Package 'matsbyname'

October 13, 2022

Type Package

Title An Implementation of Matrix Mathematics

Version 0.5.0

Date 2022-04-01

Maintainer Matthew Heun <matthew.heun@me.com>

Description An implementation of matrix mathematics wherein operations are performed ``by name."

License MIT + file LICENSE

Language en-US

Encoding UTF-8

RoxygenNote 7.1.2

Imports assertthat, dplyr, lifecycle, magrittr, purrr, RCLabels, rlang, tibble

Suggests covr, knitr, matsindf, rmarkdown, testthat, tidyr

VignetteBuilder knitr

URL https://github.com/MatthewHeun/matsbyname

BugReports https://github.com/MatthewHeun/matsbyname/issues

NeedsCompilation no

Author Matthew Heun [aut, cre] (<https://orcid.org/0000-0002-7438-214X>)

Repository CRAN

Date/Publication 2022-04-01 19:40:02 UTC

R topics documented:

`abs_byname` *Absolute value of matrix elements*

Description

Absolute value of matrix elements

Usage

```
abs_byname(a)
```

Arguments

`a`	a matrix or list of matrices

Value

a with each element replaced by its absolute value

Examples

```
abs_byname(1)
abs_byname(-1)
m <- matrix(c(-10,1,1,100), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
abs_byname(m)
```

`aggregate_byname` *Aggregate rows and columns in a matrix*

Description

Rows (`margin = 1`), columns (`margin = 2`), or both (`margin = c(1, 2)`, the default) are aggregated according to `aggregation_map`.

Usage

```
aggregate_byname(
  a,
  aggregation_map = NULL,
  margin = c(1, 2),
  pattern_type = "exact"
)
```

Arguments

`a`	A matrix or list of matrices whose rows or columns are to be aggregated.
`aggregation_map`	A named list of rows or columns to be aggregated (or `NULL`). See `details`.
`margin`	1, 2, or `c(1, 2)` for row aggregation, column aggregation, or both.
`pattern_type`	See `RCLabels::make_or_pattern()`.

Details

When `aggregation_map` is `NULL` (the default), rows (or columns or both) of same name are aggregated together.

If `aggregation_map` is not `NULL`, it must be a named list. The name of each `aggregation_map` item is the name of a row or column in output that will contain the specified aggregation. The value of each item in `aggregation_map` must be a vector of names of rows or columns in `a`. The names in the value are aggregated and inserted into the output with the name of the value. For example `aggregation_map = list(new_row = c("r1", "r2"))` will aggregate rows "r1" and "r2", delete

rows "r1" and "r2", and insert a new row whose name is "new_row" and whose value is the sum of rows "r1" and "r2'.

The values in the `aggregation_map` are interpreted as regular expressions, and they are escaped using `Hmisc::escapeRegex()` prior to use.

Note that aggregation on one margin only will sort only the aggregated margin, because the other margin is not guaranteed to have unique names.

Value

A version of a with aggregated rows and/or columns

Examples

```
library(dplyr)
library(tibble)
m <- matrix(1:9, byrow = TRUE, nrow = 3,
            dimnames = list(c("r2", "r1", "r1"), c("c2", "c1", "c1"))) %>%
  setrowtype("rows") %>% setcoltype("cols")
# Aggregate all rows by establishing an aggregation map (`am`)
am <- list(new_row = c("r1", "r2"))
aggregate_byname(m, aggregation_map = am, margin = 1)
# aggregate_byname() also works with lists and in data frames
m1 <- matrix(42, nrow = 1, dimnames = list(c("r1"), c("c1")))
m2 <- matrix(1:4, byrow = TRUE, nrow = 2,
             dimnames = list(c("a", "a"), c("a", "a")))
m3 <- matrix(1:9, byrow = TRUE, nrow = 3,
             dimnames = list(c("r2", "r1", "r1"), c("c2", "c1", "c1")))
DF <- tibble(m = list(m1, m1, m1, m2, m2, m2, m3, m3, m3),
             margin = list(1, 2, c(1,2), 1, 2, c(1, 2), 1, 2, c(1, 2))) %>%
  mutate(
    aggregated = aggregate_byname(m, margin = margin),
  )
m1
DF$aggregated[[1]] # by rows
DF$aggregated[[2]] # by cols
DF$aggregated[[3]] # by rows and cols
m2
DF$aggregated[[4]] # by rows
DF$aggregated[[5]] # by cols
DF$aggregated[[6]] # by rows and cols
m3
DF$aggregated[[7]] # by rows
DF$aggregated[[8]] # by cols
DF$aggregated[[9]] # by rows and cols
```

`aggregate_pieces_byname` *Aggregate a matrix by pieces of row and/or column names*

Description

Aggregate a matrix (or list of matrices or a column in a `matsindf` data frame) by pieces of the row and column names.

Usage

```
aggregate_pieces_byname(
  a,
  piece,
  margin = c(1, 2),
  notation,
  prepositions = RCLabels::prepositions,
  aggregation_map = NULL,
  pattern_type = "exact"
)
```

Arguments

`a`	A matrix or list of matrices
`piece`	See `rename_to_piece_byname()`.
`margin`	See `rename_to_piece_byname()`.
`notation`	See `rename_to_piece_byname()`.
`prepositions`	See `rename_to_piece_byname()`.
`aggregation_map`	See `aggregate_byname()`.
`pattern_type`	See `RCLabels::make_or_pattern()`.

Details

This is a convenience function that bundles two others for common use cases: `rename_to_piece_byname()` followed by `aggregate_byname()`.

`aggregation_map` should aggregate according to pieces, not according to the full, original row and/or column names.

Value

A version of `a` with rows and/or columns aggregated according to `aggregation_map`.

Examples

```
a <- matrix(c(1, 2, 3,
              4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE,
            dimnames = list(c("a [from b]", "c [from d]"),
                            c("e [from f]", "g [from h]", "i [from j]")))
a %>%
  aggregate_pieces_byname(piece = "suff",
                          notation = RCLabels::from_notation,
                          aggregation_map = list(rows = c("b", "d"),
```

```
                                                  cols = c("h", "j")))
m <- matrix(c(1, 0, 0,
              0, 1, 1,
              0, 1, 1), nrow = 3, ncol = 3, byrow = TRUE,
            dimnames = list(c("Gasoline [from Oil refineries]",
                              "Electricity [from Main activity producer electricity plants]",
                                  "Electricity [from Hydro]"),
                                c("Automobiles", "LED lamps", "CFL lamps"))) %>%
  setrowtype("Product") %>% setcoltype("Industry")
mT <- transpose_byname(m)
# Aggregate the "Electricity" rows.
aggregate_pieces_byname(m, piece = "noun", margin = "Product",
                        notation = RCLabels::bracket_notation)
# Also works in a list.
aggregate_pieces_byname(a = list(m, mT), piece = "noun",
                        margin = "Product",
                        notation = RCLabels::bracket_notation)
# Use an aggregation map
aggregate_pieces_byname(a = list(m, mT), piece = "noun",
                        margin = "Product",
                       aggregation_map = list(list(final = c("Electricity", "Gasoline"))),
                        notation = RCLabels::bracket_notation)
# Also works in a data frame.
df <- tibble::tibble(m = list(m, mT),
                     pce = "noun",
                     mgn = "Product",
                     agg_map = list(list(final = c("Electricity", "Gasoline"))),
                     notn = list(RCLabels::bracket_notation)) %>%
  dplyr::mutate(
    agg = aggregate_pieces_byname(a = m, piece = pce, margin = mgn,
                                  aggregation_map = agg_map,
                                  notation = notn)
  )
df$agg
```

`aggregate_to_pref_suff_byname` *Aggregate a matrix to prefixes or suffixes of row and/or column names*

Description

[Superseded] Row and column names are often constructed in the form `prefix_start prefix prefix_end suffix_start suffix suffix_end` and described by a notation vector. (See `notation_vec()`.) This function performs aggregation by prefix or suffix according to a notation vector..

Usage

```
aggregate_to_pref_suff_byname(
  a,
  aggregation_map = NULL,
```

```
  keep,
  margin = c(1, 2),
  notation,
  pattern_type = "exact"
)
```

Arguments

a	a matrix of list of matrices to be aggregated by prefix or suffix
aggregation_map	See aggregate_byname().
keep	See rename_to_pref_suff_byname()
margin	the dimension over which aggregation is to be performed; 1 for rows, 2 for columns, or c(1, 2) for both.
notation	See notation_vec().
pattern_type	See aggregate_byname().

Details

This function is a convenience function, as it bundles sequential calls to two helper functions, rename_to_pref_suff_byname() and aggregate_byname(). All arguments are passed to the helper functions.

Value

An aggregated version of a.

Examples

```
# This function is superseded.
# Instead, use `aggregate_pieces_byname()`.
# For example:
m <- matrix((1:9), byrow = TRUE, nrow = 3,
        dimnames = list(c("r1 -> b", "r2 -> b", "r3 -> a"), c("c1 -> z", "c2 -> y", "c3 -> y")))
m
aggregate_pieces_byname(m, piece = "pref", notation = RCLabels::arrow_notation)
aggregate_pieces_byname(m, piece = "suff", notation = RCLabels::arrow_notation)

# Original examples:
# Aggregation by prefixes does nothing more than rename, because all prefixes are different.
# Doing renaming like this (without also aggregating) is potentially dangerous, because
# some rows and some columns could end up with same names.
aggregate_to_pref_suff_byname(m, keep = "pref", notation = RCLabels::arrow_notation)
# Aggregation by suffix reduces the number of rows and columns,
# because there are same suffixes in both rows and columns
aggregate_to_pref_suff_byname(m, keep = "suff", notation = RCLabels::arrow_notation)
```

aggregation_map_helpers *Aggregation map conversions*

Description

Aggregation is a many-to-few operation where specifics are summed to comprise broader categories. Examples include "John", "Paul", "George", and "Ringo" aggregated to "Beatles"; and "Mick", "Keith", "Ronnie", "Bill", and "Charlie" aggregated to "Stones". An aggregation map is a named list that describes the aggregation to be performed. An aggregation map for the examples above is `list(Beatles = c("John", "Paul", "George", "Ringo"), Stones = c("Mick", "Keith", "Ronnie", "Bill", "Charlie"))` Aggregation maps can be generated from many shapes of data. These functions assist with translating from different data shapes to aggregation maps.

Usage

```
agg_table_to_agg_map(.df, few_colname, many_colname)

agg_map_to_agg_table(aggregation_map, few_colname, many_colname)
```

Arguments

`.df`	A data frame from which an aggregation map is to be extracted.
`few_colname`	The string name of a column in a data frame that corresponds to the "few" aggregated categories.
`many_colname`	The string name of a column in a data frame that corresponds to the "many" specific items that will be aggregated.
`aggregation_map`	An aggregation map to be converted to a data frame.

Value

For `agg_table_to_agg_map()`, an aggregation map. For `agg_map_to_agg_table()`, a `data.frame`, probably at `tibble`.

Examples

```
bands <- tibble::tribble(~band, ~members,
                         "The Beatles", "John",
                         "The Beatles", "Paul",
                         "The Beatles", "George",
                         "The Beatles", "Ringo",
                         # Rejects duplicates and NA
                         "The Beatles", "Ringo",
                         "The Beatles", NA,
                         "Rolling Stones", "Mick",
                         "Rolling Stones", "Keith",
```

```
                          "Rolling Stones", "Ronnie",
                          "Rolling Stones", "Bill",
                          "Rolling Stones", "Charlie")
agg_map <- agg_table_to_agg_map(bands,
                                few_colname = "band",
                                many_colname = "members")
agg_map
agg_map_to_agg_table(agg_map, few_colname = "bands", many_colname = "members")
```

`all_byname` *Are all matrix elements* `TRUE`*?*

Description

Tells whether all elements in matrix `a` are true.

Usage

```
all_byname(a)
```

Arguments

`a`	a matrix or list of matrices

Details

`a` can be a matrix or a list of matrices.

Value

`TRUE` if all elements of `a` are `TRUE`, `FALSE` otherwise

Examples

```
all_byname(matrix(rep(TRUE, times = 4), nrow = 2, ncol = 2))
all_byname(matrix(c(TRUE, FALSE), nrow = 2, ncol = 1))
```

and_byname *And "by name"*

Description

Operands should be logical, although numerical operands are accepted. Numerical operands are interpreted as FALSE when 0 and TRUE for any other number.

Usage

```
and_byname(..., .summarise = FALSE)
```

Arguments

...	Operands to the logical and function.
.summarise	Tells whether the operation should be accomplished across lists (FALSE) or down lists (TRUE).

Value

Logical and applied to the operands.

Examples

```
and_byname(TRUE)
and_byname(FALSE)
and_byname(list(TRUE, FALSE), list(TRUE, TRUE), list(TRUE, TRUE), list(TRUE, TRUE))
m1 <- matrix(c(TRUE, TRUE, TRUE, FALSE), nrow = 2, ncol = 2,
  dimnames = list(c("r1", "r2"), c("c1", "c2")))
m2 <- matrix(c(TRUE, FALSE, TRUE, TRUE), nrow = 2, ncol = 2,
  dimnames = list(c("r1", "r2"), c("c1", "c2")))
and_byname(m1, m1)
and_byname(m1, m2)
and_byname(list(m1, m1), list(m1, m1), list(m2, m2))
and_byname(list(m1, m1), list(m1, m1), list(m2, m2), .summarise = TRUE)
```

any_byname *Are any matrix elements* TRUE*?*

Description

Tells whether any elements in matrix a are true.

Usage

```
any_byname(a)
```

Arguments

a	a matrix or list of matrices

Details

a can be a matrix or a list of matrices.

Value

TRUE if any elements of a are TRUE, FALSE otherwise

Examples

```
any_byname(matrix(c(TRUE, FALSE), nrow = 2, ncol = 1))
any_byname(matrix(rep(FALSE, times = 4), nrow = 2, ncol = 2))
```

---

binaryapply_byname *Apply a binary function "by name"*

Description

If either a or b is missing or NULL, 0 is passed to FUN in its place. Note that if either a and b are lists, elements must be named the same. The names of list elements of a are applied to the output.

Usage

```
binaryapply_byname(
  FUN,
  a,
  b,
  .FUNdots = NULL,
  match_type = c("all", "matmult", "none"),
  set_rowcoltypes = TRUE,
  .organize = TRUE
)
```

Arguments

FUN	a binary function to be applied "by name" to a and b.
a	the first operand for FUN.
b	the second operand for FUN.
.FUNdots	a list of additional named arguments passed to FUN.
match_type	one of "all", "matmult", or "none". When both a and b are matrices, "all" (the default) indicates that rowtypes of a must match rowtypes of b and coltypes of a must match coltypes of b. If "matmult", coltypes of a must match rowtypes of b. If "none", neither coltypes nor rowtypes are checked.

set_rowcoltypes
: tells whether to apply row and column types from a and b to the output. Set TRUE (the default) to apply row and column types to the output. Set FALSE, to *not* apply row and column types to the output.

.organize
: a boolean that tells whether or not to automatically complete a and b relative to each other and sort the rows and columns of the completed matrices. Normally, this should be TRUE (the default). However, if FUN takes over this responsibility, set to FALSE.

Value

the result of applying FUN "by name" to a and b.

Examples

```
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>% setcoltype("Industries")
Y <- matrix(1:4, ncol = 2, dimnames = list(rev(productnames), rev(industrynames))) %>%
  setrowtype("Products") %>% setcoltype("Industries")
sum_byname(U, Y)
binaryapply_byname(`+`, U, Y)
```

---

clean_byname *Clean (delete) rows or columns of matrices that contain exclusively* clean_value

---

Description

Cleaning is performed when all entries in a row or column or both, depending on the value of margin are within +/- tol of clean_value. Internally, values are deemed within +/- of tol when abs(x - clean_value) <= tol.

Usage

```
clean_byname(a, margin = c(1, 2), clean_value = 0, tol = 0)
```

Arguments

a
: the matrix to be cleaned

margin
: the dimension over which cleaning should occur, 1 for rows, 2 for columns, or c(1, 2) for both rows and columns. Default is c(1, 2).

clean_value
: the undesirable value. Default is 0.

tol
: the tolerance with which any value is deemed equal to clean_value. Default is 0.

