R with the mefa package. *Journal of Statistical Software* **29(8)**, 1–28. http://www.jstatsoft.org/v29/i08/

http://mefa.r-forge.r-project.org/

See Also

S3 classes: `mefa`

Examples

```
## Not run:
vignette("mefa4")

## End(Not run)
```

---

abmibirds *Raw Dataset of Bird Point Counts*

Description

A data set of bird point counts collected by the Alberta Biodiversity Monitoring Institute (ABMI, http://www.abmi.ca).

Usage

```
data(abmibirds)
```

Format

A data frame with 59341 observations on the following 21 variables.

`Rotation` a factor. Reference describing when data was collected at a broad level. Code definition: Prototype = 2003–2006, Rotation 1 = 2007–2012

`ABMI.Site` a numeric vector. Reference number given to each ABMI data collection site (1–1656).

`Year` a numeric vector. Collection year.

`Field.Date` a factor. Day, month, and year data was collected.

`Field.Crew.Members` a factor. Initials for the field technicians collecting the field data.

`Identification.Date` a factor. Day, month, and year data was analyzed by specialist.

`Identification.Analyst` a factor. Initials for the technicians/specialists identifying the specimens.

`Point.Count.Station` a numeric vector. Point count station where recording was made: 9 stations were located around each ABMI site (1–9).

`Wind.Conditions` a factor. Estimate of wind conditions on a scale of 0–5. 0 = no wind, 1 = calm, 2 = leaves rustling, 3 small branches moving, 4 = large branches moving, 5 = large branches moving and the tree is swaying

`Precipitation` a factor. Classification for precipitation conditions in 5 categories. Input value: Drizzle, Fog, Rain, Sleet, Snow, None

`Start.of.Point.Count` a factor. Time of day recording was started. Input value: 24 hour clock (hh:mm).

`End.of.Point.Count` a factor. Time of day recording was finished. Input value: 24 hour clock (hh:mm).

`Common.Name` a factor. Common name of bird species detected during point counts.

`Scientific.Name` a factor. Scientific name of bird species detected during point count.

`Unique.Taxonomic.Identification.Number` a factor. Globally unique identifier of bird species detected during point count. Unique taxonomic identifiers are generally taken from the International Taxonomic Information System (ITIS; http://www.itis.gov/).

`Taxonomic.Resolution` a factor. Resolution to which bird species was identified (e.g. Family, Genus, Species etc.).

`Time.First.Detected` a factor. Approximate time the bird analyst first detects a bird species from the recording; listed in 10-second intervals.

`Interval.1` a factor. First time interval of the 10-minute point count (0–200 seconds) when bird species are detected and identified.

`Interval.2` a factor. Middle time interval of the 10-minute point count (201–400 seconds) when bird species are detected or re-detected.

`Interval.3` a factor. Last time interval of the 10-minute point count (401–600 seconds) when bird species are detected or re-detected.

`Behaviour` a factor. Classification given to each species detection (if possible).

Details

Breeding birds were measured at nine point count stations. Point count stations were in a grid pattern with point count station no. 1 located at site-centre and the remaining stations located 300 m apart surrounding site centre. We conducted breeding bird surveys from one half hour before sunrise to 10:00 hrs.

We recorded vocalizations of birds for 10 minutes at each point count station using an omni-directional microphone (CZM microphone; River Forks Research Corp.) mounted at ear level on a professional tripod and connected to a mini hard drive recorder. We recorded birds on a Marantz PM D670 or PM D660 Solid State recorder at 320 kbps in .mp3 format. We calibrated the recorder volume to be in the mid ranges.

While conducting the 10 minute bird recordings, we scanned the areas surrounding the point count station for all birds (even those vocalizing), noting species, number of individuals (including flock sizes of birds flying overhead), and distance from the point count station, for all bird observations. We also noted factors that potentially bias bird recordings, such as wind speed and precipitation. Bird recordings were later analyzed by bird identification specialists in a laboratory setting.

If a bird point fell in open water, we established a new point if we were able to get within 100 m of the original point, recording distance and direction from that original point. If it was not possible to get within 100 m of the original point (i.e., <200 m from the last point), we conducted a 10 minute visual point count of the waterbody recording observations into the microphone. We may not have sampled certain points because they were inaccessible (e.g., a stream made access hazardous or impossible).

