# Package 'SSBtools'

May 12, 2026

**Type** Package

**Title** Algorithms and Tools for Tabular Statistics and Hierarchical Computations

**Version** 1.8.7

**Date** 2026-05-12

**Imports** methods, MASS, Matrix

**Description** Includes general data manipulation functions, algorithms for statistical disclosure control (Langsrud, 2024) <doi:10.1007/978-3-031-69651-0_6> and functions for hierarchical computations by sparse model matrices (Langsrud, 2023) <doi:10.32614/RJ-2023-088>.

**License** MIT + file LICENSE

**URL** https://github.com/statisticsnorway/ssb-ssbtools, https://statisticsnorway.github.io/ssb-ssbtools/

**BugReports** https://github.com/statisticsnorway/ssb-ssbtools/issues

**Encoding** UTF-8

**Suggests** testthat, data.table, bit64

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Øyvind Langsrud [aut, cre] (ORCID: <https://orcid.org/0000-0002-1380-4396>), Daniel Lupp [aut] (ORCID: <https://orcid.org/0000-0003-3575-1691>), Bjørn-Helge Mevik [ctb], Vidar Norstein Klungre [rev] (ORCID: <https://orcid.org/0000-0003-1925-5911>), Statistics Norway [cph]

**Maintainer** Øyvind Langsrud <oyl@ssb.no>

**Repository** CRAN

**Date/Publication** 2026-05-12 13:50:12 UTC

# Contents

---

`AddLeadingZeros` *Add leading zeros to numbers while preserving other text*

---

## Description

This function is created to fix problems caused by a serious bug in Excel. Editing csv files in that program causes leading zeros to disappear.

## Usage

```
AddLeadingZeros(
  codes,
  places,
  warningText = NULL,
  viaFactor = TRUE,
  nWarning = 6,
  removeLeadingTrailingWhitespace = TRUE
)
```

### Arguments

| | |
|---|---|
| codes | Character vector |
| places | Number of places for positive numbers. Minus sign is extra |
| warningText | When non-NULL, warning will be produced |
| viaFactor | When TRUE, the algorithm uses factor coding internally. |
| nWarning | Number of elements to be written before ... in warnings. |
| removeLeadingTrailingWhitespace | Remove leading and trailing whitespace |

### Value

Character vector

### Author(s)

Øyvind Langsrud

### Examples

```
AddLeadingZeros(c("1", "ABC", "12345", " 23", "-8", "45 ", " -9", " Agent ", "007",
                  "7 James Bond "), 10)
AddLeadingZeros(c("1", "ABC", "12345", " 23", "-8", "45 ", " -9", " Agent ", "007",
                  "7 James Bond "), 4)
AddLeadingZeros(c("1", "ABC", "12345", " 23", "-8", "45 ", " -9", " Agent ", "007",
                  "7 James Bond "), 4, removeLeadingTrailingWhitespace = FALSE)
AddLeadingZeros(c("1", "ABC", "12345", " 23", "-8", "45 ", " -9", " Agent ", "007",
                  "7 James Bond "), 4, warningText = "string changes")
AddLeadingZeros(c("1", "ABC", "12345", " 23", "-8", "45 ", " -9", " Agent ", "007",
                  "7 James Bond "), 4, warningText = "", nWarning = 2)
```

---

## aggregate_by_pkg *Aggregate by base R or data.table*

---

### Description

This function aggregates data by specified grouping variables, using either base R or `data.table`.

### Usage

```
aggregate_by_pkg(
  data,
  by,
  var,
  pkg = "base",
  include_na = FALSE,
  fun = sum,
  base_order = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data frame |
| by | A character vector specifying the column names to group by. |
| var | A character vector specifying the column names of the variables to be aggregated. |
| pkg | A character string indicating which package to use for aggregation. Must be either "base" for base R or "data.table" for data.table. Default is "base". |
| include_na | A logical value indicating whether NA values in the grouping variables should be included in the aggregation. Default is FALSE. |
| fun | The function to be applied for aggregation. Default is sum. |
| base_order | A logical value indicating whether to attempt to return the results in the same order as base R when using data.table. Note that while the function strives to maintain this order, it cannot be guaranteed due to potential variations in sorting behavior across different systems. Default is TRUE. |
| ... | Further arguments passed to aggregate when pkg is "base" |

## Value

A data.frame containing the aggregated results.

## Examples

```
d <- SSBtoolsData("d2")[1:20, ]
d[[2]] <- as.numeric(d[[2]])
d$y <- as.numeric(1:20)
d$y[2] <- NA
d$county[8:9] <- NA
d$main_income[11:12] <- NA
d$k_group[19:20] <- NA
by <- c("main_income", "county", "k_group")

a1 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"))
a2 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"),
                       include_na = TRUE)
a3 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"),
                       include_na = TRUE, fun = function(x) list(x))

if (requireNamespace("data.table", quietly = TRUE)) {

  b1 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"), pkg = "data.table")
  b2 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"), pkg = "data.table",
                         include_na = TRUE)
  b3 <- aggregate_by_pkg(d, by = by, var = c("y", "freq"), pkg = "data.table",
                         include_na = TRUE, fun = function(x) list(x))

  print(identical(a1, b1))   # TRUE when base_order succeeds
  print(identical(a2, b2))
  print(identical(a3, b3))
```