  When a row (when margin = 1) or a column (when margin = 2) contains exclusively clean_value (within tol), the row or column is deleted from the matrix.

Details

If there is concern about machine precision, you might want to call this function with `tol = .Machine$double.eps`.

Value

a "cleaned" matrix, expunged of rows or columns that contain exclusively `clean_value`.

Examples

```
m <- matrix(c(-20, 1, -20, 2), nrow = 2, dimnames = list(c("r1", "r2"), c("c1", "c2")))
m
m %>% clean_byname(margin = 1, clean_value = -20) # Eliminates -20, -20 row
# Nothing cleaned, because no columns contain all 0's (the default clean_value).
m %>% clean_byname(margin = 2)
# Also works with lists
list(m, m) %>% clean_byname(margin = 1, clean_value = -20)
# Also works with data frames
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
DF %>% clean_byname(margin = 1, clean_value = -20)
m2 <- matrix(c(-20, -20, 0, -20, -20, 0, -20, -20, -20), nrow = 3,
             dimnames = list(c("r1", "r2", "r3"), c("c1", "c2", "c3")) )
m2
clean_byname(m2, margin = c(1,2), clean_value = -20)
DF2 <- data.frame(m2 = I(list()))
DF2[[1, "m2"]] <- m2
DF2[[2, "m2"]] <- m2
DF2 %>% clean_byname(margin = c(1, 2), clean_value = -20)
```

---

colprods_byname *Column products, sorted by name*

Description

Calculates column products (the product of all elements in a column) for a matrix. An optional `rowname` for the resulting row vector can be supplied. If `rowname` is `NULL` or `NA` (the default), the row name is set to the row type as given by `rowtype(a)`.

Usage

```
colprods_byname(a, rowname = NA)
```

Arguments

`a`	a matrix or data frame from which column products are desired.
`rowname`	name of the output row containing column products.

Value

a row vector of type `matrix` containing the column products of `a`.

Examples

```
library(dplyr)
M <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 3:1))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
colprods_byname(M)
colprods_byname(M, rowname = "E.ktoe")
M %>% colprods_byname %>% rowprods_byname
# This also works with lists
colprods_byname(list(M, M))
colprods_byname(list(M, M), rowname = "E.ktoe")
colprods_byname(list(M, M), rowname = NA)
colprods_byname(list(M, M), rowname = NULL)
DF <- data.frame(M = I(list()))
DF[[1,"M"]] <- M
DF[[2,"M"]] <- M
colprods_byname(DF$M[[1]])
colprods_byname(DF$M)
colprods_byname(DF$M, "prods")
res <- DF %>% mutate(
  cs = colprods_byname(M),
  cs2 = colprods_byname(M, rowname = "prod")
)
res$cs2
```

`colsums_byname` *Column sums, sorted by name*

Description

Calculates column sums for a matrix by premultiplying by an identity vector (containing all 1's). In contrast to `colSums` (which returns a `numeric` result), the return value from `colsums_byname` is a matrix. An optional `rowname` for the resulting row vector can be supplied. If `rowname` is `NA` (the default), the row name is set to the row type as given by `rowtype(a)`. If `rowname` is set to `NULL`, the row name is returned empty.

Usage

```
colsums_byname(a, rowname = NA)
```

Arguments

`a`	a matrix or list of matrices from which column sums are desired.
`rowname`	name of the output row containing column sums.

Value

a row vector of type `matrix` containing the column sums of `a`.

Examples

```
library(dplyr)
m <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 3:1))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
m
colsums_byname(m)
colsums_byname(m, rowname = "E.ktoe")
m %>%
  colsums_byname() %>%
  rowsums_byname()
# This also works with lists
colsums_byname(list(m, m))
colsums_byname(list(m, m), rowname = "E.ktoe")
colsums_byname(list(m, m), rowname = NA)
colsums_byname(list(m, m), rowname = NULL)
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
colsums_byname(DF$m[[1]])
colsums_byname(DF$m)
colsums_byname(DF$m, "sums")
res <- DF %>% mutate(
  cs = colsums_byname(m),
  cs2 = colsums_byname(m, rowname = "sum")
)
res$cs2
```

---

coltype *Column type*

Description

Extracts column type of `a`.

Usage

```
coltype(a)
```

Arguments

`a`	the object from which you want to extract column types

Value

the column type of `a`

Examples

```
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype(rowtype = "Commodities") %>% setcoltype("Industries")
coltype(U)
# This also works for lists
coltype(list(U,U))
```

`compare_byname` *Compare matrix entries to a value*

Description

Compares matrix entries to a value, returning a matrix of same size as `a` containing `TRUE` or `FALSE` values as the result of applying `compare_fun` and `val` to all entries in `a`.

Usage

```
compare_byname(a, compare_fun = c("==", "!=", "<", "<=", ">=", ">"), val = 0)
```

Arguments

`a`	a matrix or list of matrices whose values are to be counted according to `compare_fun`
`compare_fun`	the comparison function, one of "==", "!=", "<", "<=", ">=", or ">". Default is "==".
`val`	a single value against which entries in matrix `a` are compared. Default is `0`.

Value

a logical matrix of same size as `a` containing `TRUE` where the criterion is met, `FALSE` otherwise

Examples

```
m <- matrix(c(0, 1, 2, 3, 4, 0), nrow = 3, ncol = 2)
compare_byname(m, "<", 3)
compare_byname(list(m,m), "<", 3)
```

`complete_and_sort` *Complete matrices relative to one another and sort into same row, column order*

Description

Completes each matrix relative to each other, thereby assuring that both matrices have same row and column names. Missing rows and columns (relative to the other matrix) are filled with `fill`. Thereafter, rows and columns of the matrices are sorted such that they are in the same order (by name). To complete rows of `m1` relative to columns of `m2`, set the `m2` argument to `transpose_byname(m2)`.

Usage

```
complete_and_sort(
  a,
  b,
  fill = 0,
  margin = c(1, 2),
  roworder = NA,
  colorder = NA
)
```

Arguments

`a`	The first matrix
`b`	The second (optional) matrix.
`fill`	rows and columns added to `a` and `b` will contain the value `fill`. (a double)
`margin`	Specifies the dimension(s) of `a` and `b` over which completing and sorting will occur
`roworder`	Specifies a custom ordering for rows of returned matrices. Unspecified rows are dropped.
`colorder`	Specifies a custom ordering for columns of returned matrices. Unspecified columns are dropped.

Details

`margin` has nearly the same semantic meaning as in `apply`. For rows only, give `1`; for columns only, give `2`; for both rows and columns, give `c(1,2)`, the default value.

If only `m1` is specified, rows of `m1` are completed and sorted relative to columns of `m1`. If neither `m1` nor `m2` have dimnames, `m1` and `m2` are returned unmodified. If only one of `m1` or `m2` has dimnames, an error is thrown.

Value

A named list containing completed and sorted versions of `a` and `b`.

Examples

```
m1 <- matrix(c(1:6), nrow=3, dimnames = list(c("r1", "r2", "r3"), c("c2", "c1")))
m2 <- matrix(c(7:12), ncol=3, dimnames = list(c("r3", "r4"), c("c2", "c3", "c4")))
complete_and_sort(m1)
complete_and_sort(m1, m2)
complete_and_sort(m1, m2, roworder = c("r3", "r2", "r1"))
complete_and_sort(m1, m2, colorder = c("c4", "c3")) # Drops un-specified columns
complete_and_sort(m1, m2, margin = 1)
complete_and_sort(m1, m2, margin = 2)
complete_and_sort(m1, t(m2))
complete_and_sort(m1, t(m2), margin = 1)
complete_and_sort(m1, t(m2), margin = 2)
v <- matrix(1:6, ncol=2, dimnames=list(c("r3", "r1", "r2"), c("c2", "c1")))
complete_and_sort(v, v)
# Also works with lists
complete_and_sort(list(m1,m1), list(m2,m2))
```

---

complete_rows_cols *Complete rows and columns in one matrix relative to another*

Description

"Completing" rows and columns means that `a` contains a union of rows and columns between `a` and `m`, with missing data represented by the value for `fill` (0, by default).

Usage

```
complete_rows_cols(
  a = NULL,
  mat = NULL,
  fill = 0,
  fillrow = NULL,
  fillcol = NULL,
  margin = c(1, 2)
)
```

Arguments

`a`	a matrix or list of matrices to be completed.
`mat`	a `matrix` from which `dimnames` will be extracted for the purposes of completing `a` with respect to `mat`.
`fill`	rows and columns added to `a` will contain the value `fill`. (Default is 0.)
`fillrow`	a row vector of type `matrix` with same column names as `a`. Any rows added to `a` will be `fillrow`. If non-`NULL`, `fillrow` takes precedence over both `fillcol` and `fill` in the case of conflicts.

`fillcol` a column vector of type `matrix` with same row names as `a`. Any columns added to `a` will be `fillcol`. If non-`NULL`, `fillcol` takes precedence over `fill` in the case of conflicts.

`margin` specifies the subscript(s) in `a` over which completion will occur `margin` has nearly the same semantic meaning as in `apply` For rows only, give `1`; for columns only, give `2`; for both rows and columns, give `c(1,2)`, the default value.

Details

Note that `complete_rows_cols(mat1, mat2)` and `complete_rows_cols(mat2, mat1)` are not guaranteed to have the same order for rows and columns. (Nor are the values in the matrix guaranteed to have the same positions.) If `dimnames(mat)` is NULL, `a` is returned unmodified. If either `a` or `matrix` are missing names on a margin (row or column), an error is given. Matrices can be completed relative to themselves, meaning that `a` will be made square, containing the union of row and column names from `a` itself. All added rows and columns will be created from one of the `fill*` arguments. When conflicts arise, precedence among the `fill*` arguments is `fillrow` then `fillcol` then `fill`. Self-completion occurs if `a` is non-NULL and both `is.null(matrix)` and `is.null(names)`. Under these conditions, no warning is given. If `is.null(names)` and dimnames of `matrix` cannot be determined (because, for example, `matrix` doesn't have any dimnames), `a` is completed relative to itself and a warning is given.

Value

A modified version of `a` possibly containing additional rows and columns whose names are obtained from `matrix`

Examples

```
m1 <- matrix(c(1:6), nrow=3, dimnames = list(c("r1", "r2", "r3"), c("c1", "c2")))
m2 <- matrix(c(7:12), ncol=3, dimnames = list(c("r2", "r3"), c("c2", "c3", "c4")))
complete_rows_cols(m1, m2) # Adds empty column c4
complete_rows_cols(m1, t(m2)) # Creates r2, r3 columns; c2, c3, c4 rows
complete_rows_cols(m1, m2, margin = 1) # No changes because r2 and r3 already present in m1
complete_rows_cols(m1, m2, margin = 2) # Adds empty columns c3 and c4
complete_rows_cols(m1, t(m2), margin = 1) # Adds empty rows c2, c3, c4
complete_rows_cols(m1, m2, fill = 100) # Adds columns c3 and c4 with 100's
complete_rows_cols(m1, m1) # Nothing added, because everything already present
complete_rows_cols(m1, t(m1)) # Adds empty c1, c2 rows; Adds empty r1, r2, r3 columns
# Same as previous. With missing matrix, complete relative to transpose of m1.
complete_rows_cols(m1)
# Adds rows r10, r11; cols c10, c11
complete_rows_cols(m1, matrix(0, nrow = 2, ncol = 2,
                              dimnames = list(c("r10", "r11"), c("c10", "c11"))))
# Also works with lists
complete_rows_cols(a = list(m1,m1))
complete_rows_cols(a = list(m1,m1), mat = list(m2,m2))
# No changes because r2, r3 already present in m1
complete_rows_cols(a = list(m1,m1), mat = list(m2,m2), margin = 1)
complete_rows_cols(a = list(m1,m1), mat = list(m2,m2), margin = 2)
complete_rows_cols(a = list(m1,m1),
                   mat = make_list(matrix(0, nrow = 2, ncol = 2,
```

```
                                      dimnames = list(c("r10", "r11"), c("c10", "c11"))),
                                    n = 2, lenx = 1))
# fillrow or fillcol can be specified
a <- matrix(c(11, 12, 21, 22), byrow = TRUE, nrow = 2, ncol = 2,
            dimnames = list(c("r1", "r2"), c("c1", "c2")))
b <- matrix(c(1:6), byrow = TRUE, nrow = 3, ncol = 2,
            dimnames = list(c("r1", "r2", "r3"), c("c1", "c2")))
fillrow <- matrix(c(31, 32), byrow = TRUE, nrow = 1, ncol = 2,
                  dimnames = list("r42", c("c1", "c2")))
complete_rows_cols(a = a, mat = b, fillrow = fillrow)
```

`count_vals_byname` *Count the number of matrix entries that meet a criterion*

Description

Expressions can be written in a natural way such as `count_vals_byname(m, "<=", 1)`.

Usage

```
count_vals_byname(
  a,
  compare_fun = c("==", "!=", "<", "<=", ">=", ">"),
  val = 0
)
```

Arguments

`a`	a matrix or list of matrices whose values are to be counted according to `compare_fun`
`compare_fun`	the comparison function, one of "==", "!=", "<", "<=", ">", or ">=". Default is "==".
`val`	the value against which matrix entries are compared. Default is `0`.

Details

Either a single matrix or a list of matrices can be given as the `a` argument. `compare_fun` can be specified as a string (`"!="`) or as a back-quoted function (`` `!=` ``).

Value

an integer indicating the number of entries in `a` that meet the specified criterion

Examples

```
m <- matrix(c(0, 1, 2, 3, 4, 0), nrow = 3, ncol = 2)
count_vals_byname(m) # uses defaults: compare_fun = "==" and val = 0
count_vals_byname(m, compare_fun = "!=")
count_vals_byname(m, compare_fun = `!=`)
# Write expressions in a natural way
count_vals_byname(m, "<=", 1)
# Also works for lists
count_vals_byname(list(m,m), "<=", 1)
```

---

count_vals_incols_byname

Count the number of matrix entries in columns that meet a criterion

---

Description

Expressions can be written in a natural way such as `count_vals_incols_byname(m, "<=", 1)`.

Usage

```
count_vals_incols_byname(
  a,
  compare_fun = c("==", "!=", "<", "<=", ">=", ">"),
  val = 0
)
```

Arguments

`a`	a matrix or list of matrices whose values are to be counted by columns according to `compare_fun`
`compare_fun`	the comparison function, one of "==", "!=", "<", "<=", ">", or ">=". Default is "=="
`val`	the value against which matrix entries are compared. Default is `0`.

Details

Either a single matrix or a list of matrices can be given as the `a` argument. `compare_fun` can be specified as a string (`"!="`) or as a back-quoted function (`` `!=` ``).

Value

an `matrix` with a single row indicating the number of entries in `a` that meet the specified criterion in each column of `a`

Examples

```
m <- matrix(c(0, 1, 2, 3, 4, 0), nrow = 3, ncol = 2)
count_vals_incols_byname(m) # uses defaults: compare_fun = "==" and val = 0
count_vals_incols_byname(m, compare_fun = "!=")
count_vals_incols_byname(m, compare_fun = `!=`)
# Write expressions in a natural way
count_vals_incols_byname(m, "<=", 1)
# Also works for lists
count_vals_incols_byname(list(m,m), "<=", 1)
```

---

count_vals_inrows_byname

Count the number of matrix entries in rows that meet a criterion

---

Description

Expressions can be written in a natural way such as `count_vals_inrows_byname(m, "<=", 1)`.

Usage

```
count_vals_inrows_byname(
  a,
  compare_fun = c("==", "!=", "<", "<=", ">=", ">"),
  val = 0
)
```

Arguments

`a`	a matrix or list of matrices whose values are to be counted by rows according to `compare_fun`
`compare_fun`	the comparison function, one of "==", "!=", "<", "<=", ">", or ">=". Default is "==".
`val`	the value against which matrix entries are compared. Default is `0`.