We analyzed bird recordings in a laboratory setting. We identified the species, time of first detection (within 10 second intervals), behaviour (e.g., singing, calling, or alarm-calling), and the time interval that individual birds were detected. We recognized 3 time intervals: Interval 1 (0–200 seconds), Interval 2 (201–400 seconds), and Interval 3 (401–600 seconds). Individual birds were detected in 1, 2, or 3 of the time intervals. We identified red squirrel (*Tamiasciurus hudsonicus*) vocalizations in addition to bird vocalizations. Bird recordings are randomly sampled and verified by other experts in bird identification to ensure accuracy.

Throughout ABMI raw data files, the following codes and definitions are applied.

None or 0: None or 0 is applied to any variable that was examined by field crews and found to be absent. None is used for text entries and 0 is used for numerical entries. For example, when field crews examine the canopy and find no Veteran trees in the canopy, this is recorded as None. When there is no slope at the survey site, slope is recorded as 0. The numeral 0 can also be used as a nominal code, for example, wind conditions can be recorded as 0.

VNA Variable Not Applicable: Some ABMI data is collected in a nested manner. For example Tree Species is a parent variable. This variable has a number of child variables that are used to describe the parent variable in more detail (e.g., condition, DBH, decay stage). When the parent variable is recorded as None, child variables are no longer applied and are recorded as VNA. VNA is also used when the protocol calls for a modified sampling procedure based on site conditions (e.g., surface substrate protocol variant for hydric site conditions). The use of VNA implies that users of the data should not expect that any data could be present.

DNC, Did Not Collect: DNC is used to describe variables that should have been collected but were not. There are a number of reasons that data might not have been collected (e.g., staff oversight, equipment failure, safety concerns, environmental conditions, or time constraints). Regardless of the reason data was not collected, if under ideal conditions it should have been, the record in the data entry file reads DNC. The use of DNC implies that users should expect the data to be present, though it is not.

PNA, Protocol Not Available: The ABMI's protocols were, and continue to be, implemented in a staged manner. As a result, the collection of many variables began in years subsequent to the start of the prototype or operational phases or where discontinued after a few years of trial. When a variable was not collected because the protocol had yet to be implemented by the ABMI (or was discontinued by the ABMI), the data entry record reads PNA. This is a global constraint to the data (i.e., a protocol was not implemented until 2006, therefore, previous years cannot have this variable). PNA is to be used to describe the lack of data collection for entire years.

SNI, Species Not Identified: In various fields related to species identification, SNI is used to indicate that the organism was not identified. Some possible reasons that identification was not possible include insufficient or deficient sample collected and lack of field time.

Source

RAW_T26BreedingBirds28621.csv, http://www.abmi.ca

References

Raw breeding bird data (2004–2006 inclusive) from the Alberta Biodiversity Monitoring Institute was used, in whole or part, to create this product. More information on the Institute can be found at: http://www.abmi.ca

Examples

```
data(abmibirds)
str(abmibirds)
```

find_max *Utility functions for factors and compositional data*

Description

Utility functions for factors and compositional data.

Usage

```
compare_sets(x, y)
find_max(x)
find_min(x)
reclass(x, map, all = FALSE)
redistribute(x, source, target = NULL)
```

Arguments

x, y	any type for compare_sets, matrix for find_max, find_min, and redistribute, a factor for reclass.
map	a reclassification matrix with 2 columns (1st: original levels, 2nd: output levels mapped to original levels).
all	logical, whether all levels from mapping matrix should be applied on the return object.
source	numeric or character, single column index for input matrix x.
target	numeric or character, column index or indices for input matrix x.

Value

A matrix compare_sets.

A data frame for find_max and find_min.

A reclassified factor for reclass.