```
} else {
   print("The 'data.table' package is not installed.")
}
```

---

`aggregate_multiple_fun` *Wrapper to* `aggregate`

---

## Description

Wrapper to `aggregate` that allows multiple functions and functions of several variables

## Usage

```
aggregate_multiple_fun(
  data,
  by,
  vars,
  fun = NULL,
  ind = NULL,
  ...,
  name_sep = "_",
  seve_sep = ":",
  multi_sep = ",",
  forward_dots = FALSE,
  dots2dots = FALSE,
  do_unmatrix = TRUE,
  do_unlist = TRUE,
  inc_progress = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing data to be aggregated |
| `by` | A data frame defining grouping |
| `vars` | A named vector or list of variable names in `data`. The elements are named by the names of `fun`. All the pairs of variable names and function names thus define all the result variables to be generated.<br>• Parameter `vars` will converted to an internal standard by the function `fix_vars_amf`. Thus, function names and also output variable names can be coded in different ways. Multiple output variable names can be coded using `multi_sep`. See examples and examples in `fix_vars_amf`. Indices instead of variable names are allowed.<br>• Omission of (some) names is possible since names can be omitted for one function (see `fun` below). |

- A special possible feature is the combination of a single unnamed variable and all functions named. In this case, all functions are run and output variable names will be identical to the function names.

`fun` A named list of functions. These names will be used as suffixes in output variable names. Name can be omitted for one function. A vector of function as strings is also possible. When unnamed, these function names will be used directly. See the examples of `fix_fun_amf`, which is the function used to convert `fun`. Without specifying `fun`, the functions, as strings, are taken from the function names coded in `vars`.

`ind` When non-NULL, a data frame of indices. When NULL, this variable will be generated internally as `data.frame(ind = seq_len(nrow(data)))`. The parameter is useful for advanced use involving model/dummy matrices. For special use (`dummy = FALSE` in `dummy_aggregate`) `ind` can also be a two-column data frame.

`...` Further arguments passed to `aggregate` and, depending on `forward_dots`/`dots2dots`, forwarded to the functions in `fun` (see details).

`name_sep` A character string used when output variable names are generated.

`seve_sep` A character string used when output variable names are generated from functions of several variables.

`multi_sep` A character string used when multiple output variable names are sent as input.

`forward_dots` Logical vector (possibly recycled) for each element of `fun` that determines whether `...` should be forwarded (see details).

`dots2dots` Logical vector (possibly recycled) specifying the behavior when `forward_dots = TRUE` (see details).

`do_unmatrix` By default (`TRUE`), the implementation uses `unmatrix` before returning output. For special use this can be omitted (`FALSE`).

`do_unlist` By default (`TRUE`), the implementation uses `unlist` to combine output from multiple functions. For special use this can be omitted (`FALSE`).

`inc_progress` logigal, `NULL` (same as `FALSE`) or a progress indicator function taking two parameters (i and n). `TRUE` means the same as `inc_default`. Note that this feature is implemented in a hacky manner as internal/hidden variables are grabbed from `aggregate`.

## Details

One intention of `aggregate_multiple_fun` is to be a true generalization of `aggregate`. However, when many functions are involved, passing extra parameters can easily lead to errors. Therefore `forward_dots` and `dots2dots` are set to `FALSE` by default. When `forward_dots = TRUE` and `dots2dots = FALSE`, parameters will be forwarded, but only parameters that are explicitly defined in the specific `fun` function. For the `sum` function, this means that a possible `na.rm` parameter is forwarded but not others. When `forward_dots = TRUE` and `dots2dots = TRUE`, other parameters will also be forwarded to `fun` functions where `...` is included. For the `sum` function, this means that such extra parameters will, probably erroneously, be included in the summation (see examples).

For the function to work with `dummy_aggregate`, the data is subject to `unlist` before the `fun` functions are called. This does not apply in the special case where `ind` is a two-column data frame. Then, in the case of list data, the `fun` functions have to handle this themselves.

A limitation when default output, when do_unlist = TRUE, is that variables in output are forced to have the same class. This is caused by the unlist function being run on the output. This means, for example, that all the variables will become numeric when they should have been both integer and numeric.