Details

Either a single matrix or a list of matrices can be given as the `a` argument. `compare_fun` can be specified as a string (`"!="`) or as a back-quoted function (`` `!=` ``).

Value

an `matrix` with a single column indicating the number of entries in `a` that meet the specified criterion in each row of `a`

Examples

```
m <- matrix(c(0, 1, 2, 3, 4, 0), nrow = 3, ncol = 2)
count_vals_inrows_byname(m) # uses defaults: compare_fun = "==" and val = 0
count_vals_inrows_byname(m, compare_fun = "!=")
count_vals_inrows_byname(m, compare_fun = `!=`)
# Write expressions in a natural way
count_vals_inrows_byname(m, "<=", 1)
# Also works for lists
count_vals_inrows_byname(list(m,m), "<=", 1)
```

---

create_colvec_byname *Create column vectors from data*

Description

This function takes data in the `.dat` and creates column vectors.

Usage

```
create_colvec_byname(.dat, dimnames = NA, colname = NA)
```

Arguments

`.dat`	Data to be converted to column vectors.
`dimnames`	The dimension names to be used for creating the column vector, in a list format, or as a data frame column containing a list of the dimension names to be used for each observation.
`colname`	The name of the column of the colvector.

Details

The row and column names in the resulting column vector are taken from the names of `.dat` and `colname`. If set, `dimnames` overrides the names of `.dat` and `colname`.

This function is a "byname" function that can accept a single number, a vector, a list, or a data frame in `.dat`.

Row types and column types are taken from the row type and column type attributes of `.dat`.

Value

A column vector, a list of column vectors, or a data frame column of column vectors, depending on the value of `.dat`.

Examples

```
# Works with single numbers
create_colvec_byname(c(r1 = 1) %>% setrowtype("rt") %>% setcoltype("ct"),
                     colname = "r1")
# Works with vectors
create_colvec_byname(c(r1 = 1, r2 = 2), colname = "c1")
# Works with a list
create_colvec_byname(list(c(r1 = 1, r2 = 2), c(R1 = 3, R2 = 4, R3 = 5)),
                     colname = list("c1", "C1"))
# Works in a tibble, too.
# (Must be a tibble, not a data frame, so that names are preserved.)
dat <- list(c(r1 = 1, r2 = 2),
            c(R1 = 2, R2 = 3),
            c(r1 = 1, r2 = 2, r3 = 3, r4 = 4, r5 = 5, r6 = 6))
cnms <- list("c1", "C1", "c1")
df1 <- tibble::tibble(dat, cnms)
df1
df1 <- df1 %>%
  dplyr::mutate(
    colvec_col = create_colvec_byname(dat, colname = cnms)
  )
df1$colvec_col[[1]]
df1$colvec_col[[2]]
df1$colvec_col[[3]]
```

---

create_matrix_byname *Create a "byname" matrix from a vector*

Description

This function creates a "byname" matrix, or list of matrices, from `.dat`, depending on the input arguments. This function is similar to `matrix()`, but with "byname" characteristics.

Usage

```
create_matrix_byname(.dat, nrow, ncol, byrow = FALSE, dimnames)
```

Arguments

`.dat`	The data to be used to create the matrix, in a list format, or as a data frame column containing a list of the data to be used for each observation.
`nrow`	The number of rows to be used to create the matrix, in a list format, or as a data frame column containing a list of the number of rows to be used for each observation.
`ncol`	The number of columns to be used to create the matrix, in a list format, or as a data frame column containing a list of the number of columns to be used for each observation.

byrow	The argument stating whether the matrix should be filled by rows or by columns (FALSE by column, TRUE by row), in a list format, or as a data frame column containing a list of the byrow argument for each observation. Default is FALSE.
dimnames	The dimension names to be used for creating the matrices, in a list format, or as a data frame column containing a list of the dimension names to be used for each observation.

Details

Row and column names are taken from the dimnames argument.

Any row or column type information on .dat is preserved on output.

Value

A matrix, list of matrices, or column in a data frame, depending on the input arguments.

Examples

```
create_matrix_byname(c(1, 2), nrow = 2, ncol = 1,
                     dimnames = list(c("r1", "r2"), "c1"))
create_matrix_byname(list(1, 2), nrow = list(1, 1), ncol = list(1,1),
                     dimnames = list(list("r1", "c1"), list("R1", "C1")))
```

---

create_rowvec_byname *Create row vectors from data*

Description

This function takes data in the .dat and creates row vectors.

Usage

```
create_rowvec_byname(.dat, dimnames = NA, rowname = NA)
```

Arguments

.dat	Data to be converted to row vectors.
dimnames	The dimension names to be used for creating the row vector, in a list format, or as a data frame column containing a list of the dimension names to be used for each observation.
rowname	The name of the row of the row vector.

Details

The row and column names in the resulting row vector are taken from rowname and the names of .dat. If set, dimnames overrides rowname and the names of .dat.

Row types and column types are taken from the row type and column type attributes of .dat.

This function is a "byname" function that can accept a single number, a vector, a list, or a data frame in .dat.

Value

A row vector, a list of row vectors, or a data frame column of row vectors, depending on the value of `.dat`.

Examples

```
# Works with single numbers
create_rowvec_byname(c(c1 = 1) %>% setrowtype("rt") %>% setcoltype("ct"), rowname = "r1")
# Works with vectors
create_rowvec_byname(c(c1 = 1, c2 = 2), rowname = "r1")
# Works with a list
create_rowvec_byname(list(c(c1 = 1, c2 = 2), c(C1 = 3, C2 = 4, C3 = 5)),
                     rowname = list("r1", "R1"))
# Works in a tibble, too.
# (Must be a tibble, not a data frame, so that names are preserved.)
dat <- list(c(c1 = 1),
            c(C1 = 2, C2 = 3),
            c(c1 = 1, c2 = 2, c3 = 3, c4 = 4, c5 = 5, c6 = 6))
rnms <- list("r1", "R1", "r1")
df1 <- tibble::tibble(dat, rnms)
df1
df1 <- df1 %>%
  dplyr::mutate(
    rowvec_col = create_rowvec_byname(dat, rowname = rnms)
  )
df1$rowvec_col[[1]]
df1$rowvec_col[[2]]
df1$rowvec_col[[3]]
```

---

cumapply_byname *Apply a function cumulatively to a list of matrices or numbers*

Description

`FUN` must be a binary function that also accepts a single argument. The result is a list with first element `FUN(a[[1]])`. For `i >= 2`, elements are `FUN(a[[i]], out[[i-1]])`, where `out` is the result list.

Usage

```
cumapply_byname(FUN, a)
```

Arguments

`FUN`	the function to be applied
`a`	the list of matrices or numbers to which `FUN` will be applied cumulatively

Details

naryapply_byname() and cumapply_byname() are similar. Their differences can be described by considering a data frame. naryapply_byname() applies FUN to several columns (variables) of the data frame. For example, sum_byname() applied to several variables gives another column containing the sums across each row of the data frame. cumapply_byname() applies FUN to successive entries in a single column. For example sum_byname() applied to a single column gives the sum of all numbers in that column.

Value

a list of same length as a containing the cumulative application of FUN to a

Examples

```
cumapply_byname(sum, list(1, 2, 3, 4))
cumapply_byname(sum_byname, list(1, 2, 3, 4))
cumapply_byname(prod, list(1, 2, 3, 4))
cumapply_byname(hadamardproduct_byname, list(1, 2, 3, 4))
```

---

cumprod_byname *Cumulative element-product that respects row and column names*

Description

Provides cumulative element-products along a list or column of a data frame. If a is a single number, a is returned. If a is a list of numbers, a list representing the cumulative product of the numbers is returned. If a is a single matrix, a is returned. If a is a list of matrices, a list representing the cumulative product of the matrices is returned. In this case, each entry in the returned list is product "by name," such that row and column names of the matrices are respected.

Usage

```
cumprod_byname(a)
```

Arguments

a	a number, list of numbers, matrix or list of matrices for which cumulative element product is desired

Details

This function respects groups if a is a variable in a data frame.

Value

a single number, list of numbers, a single matrix, or a list of matrices, depending on the nature of a

Examples

```
cumprod_byname(list(1, 2, 3, 4, 5))
m1 <- matrix(c(1), nrow = 1, ncol = 1, dimnames = list("r1", "c1")) %>%
  setrowtype("row") %>% setcoltype("col")
m2 <- matrix(c(2), nrow = 1, ncol = 1, dimnames = list("r2", "c2")) %>%
  setrowtype("row") %>% setcoltype("col")
m3 <- matrix(c(3), nrow = 1, ncol = 1, dimnames = list("r3", "c3")) %>%
  setrowtype("row") %>% setcoltype("col")
cumprod_byname(list(m1, m2, m3))
```

---

cumsum_byname *Cumulative sum that respects row and column names*

Description

Provides cumulative sums along a list or column of a data frame. If `a` is a single number, `a` is returned. If `a` is a list of numbers, a list representing the cumulative sum of the numbers is returned. If `a` is a single matrix, `a` is returned. If `a` is a list of matrices, a list representing the cumulative sum of the matrices is returned. In this case, each entry in the returned list is sum "by name," such that row and column names of the matrices are respected.

Usage

```
cumsum_byname(a)
```

Arguments

`a`	a number, list of numbers, matrix or list of matrices for which cumulative sum is desired

Details

If cumulative sums are desired in the context of a data frame, groups in the data frame are respected if `mutate` is used. See examples.

Value

a single number, list of numbers, a single matrix, or a list of matrices, depending on the nature of `a`

Examples

```
library(dplyr)
library(tibble)
m1 <- matrix(c(1), nrow = 1, ncol = 1, dimnames = list("r1", "c1")) %>%
  setrowtype("row") %>% setcoltype("col")
m2 <- matrix(c(2), nrow = 1, ncol = 1, dimnames = list("r2", "c2")) %>%
  setrowtype("row") %>% setcoltype("col")
m3 <- matrix(c(3), nrow = 1, ncol = 1, dimnames = list("r3", "c3")) %>%
  setrowtype("row") %>% setcoltype("col")
```

```
cumsum_byname(list(m1, m2, m3))
# Groups are respected in the context of mutate.
tibble(grp = c("A", "A", "B"), m = list(m1, m2, m3)) %>% group_by(grp) %>%
  mutate(m2 = cumsum_byname(m))
```

`difference_byname` *Name-wise subtraction of matrices*

Description

Name-wise subtraction of matrices

Usage

```
difference_byname(minuend, subtrahend)
```

Arguments

`minuend`	matrix or constant
`subtrahend`	matrix or constant Performs a union and sorting of row and column names prior to differencing. Zeroes are inserted for missing matrix elements.

Value

A matrix representing the name-wise difference between `minuend` and `subtrahend`

Examples

```
library(dplyr)
difference_byname(100, 50)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
G <- matrix(rev(1:4), ncol = 2, dimnames = list(rev(commoditynames), rev(industrynames))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
U - G # Non-sensical. Row and column names not respected.
difference_byname(U, G) # Row and column names respected! Should be all zeroes.
difference_byname(100, U)
difference_byname(10, G)
difference_byname(G) # When subtrahend is missing, return minuend (in this case, G).
difference_byname(subtrahend = G) # When minuend is missing, return - subtrahend (in this case, -G)
# This also works with lists
difference_byname(list(100, 100), list(50, 50))
difference_byname(list(U,U), list(G,G))
DF <- data.frame(U = I(list()), G = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
```

```
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
difference_byname(DF$U, DF$G)
DF %>% mutate(diffs = difference_byname(U, G))
```

---

elementapply_byname *Apply a function to an element of a matrix specified by rows and columns*

Description

FUN is applied to the element of a that is specified by row and col.

Usage

```
elementapply_byname(FUN, a, row, col, .FUNdots = NULL)
```

Arguments

FUN	a unary function to be applied to specified rows and columns of a
a	the argument to FUN
row	the row name of the element to which FUN will be applied
col	the column name of the element to which FUN will be applied
.FUNdots	a list of additional arguments to FUN. (Default is NULL.)

Details

row and col can be any of row or column names or integer indices or a mix of both.

Value

a, after FUN has been applied to the element at row and col

Examples

```
divide <- function(x, divisor){
  x/divisor
}
m <- matrix(c(1:4), nrow = 2, ncol = 2, dimnames = list(c("r1", "r2"), c("c1", "c2"))) %>%
  setrowtype("row") %>% setcoltype("col")
elementapply_byname(divide, a = m, row = 1, col = 1, .FUNdots = list(divisor = 2))
elementapply_byname(divide, a = m, row = 1, col = 2, .FUNdots = list(divisor = 10))
elementapply_byname(divide, a = m, row = "r2", col = "c2", .FUNdots = list(divisor = 100))
```

equal_byname *Compare two matrices "by name" for equality*

Description

If operands are matrices, they are completed and sorted relative to one another prior to comparison.

Usage

```
equal_byname(..., .summarise = FALSE)
```

Arguments

...	Operands to be compared.
.summarise	Tells whether the operation should be accomplished across lists (FALSE) or down lists (TRUE).

Details

Comparisons are made by `isTRUE(all.equal(a, b))` so that variations among numbers within the computational precision will still return TRUE.

If EXACT comparison is needed, use identical_byname, which compares using `identical(a, b)`.

Value

TRUE iff all information is equal, including row and column types *and* row and column names *and* entries in the matrices.

Examples

```
a <- matrix(1:4, nrow = 2)
b <- matrix(1:4, nrow = 2)
equal_byname(a, b)
equal_byname(a, b + 1e-100)
identical_byname(a, b + 1e-100)
a <- a %>% setrowtype("Industries") %>% setcoltype("Commodities")
equal_byname(a, b) # FALSE because a has row and column types, but b does not.
b <- b %>% setrowtype("Industries") %>% setcoltype("Commodities")
equal_byname(a, b)
dimnames(a) <- list(c("i1", "i2"), c("c1", "c2"))
dimnames(b) <- list(c("c1", "c2"), c("i1", "i2"))
equal_byname(a, b) # FALSE, because row and column names are not equal
dimnames(b) <- dimnames(a)
equal_byname(a, b)
```

exp_byname *Exponential of matrix elements*

Description

Gives the exponential of all elements of a matrix or list of matrices

Usage

```
exp_byname(a)
```

Arguments

a	a matrix of list of matrices

Value

M with each element replaced by its exponential

Examples

```
exp_byname(1)
m <- matrix(c(log(10),log(1),log(1),log(100)),
  nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
exp_byname(m)
```

fractionize_byname *Compute fractions of matrix entries*

Description

This function divides all entries in a by the specified sum, thereby "fractionizing" the matrix.

Usage

```
fractionize_byname(a, margin)
```

Arguments

a	the matrix to be fractionized
margin	If 1 (rows), each entry in a is divided by its row's sum. If 2 (columns), each entry in a is divided by its column's sum. If c(1,2) (both rows and columns), each entry in a is divided by the sum of all entries in a.

Value

a fractionized matrix of same dimensions and same row and column types as a.

Examples

```
M <- matrix(c(1, 5,
              4, 5),
            nrow = 2, ncol = 2, byrow = TRUE,
            dimnames = list(c("p1", "p2"), c("i1", "i2"))) %>%
            setcoltype("Products") %>% setrowtype("Industries")
fractionize_byname(M, margin = c(1,2))
fractionize_byname(M, margin = 1)
fractionize_byname(M, margin = 2)
```

---

geometricmean_byname *Name- and element-wise geometric mean of two matrices.*

Description

Gives the geometric mean of corresponding entries of a and b.

Usage

```
geometricmean_byname(..., .summarise = FALSE)
```

Arguments

...	operands; constants, matrices, or lists of matrices
.summarise	Tells whether the operation should be accomplished across lists (FALSE) or down lists (TRUE).

Details

This function performs a union and sorting of row and column names prior to performing geometric mean. Zeroes are inserted for missing matrix elements.