A matrix for redistribute where the source column values are redistributed among the target columns proportionally.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

intersect, setdiff, union, relevel, reorder

Examples

```
## numeric vector
compare_sets(1:10, 8:15)
## factor with 'zombie' labels
compare_sets(factor(1:10, levels=1:10), factor(8:15, levels=1:15))

(mat <- matrix(rnorm(10*5), 10, 5))
(m <- find_max(mat))
## column indices
as.integer(m$index)
find_min(mat)

map <- cbind(c("a","b","c","d","e","f","g"),
             c("A","B","B","C","D","D","E"))
#x <- factor(sample(map[1:6,1], 100, replace=TRUE), levels=map[,1])
x <- as.factor(sample(map[1:6,1], 100, replace=TRUE))
x[2] <- NA
table(x, reclass(x, map, all = FALSE), useNA="always")
table(x, reclass(x, map, all = TRUE), useNA="always")

(mat2 <- exp(mat) / rowSums(exp(mat)))
(rmat2 <- redistribute(mat2, source = 1, target = 2:4))
colMeans(mat2)
colMeans(rmat2)
stopifnot(abs(sum(mat2) - sum(rmat2)) < 10^-6)
```

---

groupSums *Compute Summary Statistics of Data Subsets*

Description

Compute summary statistics (sums, means) of data subsets.

Usage

```
groupSums(object, ...)
## S4 method for signature 'matrix'
groupSums(object, MARGIN, by, na.rm = FALSE, ...)
## S4 method for signature 'sparseMatrix'
groupSums(object, MARGIN, by, na.rm = FALSE, ...)
## S4 method for signature 'Mefa'
groupSums(object, MARGIN, by, replace, na.rm = FALSE, ...)

groupMeans(object, ...)
## S4 method for signature 'matrix'
groupMeans(object, MARGIN, by, na.rm = FALSE, ...)
## S4 method for signature 'sparseMatrix'
```

```
groupMeans(object, MARGIN, by, na.rm = FALSE, ...)
## S4 method for signature 'Mefa'
groupMeans(object, MARGIN, by, replace, na.rm = FALSE, ...)
```

Arguments

object	an object.
MARGIN	numeric, 1 indicates rows are to be summed/averaged, 2 indicates columns are to be summed/averaged. c(1, 2) is not yet implemented, but can be calculated calling the function twice.
by	a vector of grouping elements corresponding to dimensions of object and MARGIN.
replace	a data frame to be used when applying the method on a "Mefa" object. The attribute table corresponding to MARGIN is dropped (NULL), replacement table can be specified via this argument.
na.rm	logical. Should missing values be removed? Sum is calculated by zeroing out NA values, mean is calculated as dividing the sum by the number of non-NA values when collapsing.
...	other argument, currently not implemented.

Details

The method sums/averages cells in a matrix. The functions behind these methods use sparse matrices, so calculations can be more efficient compared to using aggregate.

Value

An object similar to the input one.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

rowSums, rowMeans, colSums, colMeans

Standard aggregate in package **stats**

aggregate.mefa in package **mefa** for S3 "mefa" objects.

Examples

```
x <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),
    "zero.pseudo"), count = c(1,2,10,3,4,0))
samp <- data.frame(samples=levels(x$sample), var1=1:2)
taxa <- data.frame(specnames=levels(x$species), var2=c("b","a"))
rownames(samp) <- samp$samples
rownames(taxa) <- taxa$specnames
```

```
x2 <- Xtab(count ~ sample + species, x, cdrop=FALSE,rdrop=TRUE)
x5 <- Mefa(x2, samp, taxa, join="inner")

groupSums(as.matrix(x2), 1, c(1,1,2))
groupSums(as.matrix(x2), 2, c(1,1,2,2))
groupSums(x2, 1, c(1,1,2))
groupSums(x2, 2, c(1,1,2,2))
groupSums(x5, 1, c(1,1,2))
groupSums(x5, 2, c(1,1,2,2))

groupMeans(as.matrix(x2), 1, c(1,1,2))
groupMeans(as.matrix(x2), 2, c(1,1,2,2))
groupMeans(x2, 1, c(1,1,2))
groupMeans(x2, 2, c(1,1,2,2))
groupMeans(x5, 1, c(1,1,2))
groupMeans(x5, 2, c(1,1,2,2))
```

`mbind` *Combine R Objects by Rows and Columns*

Description

Combine R objects by rows and columns.