## Value

A data frame

## Examples

```
d2 <- SSBtoolsData("d2")
set.seed(12)
d2$y <- round(rnorm(nrow(d2)), 2)
d <- d2[sample.int(nrow(d2), size = 20), ]
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
   vars = c("freq", "y", median = "freq", median = "y", e1 = "freq"),
   fun = c(sum, median = median, e1 = function(x) x[1])
)

# With functions as named strings
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
   vars = c(sum = "y", med = "freq", med = "y"),
   fun = c(sum = "sum", med = "median")
)

# Without specifying functions
# - equivalent to `fun = c("sum", "median")`
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
   vars = c(sum = "y", median = "freq", median = "y")
)

# The single unnamed variable feature. Also functions as strings.
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
   vars = "y",
   fun = c("sum", "median", "min", "max")
)

# with multiple outputs (function my_range)
# and with function of two variables (weighted.mean(y, freq))
my_range <- function(x) c(min = min(x), max = max(x))
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
```

```
   vars = list("freq", "y", ra = "freq", wmean  = c("y", "freq")),
   fun = c(sum, ra = my_range, wmean = weighted.mean)
)

# with specified output variable names
my_range <- function(x) c(min = min(x), max = max(x))
aggregate_multiple_fun(
   data = d,
   by = d[c("k_group", "main_income")],
   vars = list("freq", "y",
               `freqmin,freqmax` = list(ra = "freq"),
                yWmean  = list(wmean  = c("y", "freq"))),
   fun = c(sum, ra = my_range, wmean = weighted.mean)
)


# To illustrate forward_dots and dots2dots
q <- d[1, ]
q$w <- 100 * rnorm(1)
for (dots2dots in c(FALSE, TRUE)) for (forward_dots in c(FALSE, TRUE)) {
  cat("\n=====================================\n")
  cat("forward_dots =", forward_dots, ", dots2dots =", dots2dots)
  out <- aggregate_multiple_fun(
    data = q, by = q["k_group"],
    vars = c(sum = "freq", round = "w"), fun = c("sum", "round"),
    digits = 3, forward_dots = forward_dots, dots2dots = dots2dots)
  cat("\n")
  print(out)
}
# In last case digits forwarded to sum (as ...)
# and wrongly included in the summation
```

---

## any_duplicated_rows *Fast alternative to* anyDuplicated()

---

### Description

Implemented similarly to RowGroups().

### Usage

```
any_duplicated_rows(x, cols = names(x))
```

### Arguments

| | |
|---|---|
| x | A data frame, tibble, or data.table. |
| cols | Columns to check for duplicates. |

## Details

With `data.table` input and the data.table package available, `anyDuplicated.data.table()` will be used.

## Value

Index of the first duplicate row, if any; otherwise 0.

## Examples

```
z <- SSBtoolsData("power10to2")
head(z, 12)
tail(z)

any_duplicated_rows(z, c("A", "B"))
any_duplicated_rows(z, c("a", "A", "B"))
any_duplicated_rows(z, c("a", "A", "b"))
```

---

# As_TsparseMatrix *Transform to TsparseMatrix/dgTMatrix*

---

## Description

To implement adaption needed after Matrix ver. 1.4-2 since `as(from, "dgTMatrix")` no longer allowed.

## Usage

```
As_TsparseMatrix(from, do_drop0 = TRUE)
```

## Arguments

| | |
|---|---|
| `from` | A matrix |
| `do_drop0` | whether to run `drop0` |

## Details

This function is made to replace `as(from, "dgTMatrix")` and `as(drop0(from), "dgTMatrix")` in `SSBtools` and related packages.

## Value

A matrix. Virtual class is `TsparseMatrix`. Class `dgTMatrix` expected.

## Note

`Matrix:::.as.via.virtual` in development version of package `Matrix` (date 2022-08-13) used to generate code.

`AutoHierarchies` *Ensure standardized coding of hierarchies*

## Description

Automatic convert list of hierarchies coded in different ways to standardized to-from coding

## Usage

```
AutoHierarchies(
  hierarchies,
  data = NULL,
  total = "Total",
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level =
    "level"),
  combineHierarchies = TRUE,
  unionComplement = FALSE,
  autoLevel = TRUE,
 autoNames = c(to = "from", parentCode = "code", parent = "child", root = "leaf"),
  ...
)
```

## Arguments

| | |
|---|---|
| `hierarchies` | List of hierarchies |
| `data` | Matrix or data frame with data containing codes of relevant variables |
| `total` | Within `AutoHierarchies`: Vector of total codes (possibly recycled) used when running `Hrc2DimList` or `FindDimLists`. If a named vector or named list is provided, names are matched against `names(hierarchies)`, and matching entries are applied to the corresponding hierarchies. Hierarchies without a matching name use the recycled default value. |
| `hierarchyVarNames` | Variable names in the hierarchy tables as in `HierarchyFix`. However:<br>• `level` is by default not required (see `autoLevel` below).<br>• If the `sign` variable is missing, it defaults to a variable of 1s.<br>• Common 'from-to' variable names are recognized (see `autoNames` below). |
| `combineHierarchies` | Whether to combine several hierarchies for same variable into a single hierarchy (see examples). |
| `unionComplement` | Logical vector as in `Hierarchies2ModelMatrix`. The parameter is only in use when hierarchies are combined. |
| `autoLevel` | When TRUE (default), the level is computed automatically, ignoring the input level variable. This parameter is passed to `HierarchyFix`.. |

| | |
|---|---|
| autoNames | Named character vector of 'from-to' variable names to be automatically recognized. These names do not need to be specified in hierarchyVarNames. Thus, autoNames can serve as an alternative to hierarchyVarNames. |
| ... | Extra unused parameters |