Value

name-wise geometric mean of operands

Examples

```
library(dplyr)
geometricmean_byname(10, 1000)
geometricmean_byname(10, 1000, 100000)
commoditynames <- c("c1", "c2")
industrynames <- "i1"
U <- matrix(c(10, 1000), ncol = 1, nrow = 2, dimnames = list(commoditynames, industrynames)) %>%
```

```
  setrowtype("Commodities") %>% setcoltype("Industries")
G <- matrix(c(1e3, 1e5), ncol = 1, nrow = 2,
            dimnames = list(rev(commoditynames), rev(industrynames))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
# Non-sensical. Row and column names not respected.
sqrt(U*G)
# Row and column names respected!
geometricmean_byname(U, G)
geometricmean_byname(1000, U)
geometricmean_byname(10, G)
# This also works with lists
geometricmean_byname(list(10, 1000), list(1000, 10))
geometricmean_byname(list(U,U), list(G,G))
DF <- data.frame(U = I(list()), G = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
geometricmean_byname(DF$U, DF$G)
DF %>% mutate(geomeans = geometricmean_byname(U, G))
```

`getcolnames_byname` *Gets column names*

Description

Gets column names in a way that is amenable to use in chaining operations in a functional programming way

Usage

```
getcolnames_byname(a)
```

Arguments

a	The matrix or data frame from which column names are to be retrieved

Value

Column names of `m`.

Examples

```
m <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:3))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
getcolnames_byname(m)
# This also works for lists
getcolnames_byname(list(m,m))
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
```

```
DF[[2,"m"]] <- m
getcolnames_byname(DF$m)
```

`getrownames_byname` *Gets row names*

Description

Gets row names in a way that is amenable to use in chaining operations in a functional programming way

Usage

```
getrownames_byname(a)
```

Arguments

a	The matrix or data frame on which row names are to be retrieved

Value

row names of a

Examples

```
m <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:3))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
getrownames_byname(m)
# This also works for lists
getrownames_byname(list(m,m))
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
getrownames_byname(DF$m)
```

`hadamardproduct_byname`
Name-wise matrix Hadamard multiplication

Description

Performs a union and sorting of names of rows and columns for both multiplicand and multiplier for each sequential multiplication step. Zeroes are inserted for missing matrix elements. Doing so ensures that the dimensions of the multiplicand and multiplier are be conformable for each sequential multiplication.

Usage

```
hadamardproduct_byname(..., .summarise = FALSE)
```

Arguments

...	Operands; constants, matrices, or lists of matrices.
.summarise	When TRUE, operands are multiplied down lists. When FALSE (the default), items multiplied across lists.

Details

The Hadamard product is also known as the entrywise product.

Value

Name-wise element product of operands.

Examples

```
library(dplyr)
hadamardproduct_byname(2, 2)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
G <- matrix(1:4, ncol = 2, dimnames = list(rev(commoditynames), rev(industrynames))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
U * G # Not what is desired, because names aren't aligned
hadamardproduct_byname(U, G)
hadamardproduct_byname(U, G, G)
hadamardproduct_byname(U, 0)
hadamardproduct_byname(0, G)
# This also works with lists
hadamardproduct_byname(list(U, U), list(G, G))
DF <- data.frame(U = I(list()), G = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
hadamardproduct_byname(DF$U, DF$G)
DF %>% mutate(entrywiseprods = hadamardproduct_byname(U, G))
# Also works down lists with `.summarise = TRUE`.
hadamardproduct_byname(list(U, G), .summarise = TRUE)
```

hatinv_byname *Hatize and invert a vector*

Description

When dividing rows or columns of a matrix by elements of a vector, the vector elements are placed on the diagonal of a new matrix, the diagonal matrix is inverted, and the result is pre- or post-multiplied into the matrix. This function performs the hatizing and inverting of vector v in one step and takes advantage of computational efficiencies to achieve the desired result. The computational shortcut is apparent when one observes that the matrix produced by hatizing and inverting a vector is a diagonal matrix whose non-zero elements are the numerical inverses of the individual elements of v. So this function first inverts each element of v then places the inverted elements on the diagonal of a diagonal matrix.

Usage

```
hatinv_byname(v, keep = NULL, inf_becomes = .Machine$double.xmax)
```

Arguments

v	The vector to be hatized and inverted.
keep	See matsbyname::hatize.
inf_becomes	A value to be substitute for any Inf produced by the inversion process. Default is .Machine$double.xmax. If FALSE (the default), Inf is not handled differently. If TRUE, Inf values in the resulting matrix are converted to zeroes.

Details

Note that this function gives the same result as invert_byname(hatize_byname(v)), except that invert_byname(hatize_byname(v)) fails due to a singular matrix error when any of the elements of v are zero. This function will give inf_becomes on the diagonal of the result for each zero element of v, arguably a better answer. The sign of Inf is preserved in the substitution. The default value of Inf_becomes is .Machine$double.xmax. Set inf_becomes to NULL to disable this behavior.

The default behavior is helpful for cases when the result of hatinv_byname is later multiplied by 0 to obtain 0. Multiplying Inf by 0 gives NaN which would effectively end the stream of calculations.

Value

a square diagonal matrix with inverted elements of v on the diagonal

Examples

```
v <- matrix(1:10, ncol = 1, dimnames = list(c(paste0("i", 1:10)), c("c1"))) %>%
  setrowtype("Industries") %>% setcoltype(NA)
r <- matrix(1:5, nrow = 1, dimnames = list(c("r1"), c(paste0("c", 1:5)))) %>%
  setrowtype(NA) %>% setcoltype("Commodities")
```

```
hatinv_byname(v, keep = "rownames")
hatinv_byname(r, keep = "colnames")
# This function also works with lists.
hatinv_byname(list(v, v), keep = "rownames")
# Watch out for 0 values
v2 <- matrix(0:1, ncol = 1, dimnames = list(c(paste0("i", 0:1)), c("p1"))) %>%
  setrowtype("Industries") %>% setcoltype(NA)
# Produces singular matrix error
## Not run: v2 %>% hatize_byname() %>% invert_byname
# Handles 0 values well
hatinv_byname(v2, keep = "rownames")
hatinv_byname(v2, inf_becomes = 42, keep = "rownames")
hatinv_byname(v2, inf_becomes = NA, keep = "rownames")
# Deals with 1x1 matrices well, if the `keep` argument is set.
m <- matrix(42, nrow = 1, ncol = 1, dimnames = list("r1", "c1")) %>%
  setrowtype("Product -> Industry") %>%
  setcoltype("Industry -> Product")
m %>%
  hatinv_byname(keep = "rownames")
m %>%
  hatinv_byname(keep = "colnames")
```

`hatize_byname` *Creates a diagonal "hat" matrix from a vector*

Description

A "hat" matrix (or a diagonal matrix) is one in which the only non-zero elements are along on the diagonal. To "hatize" a vector is to place its elements on the diagonal of an otherwise-zero square matrix. `v` must be a matrix object with at least one of its two dimensions of length 1 (i.e., a vector). The names on both dimensions of the hatized matrix are the same and taken from the dimension of `v` that is *not* 1. Note that the row names and column names are sorted prior to forming the "hat" matrix.

Usage

```
hatize_byname(v, keep = NULL)
```

Arguments

`v`	The vector from which a "hat" matrix is to be created.
`keep`	One of "rownames" or "colnames" or `NULL`. If `NULL`, the default, names are kept from the dimension that is not size 1.

Details

Hatizing a 1x1 vector is potentially undefined. The argument `keep` determines whether to keep "rownames" or "colnames". By default `keep` is `NULL`, meanding that the function should attempt to

figure out which dimension's names should be used for the hatized matrix on output. If vector v could ever be 1x1, it is best to set a value for keep when writing code that calls hatize_byname().

If the caller specifies keep = "colnames" when v is a column vector, an error is thrown. If the caller specifies keep = "rownames" when v is a row vector, an error is thrown.

Value

A square "hat" matrix with size equal to the length of v.

Examples

```
v <- matrix(1:10, ncol = 1, dimnames = list(c(paste0("i", 1:10)), c("c1"))) %>%
  setrowtype("Industries") %>% setcoltype(NA)
hatize_byname(v, keep = "rownames")
r <- matrix(1:5, nrow = 1, dimnames = list(c("r1"), c(paste0("c", 1:5)))) %>%
  setrowtype(NA) %>% setcoltype("Commodities")
hatize_byname(r, keep = "colnames")
# This also works with lists.
hatize_byname(list(v, v), keep = "rownames")
# A 1x1 column vector is a degenerate case.
# Row names and rowtype are transferred to the column.
matrix(42, nrow = 1, ncol = 1, dimnames = list("r1")) %>%
  setrowtype("Product -> Industry") %>%
  hatize_byname(keep = "rownames")
# A 1x1 row vector is a degenerate case.
# Column names and coltype are transferred to the row.
matrix(42, nrow = 1, ncol = 1, dimnames = list(NULL, "c1")) %>%
  setcoltype("Industry -> Product") %>%
  hatize_byname(keep = "colnames")
# A 1x1 matrix with both row and column names generates a failure.
## Not run:
matrix(42, nrow = 1, ncol = 1, dimnames = list("r1", "c1")) %>%
  setrowtype("Product -> Industry") %>%
  setcoltype("Industry -> Product") %>%
  hatize_byname()

## End(Not run)
# But you could specify which you want keep, row names or column names.
m <- matrix(42, nrow = 1, ncol = 1, dimnames = list("r1", "c1")) %>%
  setrowtype("Product -> Industry") %>%
  setcoltype("Industry -> Product")
m %>%
  hatize_byname(keep = "rownames")
m %>%
  hatize_byname(keep = "colnames")
```

---

identical_byname *Compare two matrices "by name" for exact equality*

Description

If operands are matrices, they are completed and sorted relative to one another prior to comparison.

Usage

```
identical_byname(..., .summarise = FALSE)
```

Arguments

`...`	Operands to be compared.
`.summarise`	Tells whether the operation should be accomplished across lists (`FALSE`) or down lists (`TRUE`).

Details

Comparisons are made by `identical(a, b)` so that variations among numbers within the computational precision will return `FALSE`.

If fuzzy comparison is needed, use `equal_byname`, which compares using `isTRUE(all.equal(a, b))`.

Value

`TRUE` iff all information is identical, including row and column types *and* row and column names *and* entries in the matrices.

Examples

```
a <- matrix(1:4, nrow = 2)
b <- matrix(1:4, nrow = 2)
identical_byname(a, b)
identical_byname(a, b + 1e-100)
a <- a %>% setrowtype("Industries") %>% setcoltype("Commodities")
identical_byname(a, b) # FALSE because a has row and column types, but b does not.
b <- b %>% setrowtype("Industries") %>% setcoltype("Commodities")
identical_byname(a, b)
dimnames(a) <- list(c("i1", "i2"), c("c1", "c2"))
dimnames(b) <- list(c("c1", "c2"), c("i1", "i2"))
identical_byname(a, b) # FALSE, because row and column names are not equal
dimnames(b) <- dimnames(a)
identical_byname(a, b)
```

`identize_byname` *Named identity matrix or vector*

Description

Creates an identity matrix (**I**) or vector (**i**) of same size and with same names and same row and column types as `a`.

Usage

```
identize_byname(a, margin = c(1, 2))
```

Arguments

`a`	the matrix whose names and dimensions are to be preserved in an identity matrix or vector
`margin`	determines whether an identity vector or matrix is returned. See details.

Details

Behaviour for different values of `margin` are as follows:

- If `margin = 1`, makes a column matrix filled with 1s. Row names and type are taken from row names and type of `a`. Column name and type are same as column type of `a`.
- If `margin = 2`, make a row matrix filled with 1s. Column names and type are taken from column name and type of `a`. Row name and type are same as row type of `a`.
- If `list(c(1,2))` (the default), make an identity matrix with 1s on the diagonal. Row and column names are sorted on output.

Value

An identity matrix or vector.

Examples

```
M <- matrix(1:16, ncol = 4, dimnames=list(c(paste0("i", 1:4)), paste0("c", 1:4))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
identize_byname(M)
identize_byname(M, margin = c(1,2))
identize_byname(M, margin = 1)
identize_byname(M, margin = 2)
N <- matrix(c(-21, -12, -21, -10), ncol = 2, dimnames = list(c("b", "a"), c("b", "a"))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
identize_byname(N)
# This also works with lists
identize_byname(list(M, M))
```

Iminus_byname *Subtract a matrix with named rows and columns from a suitably named and sized identity matrix (I)*

Description

The order of rows and columns of m may change before subtracting from I, because the rows and columns are sorted by name prior to subtracting from I. Furthermore, if m is not square, it will be made square before subtracting from I by calling complete_and_sort.

Usage

```
Iminus_byname(a)
```

Arguments

a	the matrix to be subtracted from I

Value

The difference between an identity matrix (I) and m (whose rows and columns have been completed and sorted)

Examples

```
m <- matrix(c(-21, -12, -21, -10), ncol = 2, dimnames = list(c("b", "a"), c("b", "a"))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
# Rows and columns are unsorted
diag(1, nrow = 2) - m
# Rows and columns are sorted prior to subtracting from the identity matrix
Iminus_byname(m)
# This also works with lists
Iminus_byname(list(m,m))
# If the m is not square before subtracting from I,
# it will be made square by the function complete_and_sort.
m2 <- matrix(c(1,2,3,4,5,6), ncol = 2, dimnames = list(c("a", "b", "c"), c("a", "b"))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
Iminus_byname(m2)
```

invert_byname *Invert a matrix*

Description

This function transposes row and column names as well as row and column types. Rows and columns of a are sorted prior to inverting.

Usage

```
invert_byname(a)
```

Arguments

a	the matrix to be inverted. a must be square.

Value

the inversion of a

Examples

```
m <- matrix(c(10,0,0,100), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
invert_byname(m)
matrixproduct_byname(m, invert_byname(m))
matrixproduct_byname(invert_byname(m), m)
invert_byname(list(m,m))
```

---

iszero_byname *Test whether this is the zero matrix*

Description

Note that this function tests whether the elements of `abs(a)` are `<= tol`. So, you can set `tol = 0` to discover if a is EXACTLY the zero matrix.

Usage

```
iszero_byname(a, tol = 1e-06)
```

Arguments

a	a matrix of list of matrices
tol	the allowable deviation from 0 for any element

Value

`TRUE` iff this is the zero matrix within `tol`.

Examples

```
zero <- matrix(0, nrow = 50, ncol = 50)
iszero_byname(zero)
nonzero <- matrix(1:4, nrow = 2)
iszero_byname(nonzero)
# Also works for lists
iszero_byname(list(zero, nonzero))
# And it works for data frames
DF <- data.frame(A = I(list()), B = I(list()))
DF[[1,"A"]] <- zero
DF[[2,"A"]] <- nonzero
DF[[1,"B"]] <- nonzero
DF[[2,"B"]] <- zero
iszero_byname(DF$A)
iszero_byname(DF$B)
iszero_byname(matrix(1e-10, nrow = 2))
iszero_byname(matrix(1e-10, nrow = 2), tol = 1e-11)
```

---

kvec_from_template_byname

Create a constant vector from matrix a

Description

This function creates a vector using a as a template and k as its value. Row names are taken from the row names of a. The column name of the output is given by colname. Row and column types are transferred from a to the output, directly.

Usage

```
kvec_from_template_byname(a, k = 1, colname = NA, column = TRUE)
```

Arguments

a	The template matrix for the column vector.
k	The value of the entries in the output column vector.
colname	The name of the output vector's 1-sized dimension (the only column if column is TRUE, the only row otherwise).
column	Tells whether a column vector (if TRUE, the default) or a row vector (if FALSE) should be created.

Details

If column is TRUE, the output is a column vector with row names taken from row names of a and a column named by colname. If column is FALSE, the output is a row vevtor with column names taken from column names of a and a row named by colname.

Value

A vector vector formed from a.

Examples

```
kvec_from_template_byname(matrix(42, nrow = 4, ncol = 2,
                                 dimnames = list(c("r1", "r2", "r3", "r4"), c("c1", "c2"))),
                          colname = "new column")
kvec_from_template_byname(matrix(42, nrow = 4, ncol = 2,
                                 dimnames = list(c("r1", "r2", "r3", "r4"), c("c1", "c2"))),
                          colname = "new row", column = FALSE)
```

---

list_of_rows_or_cols *Named list of rows or columns of matrices*

Description

This function takes matrix m and converts it to a list of single-row (if margin == 1) or single-column(if margin == 2) matrices. Each item in the list is named for its row (if margin == 1) or column (if margin == 2).