Usage

```
mbind(x, y, fill, ...)
mbind2(x, y, fill, ...)
```

Arguments

`x, y`	objects to combine, see Details.
`fill`	numeric value or `NA` (default) to fill up outer set (not part of union) of dimnames.
`...`	other argument, not used.

Details

`x` and `y` are combined in a left join manner, meaning that all the elements in `x` are retained, and only non-overlapping elements in `y` are used. Elements of the returning object that are not part of `x` and `y` (outer set) are filled up with `fill`.

If relational table in `x` is `NULL`, corresponding values from same table of `y` are used.

`mbind2` combines `x` and `y` so that inner set is calculated as sum of corresponding elements from `x` and `y` (unlike in `mbind` with a left join manner).

Value

An object similar to the input one.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

rbind, cbind

rBind, cBind for sparse matrices in **Matrix** package

Examples

```
x=matrix(1:4,2,2)
rownames(x) <- c("a","b")
colnames(x) <- c("A","B")
y=matrix(11:14,2,2)
rownames(y) <- c("b","c")
colnames(y) <- c("B","C")

sampx <- data.frame(x1=1:2, x2=2:1)
rownames(sampx) <- rownames(x)
sampy <- data.frame(x1=3:4, x3=10:11)
rownames(sampy) <- rownames(y)
taxay <- data.frame(x1=1:2, x2=2:1)
rownames(taxay) <- colnames(y)
taxax <- NULL

mbind(x,y)
mbind(as(x,"sparseMatrix"),as(y,"sparseMatrix"))
xy <- mbind(Mefa(x,sampx),Mefa(y,sampy,taxay))
unclass(xy)

mbind2(x,y)
mbind2(as(x,"sparseMatrix"),as(y,"sparseMatrix"))
xtab(xy) <- mbind2(x, y)
unclass(xy)
```

`Mefa` *'Mefa' Class*

Description

Creating an object of class "Mefa".

Usage

```
Mefa(xtab, samp, taxa, join = c("left", "inner"), drop = FALSE)
```

Arguments

xtab	a matrix or a sparse matrix.
samp	a data frame or NULL.
taxa	a data frame or NULL.
join	character, "left" (default) or "inner".
drop	logical, if unused levels in the data frames should be dropped.

Details

samp and taxa tables are matched with corresponding dimnames in xtab: rownames with samp, colnames with taxa. If join = "left", all rows and columns in xtab are retained, while missing items in the corresponding attribute tables are filled up with NAs. If join = "inner", only the intersection of corresponding names are retained.

The xtab slot is a sparse matrix (dgCMatrix). The input should be in class MefaMatrix that is a class union of matrix and sparseMatrix classes.

The samp and taxa slots take data frame or NULL, which two form the MefaDataFrame class union.

The virtual classes mefa and stcs are defined for seamless coercion between S3 and S4 classes.

Value

An S4 object of class "Mefa" with 4 slots: xtab, samp, taxa, join.

Note

If xtab has no dimnames, matching it up with the attribute tables can be problematic.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

Creating crosstabulations: Xtab, xtabs in package **stats**, xtabs in package **stats**

"mefa" S3 class: mefa in **mefa** package.

Accessing and replacing slots: xtab, samp, taxa.