## Details

Input can be to-from coded hierarchies, hierarchies/dimList as in sdcTable, TauArgus coded hierarchies or formulas. Automatic coding from data is also supported. Output is on a from ready for input to HierarchyCompute. A single string as hierarchy input is assumed to be a total code. Then, the hierarchy is created as a simple hierarchy where all codes in data sum up to this total. For consistence with HierarchyCompute, the codes "rowFactor" and "colFactor" are unchanged. An empty string is recoded to "rowFactor".

A special possibility is to include character vector(s) as unnamed list element(s) of hierarchies. Then the elements of the character vector(s) must be variable names within data. This will cause hierarchies to be created from selected data columns by running FindDimLists. Total coded can be specified by parameter total or by naming the character vector. See examples.

## Value

List of hierarchies

## Author(s)

Øyvind Langsrud

## See Also

FindHierarchies, DimList2Hierarchy, DimList2Hrc, Hierarchy2Formula, DummyHierarchies.

## Examples

```
# First, create different types of input
z <- SSBtoolsData("sprt_emp_withEU")
yearFormula <- c("y_14 = 2014", "y_15_16 = y_all - y_14", "y_all = 2014 + 2015 + 2016")
yearHier <- Formula2Hierarchy(yearFormula)
geoDimList <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]
geoDimList2 <- FindDimLists(z[, c("geo", "eu")])[[1]]
geoHrc <- DimList2Hrc(geoDimList)
ageHier <- SSBtoolsData("sprt_emp_ageHier")

h1 <- AutoHierarchies(list(age = ageHier, geo = geoDimList, year = yearFormula))
h2 <- AutoHierarchies(list(age = "Y15-64", geo = geoHrc, year = yearHier), data = z,
                      total = "Europe")
h3 <- AutoHierarchies(list(age = "Total", geo = geoDimList2, year = "Total"), data = z)
h4 <- FindHierarchies(z[, c(1, 2, 3, 5)])
h5 <- AutoHierarchies(list(age = "Total", geo = "", year = "colFactor"), data = z)
identical(h1, h2)
identical(h3, h4)

# Print the resulting hierarchies
```

```
h1 # = h2
h3 # = h4
h5

FindHierarchies(z[, c("geo", "eu", "age")])


# ========================================================================
#   Examples illustrating the combineHierarchies parameter
# ========================================================================

# First, create data
d <- SSBtoolsData("d2ws")[1:3]
d$isCounty1 <- "NO"
d$isCounty1[d$county == "county-1"] <- "YES"
d

# sdcTable coding showing two tree-shaped hierarchies
dimList <- FindDimLists(d)
dimList

# Two tree-shaped hierarchies can still be seen
# Hierarchies with three and two levels
hA <- AutoHierarchies(dimList, combineHierarchies = FALSE)
hA

# A single hierarchy with only one level
# Contains the information needed to create a dummy matrix
hB <- AutoHierarchies(dimList)
hB

# Dummy matrices from the hierarchies
DummyHierarchies(hA)
DummyHierarchies(hB)


# ========================================================================
#   Special examples with character vector(s) as unnamed list elements
# ========================================================================

# Same output as FindHierarchies above
AutoHierarchies(list(c("geo", "eu", "age")), data = z)

# Now combined with a named list element
AutoHierarchies(list(year = yearHier, c("geo", "eu", "age")), data = z)

# Total codes by unnamed list element as named character vector
AutoHierarchies(list(year = yearHier, c(Europe = "geo", "eu", All = "age")), data = z)

# Two types of year input. Total codes by using the parameter `total`.
AutoHierarchies(list("year", year = yearHier, c("geo", "eu", "age")), data = z,
                total = c("allYears", "unused", "Tot"))
```

```
# Avoid combineHierarchies to see effect of each year input separately
# (even earlier return possible with `combineHierarchies = NA`)
AutoHierarchies(list("year", year = yearHier, c("geo", "eu", "age")), data = z,
                total = c("allYears", "unused", "Tot"), combineHierarchies = FALSE)
```

---

`AutoSplit` *Creating variables by splitting the elements of a character vector without needing a split string*

---

## Description

Creating variables by splitting the elements of a character vector without needing a split string

## Usage

```
AutoSplit(
  s,
  split = NULL,
  border = "_",
  revBorder = FALSE,
  noSplit = FALSE,
  varNames = paste("var", 1:100, sep = ""),
  tryReverse = TRUE
)
```

## Arguments

| | |
|---|---|
| `s` | The character vector |
| `split` | Split string. When NULL (default), automatic splitting without a split string. |
| `border` | A split character or an integer (move split) to be used when the exact split position is not unique. |
| `revBorder` | When border is integer the split position is moved from the other side. |
| `noSplit` | No splitting when TRUE. |
| `varNames` | Variable names of the created variables (too many is ok) |
| `tryReverse` | When TRUE, the automatic method tries to find more variables by splitting from reversed strings. |

## Value

A data frame with s as row names.