Usage

```
list_of_rows_or_cols(a, margin)
```

Arguments

a	a matrix or list of matrices (say, from a column of a data frame)
margin	the margin of the matrices to be extracted (1 for rows, 2 for columns)

Details

Note that the result provides column vectors, regardless of the value of margin.

Value

a named list of rows or columns extracted from m

Examples

```
m <- matrix(data = c(1:6),
            nrow = 2, ncol = 3,
            dimnames = list(c("p1", "p2"), c("i1", "i2", "i3"))) %>%
  setrowtype(rowtype = "Products") %>% setcoltype(coltype = "Industries")
list_of_rows_or_cols(m, margin = 1)
list_of_rows_or_cols(m, margin = 2)
```

logarithmicmean_byname *Name- and element-wise logarithmic mean of matrices*

Description

The logarithmic mean of corresponding entries of a and b is 0 if a = 0 or b = 0, a if a = b, or (b - a) / (log(b) - log(a)) otherwise.

Usage

```
logarithmicmean_byname(a, b, base = exp(1))
```

Arguments

a	first operand (a matrix or constant value or lists of same).
b	second operand (a matrix or constant value or lists of same).
base	the base of the logarithm used when computing the logarithmic mean. (Default is base = exp(1).)

Details

This function performs a union and sorting of row and column names prior to performing logarithmic mean. Zeroes are inserted for missing matrix elements.

Internally, the third condition is implemented as (b - a) / log(b/a).

Note that (b - a) / log(b/a) = (a - b) / log(a/b), so logarithmic mean is commutative; the order of arguments a and b does not change the result.

Value

A matrix representing the name-wise logarithmic mean of a and b.

Examples

```
library(dplyr)
m1 <- matrix(c(1:6), nrow = 3, ncol = 2) %>%
  setrownames_byname(c("r1", "r2", "r3")) %>% setcolnames_byname(c("c1", "c2")) %>%
  setrowtype("row") %>% setcoltype("col")
m2 <- matrix(c(7:12), nrow = 3, ncol = 2) %>%
  setrownames_byname(c("r2", "r3", "r4")) %>% setcolnames_byname(c("c2", "c3")) %>%
  setrowtype("row") %>% setcoltype("col")
logarithmicmean_byname(m1, m2)
# This also works with lists
logarithmicmean_byname(list(m1, m1), list(m2, m2))
DF <- data.frame(m1 = I(list()), m2 = I(list()))
DF[[1,"m1"]] <- m1
DF[[2,"m1"]] <- m1
```

```
DF[[1,"m2"]] <- m2
DF[[2,"m2"]] <- m2
logarithmicmean_byname(DF$m1, DF$m2)
DF %>% mutate(logmeans = logarithmicmean_byname(m1, m2))
```

`logmean` *Logarithmic mean of two numbers*

Description

Calculates the logarithmic mean of two numbers.

Usage

```
logmean(a, b, base = exp(1))
```

Arguments

a	the first operand (must be non-negative)
b	the second operand (must be non-negative)
base	the base of the logarithm used in this calculation. (Default is `exp(1)`.)

Details

This is an internal helper function for `logarithmicmean_byname`.

Value

`0` if `a = 0` or `b = 0`; `x1` if `a == b`; and `(a - b) / log(a/b, base = base)` for all other values of `a` and `b`

Examples

```
matsbyname:::logmean(0, 0) # 0
matsbyname:::logmean(0, 1) # 0
matsbyname:::logmean(1, 0) # 0
matsbyname:::logmean(1, 1) # 1
matsbyname:::logmean(2, 1)
matsbyname:::logmean(1, 2) # commutative
matsbyname:::logmean(1, 10) # base = exp(1), the default
matsbyname:::logmean(1, 10, base = 10)
```

log_byname *Logarithm of matrix elements*

Description

Specify the base of the log with base argument.

Usage

```
log_byname(a, base = exp(1))
```

Arguments

a	a matrix or list of matrices
base	the base of the logarithm (default is exp(1), giving the natural logarithm)

Value

M with each element replaced by its base base logarithm

Examples

```
log_byname(exp(1))
m <- matrix(c(10,1,1,100), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
log_byname(m)
log_byname(m, base = 10)
```

make_list *Makes a list of items in x, regardless of x's type*

Description

Repeats x as necessary to make n of them. Does not try to simplify x.

Usage

```
make_list(x, n, lenx = ifelse(is.vector(x), length(x), 1))
```

Arguments

x	the object to be duplicated
n	the number of times to be duplicated
lenx	the length of item x. Normally lenx is taken to be length(x), but if x is itself a list, you may wish for the list to be duplicated several times. In that case, set lenx = 1.

Value

a list of x duplicated n times

Examples

```
m <- matrix(c(1:6), nrow=3, dimnames = list(c("r1", "r2", "r3"), c("c2", "c1")))
make_list(m, n = 1)
make_list(m, n = 2)
make_list(m, n = 5)
make_list(list(c(1,2), c(1,2)), n = 4)
m <- matrix(1:4, nrow = 2)
l <- list(m, m+100)
make_list(l, n = 4)
make_list(l, n = 1) # Warning because l is trimmed.
make_list(l, n = 5) # Warning because length(l) (i.e., 2) not evenly divisible by 5
make_list(list(c("r10", "r11"), c("c10", "c11")), n = 2) # Confused by x being a list
make_list(list(c("r10", "r11"), c("c10", "c11")), n = 2, lenx = 1) # Fix by setting lenx = 1
```

---

margin_from_types_byname

Translate row and column types to integer margins

Description

Converts row and column types to integer margins, based on a and types. If types is not a character vector, types is returned unmodified. If types is a character vector, an integer vector is returned corresponding to the margins on which types are found. If types are not found in the row or column types of a, NA_integer_ is returned.

Usage

```
margin_from_types_byname(a, types)
```

Arguments

a	A matrix or list of matrices.
types	A character vector or list of character vectors representing row or column types whose corresponding integer margins in a are to be determined.

Value

A vector of integers or list of vectors of integers corresponding to the margins on which types exist.

Examples

```
# Works for single matrices
m <- matrix(1) %>%
  setrowtype("Product") %>% setcoltype("Industry")
margin_from_types_byname(m, "Product")
margin_from_types_byname(m, "Industry")
margin_from_types_byname(m, c("Product", "Industry"))
margin_from_types_byname(m, c("Industry", "Product"))
# Works for lists of matrices
margin_from_types_byname(list(m, m), types = "Product")
margin_from_types_byname(list(m, m), types = "Industry")
margin_from_types_byname(list(m, m), types = c("Product", "Product"))
margin_from_types_byname(list(m, m), types = c("Industry", "Industry"))
margin_from_types_byname(list(m, m), types = c("Product", "Industry"))
margin_from_types_byname(list(m, m), types = list("Product", "Industry"))
margin_from_types_byname(list(m, m), types = list(c("Product", "Industry")))
margin_from_types_byname(list(m, m), types = list(c("Product", "Industry"),
                                                  c("Product", "Industry")))
# Works in a data frame
m2 <- matrix(2) %>%
  setrowtype("Industry") %>% setcoltype("Product")
df <- tibble::tibble(m = list(m, m2), types = list("Product", c("Product", "Industry")))
res <- df %>%
  dplyr::mutate(
    margin = margin_from_types_byname(m, types)
 )
res$margin
```

---

matricize_byname *Matricize a vector*

Description

Converts a vector with rows or columns named according to `notation` into a matrix.

Usage

```
matricize_byname(a, notation)
```

Arguments

`a`	a row (column) vector to be converted to a matrix based on its row (column) names.
`notation`	a string vector created by `notation_vec()` that identifies the notation for row or column names.

Value

a matrix created from vector `a`.

Examples

```
v <- matrix(c(1,
              2,
              3,
              4),
            nrow = 4, ncol = 1, dimnames = list(c("p1 -> i1",
                                                  "p2 -> i1",
                                                  "p1 -> i2",
                                                  "p2 -> i2"))) %>%
  setrowtype("Products -> Industries")
# Default separator is " -> ".
matricize_byname(v, notation = RCLabels::arrow_notation)
```

`matrixproduct_byname` *Name-wise matrix multiplication*

Description

Multiplies operands from left to right (when `.summarise = FALSE`). If `.summarise = TRUE`, operands are multiplied from first to last.

Usage

```
matrixproduct_byname(..., .summarise = FALSE)
```

Arguments

`...`	Operands; constants, matrices, or lists of matrices.
`.summarise`	When `TRUE`, a matrix multiplication proceeds down lists of arguments. When `FALSE` (the default), items are multiplied across lists.

Details

Performs a union and sorting of multiplicand rows and multiplier columns by name prior to multiplication. Zeroes are inserted for missing matrix elements. Doing so ensures that the dimensions of multiplicand and multiplier matrices will be conformable. I.e., the number of columns in multiplicand will equal the number of rows in multiplier, so long as the column names of multiplicand are unique and the row names of multiplier are unique. If column type of the multiplicand is not same as row type of the multiplier on any step of the multiplication, the function will fail. The result is matrix product with row names from the first multiplicand and column names from the last multiplier.

Value

A matrix representing the name-wise product of operands.

Examples

```
library(dplyr)
V <- matrix(1:6, ncol = 3, dimnames = list(c("i1", "i2"), c("c1", "c2", "c3"))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
G <- matrix(1:4, ncol = 2, dimnames = list(c("c2", "c1"), c("i2", "i1"))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
Z <- matrix(11:14, ncol = 2, dimnames = list(c("i1", "i2"), c("s1", "s2"))) %>%
  setrowtype("Industries") %>% setcoltype("Sectors")
# Succeeds because G is completed to include a row named c3 (that contains zeroes).
matrixproduct_byname(V, G)
## Not run: V %*% G # Fails because E lacks a row named c3.
matrixproduct_byname(V, G, Z)
# This also works with lists
matrixproduct_byname(list(V,V), list(G,G))
DF <- data.frame(V = I(list()), G = I(list()))
DF[[1,"V"]] <- V
DF[[2,"V"]] <- V
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
matrixproduct_byname(DF$V, DF$G)
DF %>% mutate(matprods = matrixproduct_byname(V, G))
# Also works with lists, multiplying down the lists if `.summarise = TRUE`.
matrixproduct_byname(list(V, G, Z), .summarise = TRUE)
```

---

mean_byname *Name- and element-wise arithmetic mean of matrices*

Description

Gives the arithmetic mean of operands in ....

Usage

```
mean_byname(..., .summarise = FALSE)
```

Arguments

...	Operands: constants, matrices, or lists of matrices.
.summarise	Tells whether the operation should be accomplished across lists (FALSE) or down lists (TRUE).

Details

This function performs a union and sorting of row and column names prior to performing arithmetic mean. Zeroes are inserted for missing matrix elements.

Value

name-wise arithmetic mean of operands.

Examples

```
library(dplyr)
mean_byname(100, 50)
mean_byname(10, 20, 30)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
G <- matrix(rev(1:4), ncol = 2, dimnames = list(rev(commoditynames), rev(industrynames))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
(U + G) / 2 # Non-sensical. Row and column names not respected.
mean_byname(U, G) # Row and column names respected! Should be 1, 2, 3, and 4.
mean_byname(U, G, G)
mean_byname(100, U)
mean_byname(100, 50, U)
mean_byname(10, G)
# This also works with lists
mean_byname(list(100, 100), list(50, 50))
mean_byname(list(U,U), list(G,G))
DF <- data.frame(U = I(list()), G = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
mean_byname(DF$U, DF$G)
DF %>% mutate(means = mean_byname(U, G))
```

`naryapplylogical_byname` *Apply a function logically to numbers, matrices, or lists of numbers or matrices*

Description

Operands should be logical, although numerical operands are accepted. Numerical operands are interpreted as `0` is `FALSE`, and any other number is `TRUE`.

Usage

```
naryapplylogical_byname(
  FUN,
  ...,
  .FUNdots = NULL,
  match_type = c("all", "matmult", "none"),
  set_rowcoltypes = TRUE,
  .organize = TRUE,
  .summarise = FALSE
)
```

Arguments

FUN	a binary function (that returns logical values) to be applied over operands
...	operands; constants, matrices, or lists of matrices
.FUNdots	a list of additional named arguments passed to FUN.
match_type	One of "all", "matmult", or "none". When ... are matrices, "all" (the default) indicates that rowtypes of all ... matrices must match and coltypes of all ... matrices must match. If "matmult", the coltype of the first operand must match the rowtype of the second operand for every sequential invocation of FUN. If "none", neither coltypes nor rowtypes are checked by naryapply_byname().
set_rowcoltypes	Tells whether to apply row and column types from operands in ... to the output of each sequential invocation of FUN. Set TRUE (the default) to apply row and column types. Set FALSE, to *not* apply row and column types to the output.
.organize	A boolean that tells whether or not to automatically complete operands in ... relative to each other and sort the rows and columns of the completed matrices. This organizing is done on each sequential invocation of FUN. Normally, this should be TRUE (the default). However, if FUN takes over this responsibility, set to FALSE.
.summarise	A boolean that tells whether this call is considered a summarise operation (like dplyr::summarise()). Default is FALSE.

Details

This function is not exported, thereby retaining the right to future changes.

Value

the result of FUN applied logically to ...

Examples

```
matsbyname:::naryapplylogical_byname(`&`, TRUE, TRUE, TRUE)
matsbyname:::naryapplylogical_byname(`&`, TRUE, TRUE, FALSE)
```

---

naryapply_byname *Apply a function "by name" to any number of operands*

Description

Applies FUN to all operands in .... Other arguments have similar meaning as binaryapply_byname(). See details for more information.

Usage

```
naryapply_byname(
  FUN,
  ...,
  .FUNdots = NULL,
  match_type = c("all", "matmult", "none"),
  set_rowcoltypes = TRUE,
  .organize = TRUE,
  .summarise = FALSE
)
```

Arguments

FUN	a binary function to be applied "by name" to all operands in ....
...	the operands for FUN.
.FUNdots	a list of additional named arguments passed to FUN.
match_type	One of "all", "matmult", or "none". When ... are matrices, "all" (the default) indicates that rowtypes of all ... matrices must match and coltypes of all ... matrices must match. If "matmult", the coltype of the first operand must match the rowtype of the second operand for every sequential invocation of FUN. If "none", neither coltypes nor rowtypes are checked by naryapply_byname().
set_rowcoltypes	Tells whether to apply row and column types from operands in ... to the output of each sequential invocation of FUN. Set TRUE (the default) to apply row and column types. Set FALSE, to *not* apply row and column types to the output.
.organize	A boolean that tells whether or not to automatically complete operands in ... relative to each other and sort the rows and columns of the completed matrices. This organizing is done on each sequential invocation of FUN. Normally, this should be TRUE (the default). However, if FUN takes over this responsibility, set to FALSE.
.summarise	A boolean that tells whether this call is considered a summarise operation (like dplyr::summarise()). Default is FALSE.

Details

If only one ... argument is supplied, FUN must be capable of handling one argument, and the call is routed to unaryapply_byname(). When set_rowcoltypes is TRUE, the rowcoltypes argument of unaryapply_byname() is set to "all", but when set_rowcoltypes is FALSE, the rowcoltypes argument of unaryapply_byname() is set to "none". If finer control is desired, the caller should use unaryapply_byname() directly. If more than one argument is passed in ..., FUN must be a binary function, but its use in by naryapply_byname() is "n-ary." Arguments match_type, set_rowcoltypes, and .organize have same meaning as for binaryapply_byname(). Thus, all of the operands in ... must obey the rules of type matching when match_type is TRUE.

naryapply_byname() and cumapply_byname() are similar. Their differences can be described by considering a data frame. naryapply_byname() applies FUN to several columns (variables) of the data frame. For example, sum_byname() applied to several variables gives another column containing the sums across each row of the data frame. cumapply_byname() applies FUN to successive

entries in a single column. For example sum_byname() applied to a single column gives the sum of all numbers in that column.