Examples

```
x <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),  "zero.pseudo"),
    count = c(1,2,10,3,4,0))
samp <- data.frame(samples=levels(x$sample), var1=1:2)
taxa <- data.frame(specnames=levels(x$species), var2=c("b","a"))
rownames(samp) <- samp$samples
rownames(taxa) <- taxa$specnames
```

```
## Xtab class, counts by repetitions in RHS
(x0 <- Xtab(~ sample + species, x))

## counts by LHS and repetitions in RHS
(x1 <- Xtab(count ~ sample + species, x))

## drop all empty rows
(x2 <- Xtab(count ~ sample + species, x, cdrop=FALSE,rdrop=TRUE))

## drop all empty columns
Xtab(count ~ sample + species, x, cdrop=TRUE,rdrop=FALSE)

## drop specific columns by placeholder
Xtab(count ~ sample + species, x, cdrop="zero.pseudo")

## Mefa class, standard
(x3 <- Mefa(x1, samp, taxa))
unclass(x3)
x3@xtab
x3@samp
x3@taxa
x3@join

## effects of left join, NULL taxa slot, xtab is (not sparse) matrix
(x4 <- Mefa(as.matrix(x1), samp[1:2,]))
unclass(x4)

## effects of inner join (intersect)
(x5 <- Mefa(x2, samp, taxa, join="inner"))
unclass(x5)
unclass(Mefa(x1, samp[1:2,], join="inner"))

## xtab only Mefa
(x6 <- Mefa(x1))

## creating new Mefa object without Mefa()
new("Mefa", xtab=x1, samp=samp, taxa=taxa,join="left")

## dim and dimnames
dim(x5)
dimnames(x5)
dn <- list(paste("S", 1:3, sep=""), paste("SPP", 1:4, sep=""))
dimnames(x5) <- dn
unclass(x5)
dimnames(x5)[[1]] <- paste("S", 1:3, sep="_")
unclass(x5)
dimnames(x5)[[2]] <- paste("SPP", 1:4, sep="_")
unclass(x5)

## transpose
x5
t(x5)
unclass(x5)
```

```
unclass(t(x5))

## 0 and 1 row/col Mefa object
x3[c(FALSE,FALSE,FALSE,FALSE),c(FALSE,FALSE,FALSE,FALSE)]
x3[c(TRUE,FALSE,FALSE,FALSE),c(FALSE,FALSE,FALSE,FALSE)]
x3[c(FALSE,FALSE,FALSE,FALSE),c(TRUE,FALSE,FALSE,FALSE)]
x3[c(TRUE,FALSE,FALSE,FALSE),c(TRUE,FALSE,FALSE,FALSE)]

## stack
stack(x3)
```

`Melt` *Melting Matrices*

Description

The function reverses the side effects of cross tabulation.

Usage

```
Melt(x)
```

Arguments

`x`	A matrix, or sparse matrix object, a list of sparse matrices with identical dimnames, a 'mefa' or 'Mefa' object.

Value

A data frame with columns corresponding to `rows`, `cols`, possibly `segm` (names of the list if `x` was a list of sparse matrices), and `value`. `value` is numeric, other columns are factors.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

`stack` in **utils**, and `melt` in **mefa** package.

Examples

```
xx <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),  "zero.pseudo"),
    count = c(1,2,10,3,4,0),
    segment = letters[c(6,13,6,13,6,6)])
xx
xx0 <- Xtab(count ~ sample + species, xx)
```

```
xx1 <- Xtab(count ~ sample + species + segment, xx)
(M1 <- Melt(xx0))
Melt(as.matrix(xx0))
(M2 <- Melt(xx1))
stopifnot(identical(Xtab(value ~ rows + cols, M1), xx0))
stopifnot(identical(Xtab(value ~ rows + cols + segm, M2), xx1))
```

---

nameAlnum *Utility functions, mostly for character manipulation*

Description

Utility functions, mostly for character manipulation.

Usage

```
pasteDate(..., sep = " ", collapse = NULL, sep.date = sep)
paste0date(..., collapse = NULL)
nameAlnum(x, capitalize=c("asis", "first", "none", "all", "mixed"),
    collapse=" ")
normalizeNames(x, pattern = list(" "), replacement = list("_"),
    alnum = FALSE, ...)
```

Arguments

x	caharacter.
...	one or more R objects, to be converted to character vectors. For normalizeNames it takes arguments passed to nameAlnum when alnum = TRUE.
sep	a character string to separate the terms.
collapse	an optional character string to separate the results. For nameAlnum it is the separator between the words in the output.
sep.date	a character string to separate the terms from the data itself.
capitalize	character, which letter of each words should be capitalized. "mixed" capitalizes the first letter and case is unchanged for the rest (CamelCase). "first" capitalizes first letter and uses lower case for the rest. Other options are self explanatory.
pattern	a list of character vectors that are replaced. Must match argument replacement.
replacement	a list of character vectors that are the replacements for pattern. Must match argument pattern.
alnum	logical, if nameAlnum should be applied after replacement.