## Author(s)

Øyvind Langsrud

## Examples

```
s <- c("A12-3-A-x","A12-3-B-x","B12-3-A-x","B12-3-B-x",
       "A12-3-A-y","A12-3-B-y","B12-3-A-y","B12-3-B-y")
AutoSplit(s)
AutoSplit(s,border="-")
AutoSplit(s,split="-")
AutoSplit(s,border=1)
AutoSplit(s,border=2)
AutoSplit(s,border=2,revBorder=TRUE)
AutoSplit(s,noSplit=TRUE)
AutoSplit(s,varNames=c("A","B","C","D"))
```

---

# CbindIdMatch *Combine several data frames by using id variables to match rows*

---

## Description

Combine several data frames by using id variables to match rows

## Usage

```
CbindIdMatch(
  ...,
  addName = names(x),
  sep = "_",
  idNames = sapply(x, function(x) names(x)[1]),
  idNames1 = idNames,
  addLast = FALSE
)
```

## Arguments

| | |
|---|---|
| ... | Several data frames as several input parameters or a list of data frames |
| addName | NULL or vector of strings used to name columns according to origin frame |
| sep | A character string to separate when addName apply |
| idNames | Names of a id variable within each data frame |
| idNames1 | Names of variables in first data frame that correspond to the id variable within each data frame |
| addLast | When TRUE addName will be at end |

## Details

The first data frame is the basis and the other frames will be matched by using id-variables. The default id-variables are the first variable in each frame. Corresponding variables with the same name in first frame is assumed. An id-variable is not needed if the number of rows is one or the same as the first frame. Then the element of idNames can be set to a string with zero length.

## Value

A single data frame

## Author(s)

Øyvind Langsrud

## See Also

RbindAll (same example data)

## Examples

```
zA <- data.frame(idA = 1:10, idB = rep(10 * (1:5), 2), idC = rep(c(100, 200), 5),
                 idC2 = c(100, rep(200, 9)), idC3 = rep(100, 10),
                 idD = 99, x = round(rnorm(10), 3), xA = round(runif(10), 2))
zB <- data.frame(idB = 10 * (1:5), x = round(rnorm(5), 3), xB = round(runif(5), 2))
zC <- data.frame(idC = c(100, 200), x = round(rnorm(2), 3), xC = round(runif(2), 2))
zD <- data.frame(idD = 99, x = round(rnorm(1), 3), xD = round(runif(1), 2))
CbindIdMatch(zA, zB, zC, zD)
CbindIdMatch(a = zA, b = zB, c = zC, d = zD, idNames = c("", "idB", "idC", ""))
CbindIdMatch(a = zA, b = zB, c = zC, d = zD, idNames1 = c("", "idB", "idC2", ""))
CbindIdMatch(a = zA, b = zB, c = zC, d = zD, idNames1 = c("", "idB", "idC3", ""))
CbindIdMatch(zA, zB, zC, zD, addName = c("", "bbb", "ccc", "ddd"), sep = ".", addLast = TRUE)
try(CbindIdMatch(X = zA, Y = zA[, 4:5], Z = zC, idNames = NULL)) # Error
CbindIdMatch(X = zA, Y = zA[, 4:5], Z = zD, idNames = NULL)    # Ok since equal NROW or NROW==1
CbindIdMatch(list(a = zA, b = zB, c = zC, d = zD))             # List is alternative input
```

---

# CheckInput *Checking function inputs*

---

## Description

An input vector (of length one unless okSeveral is TRUE) is checked.

## Usage

```
CheckInput(
  x,
  alt = NULL,
  min = NULL,
  max = NULL,
  type = "character",
  data = NULL,
  okSeveral = FALSE,
  okNULL = FALSE,
  okNA = FALSE,
  okDuplicates = is.null(alt) & !(type %in% c("varName", "varNr", "varNrName"))
```

```
)

check_input(
  x,
  alt = NULL,
  min = NULL,
  max = NULL,
  type = "character",
  data = NULL,
  okSeveral = FALSE,
  okNULL = FALSE,
  okNA = FALSE,
  okDuplicates = is.null(alt) & !(type %in% c("varName", "varNr", "varNrName"))
)
```