Value

the result of applying FUN to all operands in . . .

Examples

```
naryapply_byname(FUN = sum_byname, 2, 3)
naryapply_byname(FUN = sum_byname, 2, 3, 4, -4, -3, -2)
# Routes to unaryapply_byname
naryapply_byname(FUN = `^`, list(1,2,3), .FUNdots = list(2))
```

---

ncol_byname *Get the number of columns in a "byname" matrix.*

Description

The functionn gets the number of columns in a "byname" matrix, or for each "byname" matrix contained in a column of a data frame.

Usage

```
ncol_byname(a)
```

Arguments

a	A matrix or a column of a data frame populated with "byname" matrices.

Value

The number of columns of the matrix, or a list containing the number of columns in each of the matrices contained in the column of a data frame.

Examples

```
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2", "i3")
U2 <- matrix(1:3, ncol = length(industrynames),
           nrow = length(productnames), dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
productnames <- c("p1", "p2", "p3")
```

```
industrynames <- c("i1", "i2", "i3", "i4")
U3 <- matrix(1:4, ncol = length(industrynames),
           nrow = length(productnames), dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
dfUs <- data.frame(
  year = numeric(),
  matrix_byname = I(list())
)
  dfUs <- data.frame(
year = numeric(),
matrix_byname = I(list())
)
dfUs[[1, "matrix_byname"]] <- U
dfUs[[2, "matrix_byname"]] <- U2
dfUs[[3, "matrix_byname"]] <- U3
dfUs[[1, "year"]] <- 2000
dfUs[[2, "year"]] <- 2001
dfUs[[3, "year"]] <- 2002
number_cols <- ncol_byname(dfUs$matrix_byname) %>%
print()
```

`nrow_byname` *Get the number of rows in a "byname" matrix.*

Description

The functionn gets the number of rows in a "byname" matrix, or for each "byname" matrix contained in a column of a data frame.

Usage

```
nrow_byname(a)
```

Arguments

a	A matrix or a column of a data frame populated with "byname" matrices.

Value

The number of rows of the matrix, or a list containing the number of rows in each of the matrices contained in the column of a data frame.

Examples

```
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
```

```
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2", "i3")
U2 <- matrix(1:3, ncol = length(industrynames),
             nrow = length(productnames), dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
productnames <- c("p1", "p2", "p3")
industrynames <- c("i1", "i2", "i3", "i4")
U3 <- matrix(1:4, ncol = length(industrynames),
             nrow = length(productnames), dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>%
  setcoltype("Industries")
dfUs <- data.frame(
  year = numeric(),
  matrix_byname = I(list())
)
dfUs[[1, "matrix_byname"]] <- U
dfUs[[2, "matrix_byname"]] <- U2
dfUs[[3, "matrix_byname"]] <- U3
dfUs[[1, "year"]] <- 2000
dfUs[[2, "year"]] <- 2001
dfUs[[3, "year"]] <- 2002
number_rows <- matsbyname::nrow_byname(dfUs$matrix_byname)
```

`organize_args` *Organize binary arguments*

Description

Organizes arguments of binary (2 arguments) _byname functions. Actions performed are:

- if only one argument is a list, make the other argument also a list of equal length.
- if both arguments are lists, ensure that they are same length.
- if one argument is a matrix and the other is a constant, make the constant into a matrix.
- ensures that row and column types match for `typematch_margins`.
- ensures that list item names match if both `a` and `b` are lists; no complaints are made if neither `a` nor `b` has names.
- completes and sorts the matrices.

Usage

```
organize_args(a, b, match_type = "all", fill)
```

Arguments

a	the first argument to be organized
b	the second argument to be organized

match_type	one of "all", "matmult", "none". When both a and b are matrices, "all" (the default) indicates that rowtypes of a must match rowtypes of b and coltypes of a must match coltypes of b. If "matmult", coltypes of a must match rowtypes of b.
fill	a replacement value for a or b if either is missing or NULL.

Value

a list with two elements (named a and b) containing organized versions of the arguments

---

pow_byname *Powers of matrix elements*

Description

Gives the result of raising all elements of a matrix or list of matrices to a power.

Usage

```
pow_byname(a, pow)
```

Arguments

a	a matrix of list of matrices
pow	the power to which elements of a will be raised

Value

a with each element raised to pow

Examples

```
library(dplyr)
pow_byname(2, 3)
m <- matrix(2, nrow = 2, ncol = 3, dimnames = list(paste0("r", 1:2), paste0("c", 1:3))) %>%
  setrowtype("rows") %>% setcoltype("cols")
pow_byname(m, 2)
DF <- data.frame(m = I(list()), pow = I(list()))
DF[[1, "m"]] <- m
DF[[2, "m"]] <- m
DF[[1, "pow"]] <- 0.5
DF[[2, "pow"]] <- -1
DF %>% mutate(
  sqrtm = pow_byname(m, 0.5),
  mtopow = pow_byname(m, pow)
)
```

`prepare_.FUNdots` *Prepare the* `.FUNdots` *argument for* `*apply_byname` *functions.*

Description

This is a helper function for the various `*apply_byname` functions.

Usage

```
prepare_.FUNdots(a, .FUNdots)
```

Arguments

`a`	the main argument to an `*apply_byname` function.
`.FUNdots`	a list of additional arguments to be applied to `FUN` in one of the `*apply_byname` functions.

Details

We have four cases between a and any single item of .FUNdots:

- both a and the item of .FUNdots are lists
  - if the item of .FUNdots (a list itself) has length different from 1 or length(a), throw an error
  - if the item of .FUNdots (a list itself) has length 1, replicate the single item to be a list of length = length(a)
  - if the item of .FUNdots (a list itself) has length = length(a), use the item of .FUNdots as is
- a is a list but the item (argument) of .FUNdots is NOT a list
  - if the item of .FUNdots (which is not a list) has length != 1, throw an error, because there is ambiguity how the item of .FUNdots should be treated.
  - if the item of .FUNdots (which is not a list) has length = 1, replicate that single item to be a list of length = length(a)
- a is NOT a list, but the item of .FUNdots IS a list
  - pass the argument along and hope for the best. This situation is probably an error. If so, it will become apparent soon.
- neither a nor the item of .FUNdots is a list
  - a should have length = 1, but a single matrix reports its length as the number of elements of the matrix. So, we can't check length in this situation.
  - the item of .FUNdots is assumed to have length 1 and passed along

Value

a reconfigured version of `.FUNdots`, ready for use by an `*apply_byname` function.

- both a and the item of .FUNdots are lists
  - if the item of .FUNdots (a list itself) has length different from 1 or length(a), throw an error
  - if the item of .FUNdots (a list itself) has length 1, replicate the single item to be a list of length = length(a)
  - if the item of .FUNdots (a list itself) has length = length(a), use the item of .FUNdots as is
- a is NOT a list, but the item of .FUNdots IS a list
  - pass the argument along and hope for the best. This situation is probably an error. If so, it will become apparent soon.
- a is a list but the item (argument) of .FUNdots is NOT a list This situation could be ambiguous. Let's say the list of `a` values has length 2, and an argument `margin = c(1, 2)`. Should `margin = 1` be applied to `a[[1]]` and `margin = 2` be applied to `a[[2]]`? Or should `margin = c(1, 2)` be applied to both `a[[1]]` and `a[[2]]`? This ambiguity should be handled by using the function `prep_vector_arg()` within the function that calls `unaryapply_byname()`. For an example, see `identize_byname()`. When the arguments are coming in from a data frame, there will be no ambiguity, but the information will not be coming `.FUNdots[[i]]` as a list. Optimizing for the data frame case, this function allows vectors of length equal to the length of the list `a`, interpreting such vectors as applying in sequence to each `a` in turn. So the algorithm is as follows:
  - if a non-NULL item of .FUNdots (which is not a list) has length other than 1 or length(a), throw an error.
  - if a non-NULL item of .FUNdots (which is not a list) has length = 1, replicate that single item to be a list of length = length(a).
  - if a non-NULL item of .FUNdots (which is not a list) has length = length(a), leave it as-is.
- neither a nor the item of .FUNdots is a list
  - a should have length = 1, but a single matrix reports its length as the number of elements of the matrix. So, we can't check length in this situation.
  - the item of .FUNdots is assumed to have length 1 and passed along

---

`prep_vector_arg` *Prepare a vector argument*

Description

This is a helper function for many `*_byname` functions.

Usage

```
prep_vector_arg(a, vector_arg)
```

Arguments

a	a matrix or list of matrices
vector_arg	the vector argument over which to apply a calculation

Details

It is potentially ambiguous to specify a vector or matrix argument, say, `margin = c(1, 2)` when applying the `*_byname` functions to unary list of `a`. Rather, one should specify, say, `margin = list(c(1, 2)` to avoid ambiguity. If `a` is a list, `vector_arg` is not a list and has length > 1 and length not equal to the length of `a`, this function returns a list value for `vector_arg`. If `a` is not a list and `vector_arg` is a list, this function returns an un-recursive, unlisted version of `vector_arg`.

Note that if `vector_arg` is a single matrix, it is automatically enclosed by a list when `a` is a list.

Value

`vector_arg`, possibly modified when `a` is a list

Examples

```
m <- matrix(c(2, 2))
matsbyname:::prep_vector_arg(list(m, m), vector_arg = c(1,2))
```

---

prodall_byname *Product of all elements in a matrix*

Description

This function is equivalent to `a %>% rowprods_byname() %>% colprods_byname()`, but returns a single numeric value instead of a 1x1 matrix.

Usage

```
prodall_byname(a)
```

Arguments

a	the matrix whose elements are to be multiplied

Value

the product of all elements in `a` as a numeric.

Examples

```
library(dplyr)
M <- matrix(2, nrow=2, ncol=2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Product")
prodall_byname(M)
rowprods_byname(M) %>% colprods_byname
# Also works for lists
prodall_byname(list(M,M))
DF <- data.frame(M = I(list()))
DF[[1,"M"]] <- M
DF[[2,"M"]] <- M
prodall_byname(DF$M[[1]])
prodall_byname(DF$M)
res <- DF %>% mutate(
  prods = prodall_byname(M)
)
res$prods
```

---

quotient_byname *Name-wise matrix element division*

Description

Element-wise division of two matrices.

Usage

```
quotient_byname(dividend, divisor)
```

Arguments

dividend	Dividend matrix or constant
divisor	Divisor matrix or constant

Details

Performs a union and sorting of names of rows and columns for both `dividend` and `divisor` prior to element division. Zeroes are inserted for missing matrix elements. Doing so ensures that the dimensions of the `dividend` and `divisor` will be conformable.

Value

A matrix representing the name-wise element quotient of `dividend` and `divisor`

Examples

```
library(dplyr)
quotient_byname(100, 50)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
G <- matrix(rev(1:4), ncol = 2, dimnames = list(rev(commoditynames), rev(industrynames))) %>%
  setrowtype("Commodities") %>% setcoltype("Industries")
U / G # Non-sensical.  Names aren't aligned
quotient_byname(U, G)
quotient_byname(U, 10)
quotient_byname(10, G)
# This also works with lists
quotient_byname(10, list(G,G))
quotient_byname(list(G,G), 10)
quotient_byname(list(U, U), list(G, G))
DF <- data.frame(U = I(list()), G = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
DF[[1,"G"]] <- G
DF[[2,"G"]] <- G
quotient_byname(DF$U, DF$G)
DF %>% mutate(elementquotients = quotient_byname(U, G))
```

---

rename_to_piece_byname

Rename matrix rows and columns by piece of row or column names

---

Description

It can be convenient to rename rows or columns of matrices based on retaining only a piece of the row and/or column names. This function provides that capability.

Usage

```
rename_to_piece_byname(
  a,
  piece,
  margin = c(1, 2),
  notation = RCLabels::bracket_notation,
  prepositions = RCLabels::prepositions
)
```

Arguments

a	A matrix or list of matrices whose rows or columns will be renamed.

piece	A character string indicating which piece of the row or column names to retain, one of "noun", "pps", "pref" or "suff", or a preposition, indicating which part of the row or column name is to be retained.
margin	As a character, the row type or column type to be renamed. As an integer, the margin to be renamed. Default is c(1, 2), meaning that both rows (margin = 1) and columns (margin = 2) will be renamed.
notation	The notation used for row and column labels. Default is RCLabels::bracket_notation. See RCLabels.
prepositions	Prepositions that can be used in the row and column label. Default is RCLabels::prepositions.

Details

Internally, this function finds pieces of row and column names via the RCLabels package. piece can be anything that RCLabels::get_piece() understands. Note that margin can be either an integer vector or a character vector. If margin is a character vector, it is interpreted as a row or column type, and margin_from_types_byname() is called internally to resolve the integer margins of interest.

Note that if row and/or column type are present, the row and/or column type are also renamed according to piece.

Value

A version of a with renamed rows and columns.

Examples

```
m <- matrix(c(1, 2,
              3, 4,
              5, 6), nrow = 3, byrow = TRUE,
            dimnames = list(c("a -> b", "r2", "r3"), c("a -> b", "c -> d")))
m
rename_to_piece_byname(m, piece = "pref", notation = RCLabels::arrow_notation)
m2 <- m %>%
  setrowtype("rows") %>% setcoltype("cols")
m2
rename_to_piece_byname(m2, piece = "pref", margin = "rows",
                       notation = RCLabels::arrow_notation)
rename_to_piece_byname(m2, piece = "suff", margin = "rows",
                       notation = RCLabels::arrow_notation)
```

---

rename_to_pref_suff_byname

Rename matrix rows and columns by prefix and suffix

Description

[Superseded] It can be convenient to rename rows or columns of matrices based on retaining prefixes or suffixes. This function provides that capability.

Usage

```
rename_to_pref_suff_byname(a, keep, margin = c(1, 2), notation)
```

Arguments

a	a matrix or list of matrices whose rows or columns will be renamed.
keep	one of "prefix" or "suffix" indicating which part of the row or column name to retain.
margin	one of 1, 2, or c(1, 2) where 1 indicates rows and 2 indicates columns.
notation	See notation_vec().

Details

A prefix is defined by an opening string (prefix_open) and a closing string (prefix_close). A suffix is defined by an opening string (suffix_open) and a closing string (suffix_close). If sep is provided and none of prefix_open, prefix_close, suffix_open, and suffix_close are provided, default arguments become: * prefix_open: "", * prefix_close: sep, * suffix_open: sep, and * suffix_close: "".

The keep parameter tells which portion to retain (prefixes or suffixes),

If prefixes or suffixes are not found in a row and/or column name, that name is unchanged.

Value

a with potentially different row or column names.

Examples

```
# This function is superseded.
# Instead, use `rename_to_pieces_byname()`.
# For example:
m <- matrix(c(1, 2,
              3, 4,
              5, 6), nrow = 3, byrow = TRUE,
            dimnames = list(c("a -> b", "r2", "r3"), c("a -> b", "c -> d")))
m
rename_to_piece_byname(m, piece = "pref", notation = RCLabels::arrow_notation)
# Note, labels are lost, because some labels are missing a suffix.
rename_to_piece_byname(m, piece = "suff", notation = RCLabels::arrow_notation)
# Original documentation:
rename_to_pref_suff_byname(m, keep = "pref", notation = RCLabels::arrow_notation)
rename_to_pref_suff_byname(m, keep = "suff", notation = RCLabels::arrow_notation)
```

replaceNaN_byname *Replace NaN values with a value*

Description

In a matrix or within matrices in a list, replace all NaN matrix values with val.

Usage

```
replaceNaN_byname(a, val = 0)
```

Arguments

a	a matrix of list of matrices in which NaN will be replaced by val
val	NaNs are replace by val

Value

a matrix or list of matrices in which all NaN are replaced by val

Examples

```
suppressWarnings(a <- matrix(c(1, sqrt(-1))))
replaceNaN_byname(a)
replaceNaN_byname(a, 42)
```

rowprods_byname *Row products, sorted by name*

Description

Calculates row products (the product of all elements in a row) for a matrix. An optional colname for the resulting column vector can be supplied. If colname is NULL or NA (the default), the column name is set to the column type as given by coltype(a).