Value

Character vector with desired changes.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

paste, paste0, strsplit, gsub

Examples

```
paste0(pasteDate("file", "name", sep="-", sep.date="_"), ".csv")
paste0(paste0date("file", "name", sep.date="_"), ".csv")

data(abmibirds)
x <- data.frame(Name=levels(abmibirds[["Common.Name"]]),
    NameAlnum=nameAlnum(levels(abmibirds[["Common.Name"]])))
x[grep("'", x$Name),]

z <- data.frame(Name=levels(abmibirds[["Common.Name"]]),
    NameNormalized=normalizeNames(levels(abmibirds[["Common.Name"]]),
        pattern=list("'", "-", " "), replacement=list("", "_", "_")))
z[grepl("'", z$Name) & grepl("-", z$Name),]
```

---

nonDuplicated *Non Duplicated Rows in Data Frame*

Description

Subset a data frame using non duplicated elements in a vector.

Usage

```
nonDuplicated(x, y, change.rownames = FALSE, na.rm = FALSE)
```

Arguments

x	a data frame.
y	a vector. It can be a name of a column in x without quotes.
change.rownames	if original rownames of x are to be replaced by unique non duplicated values of y.
na.rm	logical. If rows should be removed where y is NA. This is to be applied if values of y are used as rownames by setting change.rownames = TRUE

Details

This function is handy to keep only one set of duplicated data that is common in long formatted database files.

Value

A data frame.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

duplicated

Examples

```
data(abmibirds)
x <- nonDuplicated(abmibirds, abmibirds$ABMI.Site, TRUE)
## or equivalently
#x <- nonDuplicated(abmibirds, ABMI.Site, TRUE)
dim(abmibirds)
dim(x)
length(unique(abmibirds$ABMI.Site))
```

---

r2rmd *Parse R source file and return R markdown*

Description

Parses an R source file and returns an R markdown document that can be turned into a human readable documentation of what the source file does.

Usage

```
r2rmd(file, out=paste(file, "md", sep=""), header=TRUE, extra)
```

Arguments

file	a file name or connection (see readLines.
out	an output file name passed to writeLines, or NULL (no file written).
header	logical, if a yaml header (enclosed between tribple dashes, ---) is to be parsed.
extra	character, optional string that is placed into the code chunk openings.

Details

Leading double hashes ## treated as non-code. Leading # followed by other than # is code comment. Leading # after whitespace is code comment. A newline is code when preceded and followed by code.

The leading double hash ## is trimmed for comment lines. R markdown chunk start/end stuff is added for code chunks. The argument extra adds chunk arguments, e.g. extra=', eval=FALSE' etc. See R markdown website at http://rmarkdown.rstudio.com/

Value

Returns a character vector invisibly, and writes a file as a side effects unless `out=NULL` in which case no file is written.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

http://rmarkdown.rstudio.com/

Examples

```
## Not run:
(r2rmd(system.file("r2rmd_example.R", package="mefa4"),
    out=NULL, extra=", eval=FALSE"))

## End(Not run)
```

`samp` *Accessing and Replacing Parts of 'Mefa' Objects*

Description

Methods to access and replace parts (elements, slots) of `"Mefa"` objects.

Usage

```
xtab(x)
"xtab<-"(x, value)

samp(x)
"samp<-"(x, value)

taxa(x)
"taxa<-"(x, value)
```

Arguments

`x`	an object of S4 class `"Mefa"`.
`value`	replacement value.

Details

The `[` method ensures that the `xtab` sparse matrix part and the corresponding attribute tables are subsetted correctly.

Validity check is performed when replacing slots of an object.

Value

An object of S4 class "Mefa".