## Arguments

| | |
|---|---|
| x | Input vector to be checked |
| alt | NULL or vector of allowed values |
| min | NULL or minimum value (when type is numeric or integer) |
| max | NULL or maximum value (when type is numeric or integer) |
| type | One of: "character", "numeric", "integer", "logical", "varName", "varNr", "varNrName". numeric/integer is not checked against exact class, but whether the value fit into the class. Also see data below. |
| data | A data frame or matrix. When above type is varNames, x is checked against colnames(data). When type is varNr, x is checked against column numbers. When type is varNrName, x can be either column numbers or column names. |
| okSeveral | When TRUE, length(x)>1 is allowed |
| okNULL | When TRUE, NULL is allowed |
| okNA | When TRUE, NA is allowed |
| okDuplicates | When TRUE, duplicated values are allowed. Default is TRUE if alt is NULL and if type does not refer to column(s) of data. |

## Details

x is checked according to the other input parameters. When x is wrong an error is produced with appropriate text.

*The function was originally created in 2016 and has been included in internal packages at Statistics Norway (SSB). Due to its widespread use, it was beneficial to include it in this CRAN package.*

## Note

check_input and CheckInput are identical

## Author(s)

Øyvind Langsrud

## Examples

```
a <- c("no", "yes")
b <- c(3.14, 4, 5)
z <- data.frame(A = a, B = b[1:2], C = TRUE)

# Lines causing error are embedded in 'try'

try(CheckInput(a, type = "character"))
CheckInput(a, type = "character", alt = c("no", "yes", "dontknow"), okSeveral = TRUE)
try(CheckInput("yesno", type = "character", alt = c("no", "yes", "dontknow")))
CheckInput(a[1], type = "character", alt = c("no", "yes", "dontknow"))

try(CheckInput(b, type = "integer", max = 100, okSeveral = TRUE))
try(CheckInput(b, type = "numeric", min = 4, okSeveral = TRUE))
CheckInput(b, type = "numeric", max = 100, okSeveral = TRUE)
try(CheckInput(b, type = "numeric", alt = 1:10, okSeveral = TRUE))
CheckInput(b[2], type = "numeric", alt = 1:10)

try(CheckInput("TRUE", type = "logical"))
CheckInput(TRUE, type = "logical")

try(CheckInput("A", type = "varName"))
CheckInput("A", type = "varName", data = z)
CheckInput(c("A", "B"), type = "varNrName", data = z, okSeveral = TRUE)
try(CheckInput("ABC", type = "varNrName", data = z))
try(CheckInput(5, type = "varNrName", data = z))
CheckInput(3, type = "varNr", data = z)
CheckInput(2:3, type = "varNr", data = z, okSeveral = TRUE)
```

---

# convert_integer64 *Convert* integer64 *data to base R numeric, integer, or character types*

---

## Description

Converts integer64 data (from the **bit64** package) in a data.frame, list, or vector to base R numeric, integer, or character types.

## Usage

```
convert_integer64(
  df,
  to_integer = "if_fits",
  precision_loss = "character",
  always_character = FALSE
)
```

## Arguments

| | |
|---|---|
| df | A data.frame, list, or vector to process. |
| to_integer | Character string controlling how conversion to integer is handled. The rule is applied **variable by variable**. Must be one of:<br>• "never" — always convert to numeric, never to integer<br>• "if_fits" — convert to integer if all values fit within 32-bit range *(default)*<br>• "if_summable" — convert to integer if the sum of absolute values fits within 32-bit range<br>• "always" — always convert to integer, with potential coercion warnings<br>• "always_quiet" — always convert to integer, suppressing coercion warnings |
| precision_loss | Character string controlling what happens when 64-bit integers cannot be represented exactly as 64-bit floating-point numbers. Must be one of:<br>• "character" — convert to character if the value cannot be represented exactly *(default)*<br>• "warn" — convert to numeric and allow warnings about precision loss<br>• "quiet" — convert to numeric but suppress such warnings |
| always_character | Logical. If TRUE, all integer64 values are converted directly to character, overriding both to_integer and precision_loss. Default is FALSE. |

## Details

Variables with class integer64 often appear when reading data from Arrow files, for example using arrow::read_parquet(). Arrow supports 64-bit integer values, while the R language (and thus all R packages, including the tidyverse) only supports 32-bit integers and 64-bit floating-point numbers. These 64-bit integers therefore need conversion when loaded into R.

When the input is a data.frame or list, conversion is performed **variable by variable**, and only those with class integer64 are modified.

Depending on settings, integer64 data are converted to base R integer, numeric (double), or character.

Note that a simpler helper that always converts directly to numeric, without any checks or dependency tests, can be defined as:

```
convert_integer64_to_numeric <- function(df) {
  df[] <- lapply(df, function(x) {
    if (inherits(x, "integer64")) as.numeric(x) else x
  })
  df
}
```

## Value

The same type of object as the input (data.frame, list, or vector), with all integer64 values converted to base R integer, numeric, or character depending on settings.