Usage

```
rowprods_byname(a, colname = NA)
```

Arguments

a	a matrix or list of matrices from which row products are desired.
colname	name of the output column containing row products

Value

a column vector of type `matrix` containing the row products of `a`

Examples

```
library(dplyr)
M <- matrix(c(1:6), ncol = 2, dimnames = list(paste0("i", 3:1), paste0("c", 1:2))) %>%
  setrowtype("Industries") %>% setcoltype("Products")
rowprods_byname(M)
rowprods_byname(M, "E.ktoe")
# This also works with lists
rowprods_byname(list(M, M))
rowprods_byname(list(M, M), "E.ktoe")
rowprods_byname(list(M, M), NA)
rowprods_byname(list(M, M), NULL)
DF <- data.frame(M = I(list()))
DF[[1,"M"]] <- M
DF[[2,"M"]] <- M
rowprods_byname(DF$M[[1]])
rowprods_byname(DF$M)
ans <- DF %>% mutate(rs = rowprods_byname(M))
ans
ans$rs[[1]]
# Nonsensical
## Not run: rowprods_byname(NULL)
```

`rowsums_byname` *Row sums, sorted by name*

Description

Calculates row sums for a matrix by post-multiplying by an identity vector (containing all 1's). In contrast to `rowSums` (which returns a `numeric` result), the return value from `rowsums_byname` is a matrix. An optional `colname` for the resulting column vector can be supplied. If `colname` is `NULL` or `NA` (the default), the column name is set to the column type as given by `coltype(a)`. If `colname` is set to `NULL`, the column name is returned empty.

Usage

```
rowsums_byname(a, colname = NA)
```

Arguments

`a`	A matrix or list of matrices from which row sums are desired.
`colname`	The name of the output column containing row sums.

Value

A column vector of type `matrix` containing the row sums of `m`

Examples

```
library(dplyr)
m <- matrix(c(1:6), ncol = 2, dimnames = list(paste0("i", 3:1), paste0("c", 1:2))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
m
rowsums_byname(m)
rowsums_byname(m, "E.ktoe")
# This also works with lists
rowsums_byname(list(m, m))
rowsums_byname(list(m, m), "E.ktoe")
rowsums_byname(list(m, m), NA)
rowsums_byname(list(m, m), NULL)
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
rowsums_byname(DF$m[[1]])
rowsums_byname(DF$m)
ans <- DF %>% mutate(rs = rowsums_byname(m))
ans
ans$rs[[1]]
# Nonsensical
## Not run: rowsums_byname(NULL)
```

---

rowtype *Row type*

Description

Extracts row type of a.

Usage

```
rowtype(a)
```

Arguments

a	the object from which you want to extract row types

Value

the row type of a

Examples

```
library(dplyr)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames)) %>%
  setrowtype(rowtype = "Commodities") %>% setcoltype("Industries")
```

```
rowtype(U)
# This also works for lists
rowtype(list(U,U))
```

---

samestructure_byname *Test whether matrices or lists of matrices have same structure*

Description

Matrices are said to have the same structure if row and column types are identical and if row and column names are identical. Values can be different.

Usage

```
samestructure_byname(..., .summarise = FALSE)
```

Arguments

...	Operands to be compared.
.summarise	Tells whether the operation should be accomplished across lists (FALSE) or down lists (TRUE).

Value

TRUE if all operands have the same structure, FALSE otherwise.

Examples

```
samestructure_byname(2, 2)
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>% setcoltype("Industries")
samestructure_byname(U, U)
samestructure_byname(U, U %>% setrowtype("row"))
samestructure_byname(U %>% setcoltype("col"), U)
# Also works with lists
samestructure_byname(list(U, U), list(U, U))
```

select_cols_byname *Select columns of a matrix (or list of matrices) by name*

Description

Arguments indicate which columns are to be retained and which are to be removed. For maximum flexibility, arguments are extended regex patterns that are matched against column names.

Usage

```
select_cols_byname(a, retain_pattern = "$^", remove_pattern = "$^")
```

Arguments

a	a matrix or a list of matrices
retain_pattern	an extended regex or list of extended regular expressions that specifies which columns of m to retain. Default pattern ($^) retains nothing.
remove_pattern	an extended regex or list of extended regular expressions that specifies which columns of m to remove Default pattern ($^) removes nothing.

Details

If a is NULL, NULL is returned.

Patterns are compared against column names using extended regex. If no column names of a match the retain_pattern, NULL is returned. If no column names of a match the remove_pattern, a is returned.

Retaining columns takes precedence over removing columns, always.

Some typical patterns are:

- ^Electricity$|^Oil$: column names that are EXACTLY Electricity or Oil.
- ^Electricity|^Oil: column names that START WITH Electricity or Oil.
- Electricity|Oil: column names that CONTAIN Electricity or Oil anywhere within them.

Given a list of column names, a pattern can be constructed easily using the make_pattern function.

RCLabels::make_or_pattern() escapes regex strings using Hmisc::escaprRegex(). This function assumes that retain_pattern and remove_pattern have already been suitably escaped.

Note that the default retain_pattern and remove_pattern ($^) retain nothing and remove nothing.

Note that if all columns are removed from a, NULL is returned.

Value

a matrix that is a subset of a with columns selected by retain_pattern and remove_pattern.

Examples

```
m <- matrix(1:16, ncol = 4, dimnames=list(c(paste0("i", 1:4)), paste0("p", 1:4))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
select_cols_byname(m,
                   retain_pattern = RCLabels::make_or_pattern(c("p1", "p4"),
                   pattern_type = "exact"))
select_cols_byname(m,
                   remove_pattern = RCLabels::make_or_pattern(c("p1", "p3"),
                   pattern_type = "exact"))
# Also works for lists and data frames
select_cols_byname(list(m,m), retain_pattern = "^p1$|^p4$")
```

---

select_rows_byname *Select (or de-select) rows of a matrix (or list of matrices) by name*

Description

Arguments indicate which rows are to be retained and which are to be removed. For maximum flexibility, arguments are extended regex patterns that are matched against row names.

Usage

```
select_rows_byname(a, retain_pattern = "$^", remove_pattern = "$^")
```

Arguments

a	a matrix or a list of matrices
retain_pattern	an extended regex or list of extended regular expressions that specifies which rows of m to retain. Default pattern ($^) retains nothing.
remove_pattern	an extended regex or list of extended regular expressions that specifies which rows of m to remove Default pattern ($^) removes nothing.

Details

If a is NULL, NULL is returned.

Patterns are compared against row names using extended regex. If no row names of m match the retain_pattern, NULL is returned. If no row names of m match the remove_pattern, m is returned. Note that the default retain_pattern and remove_pattern ($^) retain nothing and remove nothing.

Retaining rows takes precedence over removing rows, always.

Some typical patterns are:

- ^Electricity$|^Oil$: row names that are EXACTLY Electricity or EXACTLY Oil.
- ^Electricity|^Oil: row names that START WITH Electricity or START WITH Oil.
- Electricity|Oil: row names that CONTAIN Electricity or CONTAIN Oil anywhere within them.

Given a list of row names, a pattern can be constructed easily using the `make_pattern` function. `RCLabels::make_or_pattern()` escapes regex strings using `Hmisc::escapeRegex()`. This function assumes that `retain_pattern` and `remove_pattern` have already been suitably escaped.

Note that if all rows are removed from `a`, `NULL` is returned.

Value

a matrix that is a subset of `m` with rows selected by `retain_pattern` and `remove_pattern`.

Examples

```
m <- matrix(1:16, ncol = 4, dimnames=list(c(paste0("i", 1:4)), paste0("p", 1:4))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
select_rows_byname(m,
                   retain_pattern = RCLabels::make_or_pattern(c("i1", "i4"),
                   pattern_type = "exact"))
select_rows_byname(m,
                   remove_pattern = RCLabels::make_or_pattern(c("i1", "i3"),
                   pattern_type = "exact"))
# Also works for lists and data frames
select_rows_byname(list(m,m), retain_pattern = "^i1$|^i4$")
```

---

setcolnames_byname *Sets column names*

Description

Sets column names in a way that is amenable to use in piping operations in a functional programming way. if `a` is `NULL`, `NULL` is returned. If `a` is a constant, it is converted to a matrix and `colnames` are applied. If `a` is a matrix, `colnames` should be a vector of new column names that is as long as the number of columns in `a`. If `a` is a list of matrices, `colnames` can also be a list, and it should be as long as `a`. Or `colnames` can be a vector of column names which will be applied to every matrix in the list of `a`. Each item in the list should be a vector containing column names for the corresponding matrix in `a`.

Usage

```
setcolnames_byname(a, colnames)
```

Arguments

`a`	A matrix or a list of matrices in which column names are to be set
`colnames`	A vector of new column names or a list of vectors of new column names

Value

a copy of `a` with new column names

Examples

```
m <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:3))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
setcolnames_byname(m, c("a", "b", "c"))
```

---

setcoltype *Sets column type for a matrix or a list of matrices*

Description

This function is a wrapper for `attr` so that setting can be accomplished by the pipe operator (`%>%`). Column types are strings stored in the `coltype` attribute.

Usage

```
setcoltype(a, coltype)
```

Arguments

`a`	the matrix on which column type is to be set
`coltype`	the type of item stored in columns

Details

#' If `is.null(coltype)`, the coltype attribute is deleted and subsequent calls to `coltype` will return `NULL`.

Value

`a` with `coltype` attribute set.

Examples

```
library(dplyr)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames))
U %>% setcoltype("Industries")
# This also works for lists
setcoltype(list(U,U), coltype = "Industries")
setcoltype(list(U,U), coltype = list("Industries", "Industries"))
DF <- data.frame(U = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
setcoltype(DF$U, "Industries")
DF <- DF %>% mutate(newcol = setcoltype(U, "Industries"))
DF$newcol[[1]]
DF$newcol[[2]]
```

`setrownames_byname` *Sets row names*

Description

Sets row names in a way that is amenable to use in piping operations in a functional programming way. If `a` is `NULL`, `NULL` is returned. If `a` is a constant, it is converted to a matrix and `rownames` are applied. If `a` is a matrix, `rownames` should be a vector of new row names that is as long as the number of rows in `a`. If `a` is a list of matrices, `rownames` can also be a list, and it should be as long `a`. Or `rownames` can be a vector of row names which will be applied to every matrix in the list of `a`. Each item in the list should be a vector containing row names for the corresponding matrix in `a`.

Usage

```
setrownames_byname(a, rownames)
```

Arguments

`a`	A matrix or a list of matrices in which row names are to be set
`rownames`	A vector of new row names or a list of vectors of new row names

Value

a copy of `m` with new row names

Examples

```
library(dplyr)
m <- matrix(c(1:6), nrow = 2, dimnames = list(paste0("i", 1:2), paste0("c", 1:3))) %>%
  setrowtype("Industries") %>% setcoltype("Commodities")
setrownames_byname(m, c("a", "b"))
setrownames_byname(m %>% setrowtype("Industries") %>% setcoltype("Commodities"), c("c", "d"))
m %>% setrownames_byname(NULL)
m %>% setrownames_byname(c(NA, NA))
2 %>% setrownames_byname("row")
# This also works for lists
setrownames_byname(list(m,m), list(c("a", "b")))
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
setrownames_byname(DF$m, list(c("r1", "r2")))
setrownames_byname(DF$m, list(c("c", "d")))
DF <- DF %>% mutate(m = setrownames_byname(m, list(c("r1", "r2"))))
DF$m[[1]]
```

setrowtype *Sets row type for a matrix or a list of matrices*

Description

This function is a wrapper for `attr` so that setting can be accomplished by the pipe operator (`%>%`). Row types are strings stored in the `rowtype` attribute.

Usage

```
setrowtype(a, rowtype)
```

Arguments

`a`	the matrix on which row type is to be set
`rowtype`	the type of item stored in rows

Details

If `is.null(rowtype)`, the rowtype attribute is deleted and subsequent calls to `rowtype` will return `NULL`.

Value

`a` with rowtype attribute set to `rowtype`.

Examples

```
library(dplyr)
commoditynames <- c("c1", "c2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(commoditynames, industrynames))
U %>% setrowtype("Commodities")
# This also works for lists
setrowtype(list(U,U), rowtype = "Commodities")
setrowtype(list(U,U), rowtype = list("Commodities", "Commodities"))
DF <- data.frame(U = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
setrowtype(DF$U, "Commodities")
DF <- DF %>% mutate(newcol = setrowtype(U, "Commodities"))
DF$newcol[[1]]
DF$newcol[[2]]
```

`sort_rows_cols` *Sorts rows and columns of a matrix*

Description

Checks that row names are unique and that column names are unique. Then, sorts the rows and columns in a way that ensures any other matrix with the same row and column names will have the same order.

Usage

```
sort_rows_cols(a, margin = c(1, 2), roworder = NA, colorder = NA)
```

Arguments

`a`	a matrix or data frame whose rows and columns are to be sorted
`margin`	specifies the subscript(s) in `a` over which sorting will occur. `margin` has nearly the same semantic meaning as in `apply`. For rows only, give `1`; for columns only, give `2`; for both rows and columns, give `c(1,2)`, the default value.
`roworder`	specifies the order for rows with default `sort(rownames(a))`. If `NA` (the default), default sort order is used. Unspecified rows are removed from the output, thus providing a way to delete rows from `a`. Extraneous row names (row names in `roworder` that do not appear in `a`) are ignored.
`colorder`	specifies the order for rows with default `sort(colnames(a))`. If `NA` (the default), default sort order is used. Unspecified columns are removed from the output, thus providing a way to delete columns from `a`. Extraneous column names (column names in `colorder` that do not appear in `a`) are ignored.

Details

Default sort order is given by `base::sort()` with `decreasing = FALSE`.

Value

A modified version of `a` with sorted rows and columns

Examples

```
m <- matrix(c(1:6), nrow=3, dimnames = list(c("r3", "r5", "r1"), c("c4", "c2")))
sort_rows_cols(m)
sort_rows_cols(t(m))
sort_rows_cols(m, margin=1) # Sorts rows
sort_rows_cols(m, margin=2) # Sorts columns
v <- matrix(c(1:5), ncol=1, dimnames=list(rev(paste0("r", 1:5)), "c1")) # Column vector
sort_rows_cols(v)
sort_rows_cols(v, margin = 1) # Sorts rows
sort_rows_cols(v, margin = 2) # No effect: only one column
```

```
r <- matrix(c(1:4), nrow=1, dimnames=list("r1", rev(paste0("c", 1:4)))) # Row vector
sort_rows_cols(r) # Sorts columns
n <- matrix(c(1,2), nrow = 1, dimnames = list(NULL, c("c2", "c1"))) # No row name
sort_rows_cols(n) # Sorts columns, because only one row.
# Also works with lists
sort_rows_cols(list(m,m)) # Sorts rows and columns for both m's.
# Sort rows only for first one, sort rows and columns for second one.
# Row order is applied to all m's.  Column order is natural.
sort_rows_cols(a = list(m,m), margin = 1, roworder = list(c("r5", "r3", "r1")))
# Columns are sorted as default, because no colorder is given.
# roworder is ignored.
sort_rows_cols(a = list(m,m), margin = 2, roworder = list(c("r5", "r3", "r1")))
# Both columns and rows sorted, rows by the list, columns in natural order.
sort_rows_cols(a = list(m,m), margin = c(1,2), roworder = list(c("r5", "r3", "r1")))
```

`sumall_byname` *Sum of all elements in a matrix*

Description

This function is equivalent to a `%>%` `rowsums_byname()` `%>%` `colsums_byname()`, but returns a single numeric value instead of a 1x1 matrix.

Usage

```
sumall_byname(a)
```

Arguments

a	the matrix whose elements are to be summed

Value

the sum of all elements in a as a numeric

Examples

```
library(dplyr)
m <- matrix(2, nrow=2, ncol=2, dimnames = list(paste0("i", 1:2), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
sumall_byname(m)
rowsums_byname(m) %>% colsums_byname
# Also works for lists
sumall_byname(list(m,m))
DF <- data.frame(m = I(list()))
DF[[1,"m"]] <- m
DF[[2,"m"]] <- m
sumall_byname(DF$m[[1]])
sumall_byname(DF$m)
res <- DF %>% mutate(
```

```
  sums = sumall_byname(m)
)
res$sums
```

---

sum_byname *Name-wise addition of matrices*

Description

Performs a union and sorting of addend and augend row and column names prior to summation. Zeroes are inserted for missing matrix elements. Treats missing or NULL operands as 0.