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

Mefa

Examples

```
x <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),  "zero.pseudo"),
    count = c(1,2,10,3,4,0))
samp <- data.frame(samples=levels(x$sample), var1=1:2)
taxa <- data.frame(specnames=levels(x$species), var2=c("b","a"))
rownames(samp) <- samp$samples
rownames(taxa) <- taxa$specnames
x1 <- Xtab(count ~ sample + species, x)
x3 <- Mefa(x1, samp, taxa)

## accessing the xtab slot
xtab(x3)
## replacing the slot value
x1[3,1] <- 999
xtab(x3) <- x1
xtab(x3)

## accessing and replacing the samp slot
samp(x3)
samp(x3) <- NULL
samp(x3)
samp(x3) <- samp[1:3,]
samp(x3)

## accessing and replacing the taxa slot
taxa(x3)
taxa(x3) <- NULL
taxa(x3)
taxa(x3) <- taxa[1:3,]
taxa(x3)

## subsetting
unclass(x3[3:2, 1:2])
unclass(x3[3:2,])
unclass(x3[,1:2])
```

Xtab *Sparse Cross Tabulation*

Description

Create a contingency table from cross-classifying factors, usually contained in a data frame, using a formula interface.

Usage

```
Xtab(formula = ~., data = parent.frame(), rdrop, cdrop,
subset, na.action, exclude = c(NA, NaN), drop.unused.levels = FALSE)
```

Arguments

formula	a formula object with the cross-classifying variables (separated by +) on the right hand side (or an object which can be coerced to a formula). Interactions are not allowed. On the left hand side, one may optionally give a vector or a matrix of counts; in the latter case, the columns are interpreted as corresponding to the levels of a variable. This is useful if the data have already been tabulated, see the examples below.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
rdrop, cdrop	logical (should zero marginal rows/columns be removed after cross tabulation), character or numeric (what rows/columns should be removed).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs.
exclude	a vector of values to be excluded when forming the set of levels of the classifying factors.
drop.unused.levels	a logical indicating whether to drop unused levels in the classifying factors. If this is FALSE and there are unused levels, the table will contain zero marginals, and a subsequent chi-squared test for independence of the factors will not work.

Details

The function creates two- or three-way cross tabulation. Only works for two or three factors.

If a left hand side is given in formula, its entries are simply summed over the cells corresponding to the right hand side; this also works if the left hand side does not give counts.

Value

A sparse numeric matrix inheriting from sparseMatrix, specifically an object of S4 class dgCMatrix.

For three factors, a list of sparse matrices.

Author(s)

This function is a slight modification of the `xtabs` function in the **stats** package.

Modified by Peter Solymos <solymos@ualberta.ca>

See Also

See also `xtabs` in **stats** package.

`"mefa"` S3 class: `mefa` in **mefa** package.

Examples

```
x <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),  "zero.pseudo"),
    count = c(1,2,10,3,4,0))
x
## Xtab class, counts by repetitions in RHS
(x0 <- Xtab(~ sample + species, x))
## counts by LHS and repetitions in RHS
(x1 <- Xtab(count ~ sample + species, x))
## drop all empty rows
(x2 <- Xtab(count ~ sample + species, x, cdrop=FALSE,rdrop=TRUE))
## drop all empty columns
Xtab(count ~ sample + species, x, cdrop=TRUE,rdrop=FALSE)
## drop specific columns by placeholder
Xtab(count ~ sample + species, x, cdrop="zero.pseudo")

## 2 and 3 way crosstabs
xx <- data.frame(
    sample = paste("Sample", c(1,1,2,2,3,4), sep="."),
    species = c(paste("Species", c(1,1,1,2,3), sep="."),  "zero.pseudo"),
    count = c(1,2,10,3,4,0),
    segment = letters[c(6,13,6,13,6,6)])
xx
Xtab(count ~ sample + species, xx)
Xtab(count ~ sample + species + segment, xx)
```

%notin% *Negated Value Matching*

Description

`%notin%` is the negation of `%in%`, which returns a logical vector indicating if there is a non-match or not for its left operand.

Usage

```
x %notin% table
```

Arguments

x	vector or NULL: the values to be matched.
table	vector or NULL: the values to be matched against.

Value

A logical vector, indicating if a non-match was located for each element of x: thus the values are TRUE or FALSE and never NA.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

All the opposite of what is written for %in%.

Examples

```
1:10 %notin% c(1,3,5,9)
sstr <- c("c","ab","B","bba","c",NA,"@","bla","a","Ba","%")
sstr[sstr %notin% c(letters, LETTERS)]
```

Index