## Note

This function is written and documented with help from ChatGPT.

## See Also

bit64::as.integer64()

## Examples

```
if (requireNamespace("bit64", quietly = TRUE)) {
  x  <- bit64::seq.integer64(2025, 10^9, 3 * 10^8)
  print(x)

  print(convert_integer64(x*4, "always_quiet"))

  df <- data.frame(a = 11:14, b = x, c = 2 * x, d = 3 * x, e = x * x, f = c(22, 23, 24, 25))
  print(df)

  df1 <- convert_integer64(df, "never")
  df2 <- convert_integer64(df, "if_fits")
  df3 <- convert_integer64(df, "if_summable")
  df4 <- convert_integer64(df, "always_quiet")

  print(sapply(df,  class))
  print(sapply(df1, class))
  print(sapply(df2, class))
  print(sapply(df3, class))
  print(sapply(df4, class))

  print(df2)
  print(df4)

  cat("# Examples showing that integer64 is problematic:\n")
      y <- bit64::seq.integer64(1, 3)
      print(y)
      print(0.5 * y)
      print(y * 0.5)
      matrix(y, 1, 3)
}
```

---

`DataDummyHierarchy` *Create a (signed) dummy matrix for hierarcical mapping of codes in data*

---

## Description

Create a (signed) dummy matrix for hierarcical mapping of codes in data

## Usage

```
DataDummyHierarchy(dataVector, dummyHierarchy)

DataDummyHierarchies(data, dummyHierarchies, colNamesFromData = FALSE)
```

## Arguments

| | |
|---|---|
| dataVector | A vector of codes in data |
| dummyHierarchy | Output from DummyHierarchy |
| data | data |
| dummyHierarchies | Output from DummyHierarchies |
| colNamesFromData | Column names from data when TRUE |

## Details

DataDummyHierarchies is a user-friendly wrapper for the original function DataDummyHierarchy. When colNamesFromData is FALSE (default), this function returns mapply(DataDummyHierarchy, data[names(dummyHierarchies)], dummyHierarchies).

## Value

A sparse matrix. Column names are taken from dataVector (if non-NULL) and row names are taken from the row names of dummyHierarchy.

## Author(s)

Øyvind Langsrud

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")[1:9, ]
hi <- FindHierarchies(z[, c("geo", "eu", "age", "year")])
dhi <- DummyHierarchies(hi, inputInOutput = TRUE)
DataDummyHierarchies(z, dhi, colNamesFromData = TRUE)
```

---

# data_diff_groups *Add diff_groups results as columns in a data frame*

---

## Description

data_diff_groups() is a wrapper around diff_groups() that runs the same analysis on two variables in a data frame and adds selected results back as new columns.

## Usage

```
data_diff_groups(
  data,
  input_vars,
 output_vars = c(is_common = "is_common", diff_1_2 = "diff_1_2", diff_2_1 = "diff_2_1",
    sum_1_2 = "sum_1_2", sum_2_1 = "sum_2_1"),
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data frame containing the variables to be compared. |
| input_vars | Character vector of length two specifying the names of the two variables in data to be compared. |
| output_vars | Named character vector defining which variables from the group results are added to data, and what their names will be in the output. |
| ... | Additional arguments passed to diff_groups(). |

## Value

A data frame identical to data, but with additional variables describing relationships between the two specified columns.

## Examples

```
df <- cbind(v1 = 1, SSBtoolsData("code_pairs"), v4 = 4)

data_diff_groups(df, input_vars = c("code_1", "code_2"))
```

---

# diff_groups *Difference and Sum Groups*

---

## Description

This function is a wrapper around RowGroups() for the specific case where the input contains two columns. It calls RowGroups() with returnGroups = TRUE, and extends the resulting data frame of unique code combinations with additional information about common groups, difference groups, and sum groups.

## Usage

```
diff_groups(
  x,
  ...,
  hiddenNA = TRUE,
```

```
  sep_common = "_=_",
  sep_diff = "_-_",
  sep_sum = c("_=_", "_+_"),
  outputNA = "NA",
  diff_extra = FALSE
)
```

## Arguments

| | |
|---|---|
| x | A data frame with exactly two columns. |
| ... | Additional arguments passed to RowGroups(). |
| hiddenNA | Logical. When TRUE (default), missing codes (NA) are treated as hidden categories — they are not available for computing difference and sum groups. See *Note* for details on how this differs from the NAomit parameter in RowGroups(). |
| sep_common | A character string used in the common column to separate codes that are identical across the two input columns. |
| sep_diff | A character string used in the diff_1_2 and diff_2_1 columns to indicate difference groups. The first column contains the parent code, and one or more child codes from the other column are subtracted. |
| sep_sum | A character vector of one or two elements used in the sum_1_2 and sum_2_1 columns to describe relationships where a code in one column represents the sum of several codes in the other. The first element (sep_sum[1]) acts as an equality sign, and the second element (sep_sum[2]) acts as a plus sign. If sep_sum has length 1, the same value is used for both positions. |
| outputNA | Character string used to represent NA values within the newly constructed text strings in the additional output columns. Only relevant when hiddenNA = FALSE. |
| diff_extra | Logical. When TRUE, additional difference-group variables are returned when found. |