Usage

```
sum_byname(..., .summarise = FALSE)
```

Arguments

...	Operands: constants, matrices, or lists of matrices.
.summarise	When TRUE, a operands are summed down lists. When FALSE (the default), items are summed across lists.

Details

For this function, a list of lists of operands is ambiguous. Should the operands be summed across lists (first items summed across all lists, second items summed across all list, etc.) or should each list be summed along each list? In the first case, the return object will have length equal to the length of the lists in the ... argument. In the second case, the return object will have length equal to the number of lists in the ... argument. The first case is like summing across rows of a data frame. The second case is like summing down columns of a data frame. The summarise argument distinguishes between these two cases. The default value for summarise is FALSE, giving the first behavior. Set summarise to TRUE to cause this function to act like dplyr::summarise() for its list of arguments. If .summarise = TRUE, the data value is guaranteed to be a list. If the call to sum_byname(.summarise = TRUE) is made in the context of a data frame, the column returned is guaranteed to be a list column. See the aggregation vignette for additional details and examples.

Value

A matrix representing the name-wise sum of arguments.

Examples

```
library(dplyr)
sum_byname(2, 2)
sum_byname(2, 2, 2)
sum_byname(2, 2, -2, -2)
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
```

```
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>% setcoltype("Industries")
Y <- matrix(1:4, ncol = 2, dimnames = list(rev(productnames), rev(industrynames))) %>%
  setrowtype("Products") %>% setcoltype("Industries")
sum_byname(U, 100)
sum_byname(200, Y)
U + Y # Non-sensical.  Row and column names not respected.
sum_byname(U, U)
sum_byname(U, Y)
sum_byname(U, U, Y, Y)
V <- matrix(1:4, ncol = 2, dimnames = list(industrynames, productnames)) %>%
  setrowtype("Industries") %>% setcoltype("Products")
U + V # row and column names are non-sensical and blindly taken from first argument (U)
## Not run: sum_byname(U, V) # Fails, because row and column types are different
# This also works with lists
sum_byname(list(U,U), list(Y,Y))
sum_byname(list(U,U), list(100,100))
sum_byname(list(U,U), as.list(rep_len(100, 2)))
DF <- data.frame(U = I(list()), Y = I(list()))
DF[[1,"U"]] <- U
DF[[2,"U"]] <- U
DF[[1,"Y"]] <- Y
DF[[2,"Y"]] <- Y
sum_byname(DF$U, DF$Y)
DF %>% mutate(sums = sum_byname(U, Y))
sum_byname(U) # If only one argument, return it.
sum_byname(2, NULL) # Gives 2
sum_byname(2, NA)   # Gives NA
sum_byname(NULL, 1) # Gives 1
sum_byname(list(NULL, 1), list(1, 1))
DF2 <- data.frame(U = I(list()), Y = I(list()))
DF2[[1,"U"]] <- NULL
DF2[[2,"U"]] <- U
DF2[[1,"Y"]] <- Y
DF2[[2,"Y"]] <- Y
sum_byname(DF2$U, DF2$Y)
DF3 <- DF2 %>% mutate(sums = sum_byname(U, Y))
DF3
DF3$sums[[1]]
DF3$sums[[2]]
```

`switch_notation_byname` *Change row and column name notation*

Description

This function switches matrix row and/or column names from one type of notation to another based on the `from` and `to` arguments. Optionally, prefix and suffix can be `flipped`.

Usage

```
switch_notation_byname(a, margin = c(1, 2), from, to, flip = FALSE)
```

Arguments

a	A matrix or list of matrices whose row and/or column notation is to be changed.
margin	1 For rows, 2 for columns, or c(1, 2) for both rows and columns. Default is c(1, 2).
from	The notation to switch *away from*.
to	The notation to switch *to*.
flip	A boolean that tells whether to also flip the notation. Default is FALSE.

Value

Matrices with row and column names with switched notation, per arguments.

Examples

```
m <- matrix(c(1, 2,
              3, 4), nrow = 2, ncol = 2, byrow = TRUE,
            dimnames = list(c("b [a]", "d [c]"), c("f [e]", "h [g]"))) %>%
  setrowtype("Products [Industries]") %>% setcoltype("Industries [Products]")
m
switch_notation_byname(m, from = RCLabels::bracket_notation, to = RCLabels::arrow_notation,
                       flip = TRUE)
# Also works for lists.
# Note that margin must be specified as a list here.
switch_notation_byname(list(m, m), margin = list(c(1, 2)),
                       from = RCLabels::bracket_notation,
                       to = RCLabels::arrow_notation, flip = TRUE)
```

---

transpose_byname *Transpose a matrix by name*

Description

Gives the transpose of a matrix or list of matrices

Usage

```
transpose_byname(a)
```

Arguments

a	the matrix to be transposed

Value

the transposed matrix

Examples

```
m <- matrix(c(11,21,31,12,22,32), ncol = 2, dimnames = list(paste0("i", 1:3), paste0("c", 1:2))) %>%
  setrowtype("Industry") %>% setcoltype("Commodity")
transpose_byname(m)
transpose_byname(list(m,m))
```

---

trim_rows_cols *Trim rows and/or columns from a matrix*

Description

By default, the matsbyname package expends matrices with 0 rows or columns prior to matrix operations to ensure that rows and columns match. There are times when trimming rows or columns is preferred over the default behavior. This function trims rows or columns in a to match the rows or columns of mat. The return value will have rows or columns of a removed if they do not appear in mat.

Usage

```
trim_rows_cols(
  a = NULL,
  mat = NULL,
  margin = c(1, 2),
  warn_if_a_incomplete = TRUE,
  a_piece = "all",
  mat_piece = "all",
  notation = RCLabels::bracket_notation,
  prepositions = RCLabels::prepositions
)
```

Arguments

a	A matrix to be trimmed.
mat	The matrix
margin	The dimension of a to be trimmed. 1 means rows; 2 means columns. Default is c(1,2).
warn_if_a_incomplete	When TRUE (the default), a warning is emitted if a is missing entries on margin that are present in mat.
a_piece	The portion of a labels to be used for comparison. Default is "all".
mat_piece	The portion of mat labels to be used for comparison. Default is "all".

notation	The notation for row and column labels. Default is RCLabels::bracket_notation.
prepositions	The strings to be treated as prepositions in row and column labels. Default is RCLabels::prepositions.

Details

If a is NULL, NULL is returned. If mat is NULL, a is returned unmodified. If mat has NULL dimnames, a is returned unmodified. If mat has NULL for dimnames on margin, an error is returned.

A common use case for this function is to trim a, because it has too many entries on margins compared to mat. This trimming will result in a smaller result for any mathematical operations involving a and mat. Typically, a should cover all the entries in mat on margin. Thus, by default, this function warns if a is missing entries on margin that are present in mat. To turn off this checking behavior, set warn_if_a_incomplete = FALSE.

a_piece and mat_piece control which part of row and column names are compared before trimming. The default values for a_piece and mat_piece are "all", meaning that the entire label should be matched. Other options for a_piece and mat_piece are "pref" and "suff", which will match the prefix or suffix of the labels. Alternatively, prepositions can be given such that objects of prepositions will be matched. Examples include "from" or "in".

Value

Matrix a with rows or columns trimmed to match mat.

See Also

RCLabels::get_piece(), which is used internally.

Examples

```
a <- matrix(c(1, 2, 3,
              4, 5, 6,
              7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE,
            dimnames = list(c("r1", "r2", "r3"), c("c1", "c2", "c3"))) %>%
 setrowtype("rowtype") %>% setcoltype("coltype")
mat <- matrix(c(1, 2, 3,
                4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE,
            dimnames = list(c("r1", "bogus"), c("c1", "bogus", "c2"))) %>%
 setrowtype("rowtype") %>% setcoltype("coltype")
trim_rows_cols(a, mat, margin = 1)
trim_rows_cols(a, mat, margin = 2)
trim_rows_cols(a, mat)
```

unaryapply_byname *Apply a unary function by name*

Description

FUN is applied to a using additional arguments .FUNdots to FUN. If a is a list, the names of a are applied to the output.

Usage

```
unaryapply_byname(
  FUN,
  a,
  .FUNdots = NULL,
  rowcoltypes = c("all", "transpose", "row", "col", "none")
)
```

Arguments

FUN	a unary function to be applied "by name" to a.
a	the argument to FUN.
.FUNdots	a list of additional named arguments passed to FUN.
rowcoltypes	a string that tells how to transfer row and column types of a to output. See details.

Details

Note that .FUNdots can be a rectangular two-dimensional list of arguments to FUN. If so, .FUNdots is interpreted as follows:

- The first dimension of .FUNdots contains named arguments to FUN.
- The second dimension of .FUNdots contains unique values of the named arguments to be applied along the list that is a.

The length of the first dimension of .FUNdots is the number of arguments supplied to FUN. The length of the second dimension of .FUNdots must be equal to the length of a.

See prepare_.FUNdots() for more details on the .FUNdots argument.

Options for the rowcoltypes argument are:

- "all": transfer both row and column types of a directly to output.
- "transpose": rowtype of a becomes coltype of output; coltype of a becomes rowtype of output. "transpose" is helpful for FUNs that transpose a upon output.
- "row": rowtype of a becomes both rowtype and coltype of output.
- "col": coltype of a becomes both rowtype and coltype of output.
- "none": rowtype and coltype not set by unaryapply_byname. Rather, FUN will set rowtype and coltype.

Note that rowcoltypes should not be a vector or list of strings. Rather, it should be a single string.

Value

the result of applying FUN "by name" to a.

Examples

```
productnames <- c("p1", "p2")
industrynames <- c("i1", "i2")
U <- matrix(1:4, ncol = 2, dimnames = list(productnames, industrynames)) %>%
  setrowtype("Products") %>% setcoltype("Industries")
difference_byname(0, U)
unaryapply_byname(`-`, U)
```

---

vectorize_byname *Vectorize a matrix*

Description

Converts a matrix into a column vector. Each element of the matrix becomes an entry in the column vector, with rows named via the notation argument. Callers may want to transpose the matrix first with transpose_byname().

Usage

```
vectorize_byname(a, notation)
```

Arguments

a	the matrix to be vectorized.
notation	a string vector created by notation_vec().

Details

The notation argument is also applied to rowtype and coltype attributes.

Value

a column vector containing all elements of a, with row names assigned as "rowname sep colname".

Examples

```
m <- matrix(c(1, 5,
              4, 5),
            nrow = 2, ncol = 2, byrow = TRUE,
            dimnames = list(c("p1", "p2"), c("i1", "i2"))) %>%
  setrowtype("Products") %>% setcoltype("Industries")
m
vectorize_byname(m, notation = RCLabels::arrow_notation)
# If a single number is provided, the number will be returned as a 1x1 column vector
```

```
# with some additional attributes.
vectorize_byname(42, notation = RCLabels::arrow_notation)
attributes(vectorize_byname(42, notation = RCLabels::arrow_notation))
```

`vec_from_store_byname` *Create a vector with labels from a matrix and values from a vector store*

Description

When a matrix is multiplied by a vector byname, naming can be tricky. There are times when pieces of the vector labels should be matched to pieces of the matrix labels. This function helps by performing the matching byname. For this function, vector `v` is considered a store of values from which the output vector is created using special matching rules between `a` and `v`.

Usage

```
vec_from_store_byname(
  a,
  v,
  a_piece = "all",
  v_piece = "all",
  colname = NULL,
  column = TRUE,
  notation = if (is.list(a)) {     list(RCLabels::bracket_notation) } else {
    RCLabels::bracket_notation },
  prepositions = if (is.list(a)) {     list(RCLabels::prepositions) } else {
    RCLabels::prepositions }
)
```

Arguments

`a`	A matrix from which row or column labels are taken. Can also be a list or the name of a column in a data frame.
`v`	A vector from which values are taken, when `a_piece` matches `v_piece`. Can also be a list or the name of a column in a data frame.
`a_piece`	The piece of labels on `a` that is to be matched. Default is "all".
`v_piece`	The piece of labels on `v` that is to be matched. Default is "all".
`colname`	The name of the output vector's 1-sized dimension (the only column if `column` is `TRUE`, the only row otherwise). Default is `NULL`, meaning that the name of the 1-sized dimension in `v` should be used.
`column`	Tells whether a column vector (if `TRUE`, the default) or a row vector (if `FALSE`) should be created.
`notation`	The notation for the row and column labels. Default is `RCLabels::bracket_notation`, wrapped as a list if `a` is a list.
`prepositions`	The strings that will count for prepositions. Default is `RCLables::prepositions`, wrapped as a list if `a` is a list..

Details

The output of this function is a vector (a column vector if column is TRUE, the default; a row vector if column is FALSE). The label of the size = 1 dimension is taken from colname (so named, because the default is to return a column vector). The labels of the long dimension are taken from matrix a (the row names of a if column is TRUE; the column names of a if column is FALSE). The values of the output vector are obtained from v when a_piece matches v_piece using the RCLabels package. The v_pieces of v must be unique. The default values for a_piece and v_piece are "all", meaning that the entire label should be matched. Other options for a_piece and v_piece are "pref" and "suff", which will match the prefix or suffix of the labels. Alternatively, prepositions can be given such that objects of prepositions will be matched. Examples include "from" or "in". Row and column types from v are applied to the output. If the piece given in a_piece is not present in row or column names of a, NA_real_ is returned. If the piece given in v_piece is not present in row or column names of v, NA_real_ is returned.

Note that notation and prepositions should be lists if a is a list but a single value otherwise. The default values of notation and prepositions take care of this requirement, switching on the type of a (list or not).

Value

A vector with names from a and values from v.

Examples

```
a <- matrix(42, nrow = 3, ncol = 5,
            dimnames = list(c("Electricity [from b in c]",
                              "Coal [from e in f]",
                              "Crude oil [from Production in USA]"),
                            c("Main activity producer electricity plants",
                              "Wind turbines",
                              "Oil refineries",
                              "Coal mines",
                              "Automobiles"))) %>%
  setrowtype("Product") %>% setcoltype("Industry")
a
v <- matrix(1:7, nrow = 7, ncol = 1,
            dimnames = list(c("Electricity",
                              "Peat",
                              "Hydro",
                              "Crude oil",
                              "Coal",
                              "Hard coal (if no detail)",
                              "Brown coal"),
                            "phi")) %>%
  setrowtype("Product") %>% setcoltype("phi")
v
vec_from_store_byname(a, v, a_piece = "pref")
vec_from_store_byname(a, v, a_piece = "noun")

v2 <- matrix(1:7, nrow = 7, ncol = 1,
             dimnames = list(c("Electricity",
```

```
                                "Peat",
                                "USA",
                                "c",
                                "Coal",
                                "Hard coal (if no detail)",
                                "f"),
                              "phi")) %>%
  setrowtype("Product") %>% setcoltype("phi")
vec_from_store_byname(a, v2, a_piece = "in")

# Works with lists
v3 <- matrix(1:7, nrow = 7, ncol = 1,
             dimnames = list(c("Electricity [from USA]",
                               "Peat [from nowhere]",
                               "Production [from GHA]",
                               "e [from ZAF]",
                               "Coal [from AUS]",
                               "Hard coal (if no detail) [from GBR]",
                               "b [from Nebraska]"),
                             "phi")) %>%
  setrowtype("Product") %>% setcoltype("phi")
a_list <- list(a, a)
v_list <- list(v3, v3)
vec_from_store_byname(a_list, v_list, a_piece = "in", v_piece = "from")

# Also works in a data frame
df <- tibble::tibble(a = list(a, a, a),
                     v = list(v3, v3, v3))
df %>%
  dplyr::mutate(
    actual = vec_from_store_byname(a = a, v = v, a_piece = "in", v_piece = "from")
  )
```

Index