## Details

The returned list contains the same elements as from RowGroups(), but with an extended groups data frame. The columns describe relationships between the two input columns as follows:

- **is_common** — TRUE when the two codes on the row are identical.
- **is_child_1**, **is_child_2** — TRUE when the code in the column is a subset or subgroup of a code in the other column.
- **common** — identical code pairs, formatted using sep_common.
- **diff_1_2**, **diff_2_1** — difference groups. The first element is the parent from the source column, followed by one or more child codes from the opposite column, joined using sep_diff.
- **sum_1_2**, **sum_2_1** — sum groups where a parent code in one column equals the sum of several codes in the other.

## Value

A list (as returned by RowGroups()), where the groups data frame is extended with additional descriptive columns indicating common, difference, and sum relationships between the two code columns.

## Note

The parameter `NAomit` from `RowGroups()` can still be set via `...`, but using it will remove rows containing `NA` before processing. The relationships found will then reflect the reduced data, which is usually not the intended behaviour when identifying relationships between code sets.

## See Also

`data_diff_groups()` for adding the results back as new columns in the data frame.

## Examples

```
df <- SSBtoolsData("code_pairs")

df

diff_groups(df)


d2 <- SSBtoolsData("d2")
diff_groups(d2[1:2])$groups
diff_groups(d2[2:3])$groups
```

---

# DimList2Hierarchy *DimList2Hierarchy*

---

## Description

From hierarchy/dimList as in sdcTable to to-from coded hierarchy

## Usage

```
DimList2Hierarchy(x)
```

## Arguments

| | |
|---|---|
| `x` | An element of a dimList as in sdcTable |

## Value

Data frame with to-from coded hierarchy

## Author(s)

Øyvind Langsrud

## See Also

DimList2Hrc, Hierarchy2Formula, AutoHierarchies.

## Examples

```
# First generate a dimList element
x <- FindDimLists(SSBtoolsData("sprt_emp_withEU")[, c("geo", "eu")], , total = "Europe")[[1]]
x

DimList2Hierarchy(x)
```

---

DimList2Hrc *DimList2Hrc/Hrc2DimList*

---

## Description

Conversion between hierarchies/dimList as in sdcTable and TauArgus coded hierarchies

## Usage

```
DimList2Hrc(dimList)

Hrc2DimList(hrc, total = "Total")
```

## Arguments

| | |
|---|---|
| dimList | List of data frames according to the specifications in sdcTable |
| hrc | List of character vectors |
| total | String used to name totals. |

## Value

See Arguments

## Author(s)

Øyvind Langsrud

## See Also

DimList2Hierarchy, Hierarchy2Formula, AutoHierarchies.

## Examples

```
# First generate dimList
dimList <- FindDimLists(SSBtoolsData("sprt_emp_withEU")[, c("geo", "eu", "age")])
dimList
hrc <- DimList2Hrc(dimList)
hrc
dimList2 <- Hrc2DimList(hrc)
identical(dimList, dimList2)
```

---

DummyApply *Apply a function to subsets defined by a dummy matrix*

---

## Description

For each column, `i`, of the matrix `x` of zeros and ones, the output value is equivalent to `FUN(y[x[, i] != 0])`.

## Usage

```
DummyApply(x, y, FUN = sum, simplify = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | A (sparse) dummy matrix |
| `y` | Vector of input values |
| `FUN` | A function |
| `simplify` | Parameter to `aggregate`. When `FALSE`, list output is ensured. |

## Details

With a dummy `x` and `FUN = sum`, output is equivalent to `z = t(x) %*% y`.

## Value

Vector of output values or a matrix when multiple outputs from `FUN` (see examples). List output is also possible (ensured when `simplify = FALSE`).

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")
z$age[z$age == "Y15-29"] <- "young"
z$age[z$age == "Y30-64"] <- "old"

a <- ModelMatrix(z, formula = ~age + geo, crossTable = TRUE)

cbind(as.data.frame(a$crossTable),
      sum1 = (Matrix::t(a$modelMatrix) %*% z$ths_per)[,1],
      sum2 = DummyApply(a$modelMatrix, z$ths_per, sum),
```

```
       max = DummyApply(a$modelMatrix, z$ths_per, max))

DummyApply(a$modelMatrix, z$ths_per, range)
DummyApply(a$modelMatrix, z$ths_per, range, simplify = FALSE)

a$modelMatrix[, c(3, 5)] <- 0   # Introduce two empty columns.
DummyApply(a$modelMatrix, z$ths_per, function(x){
  c(min = min(x),
    max = max(x),
    mean = mean(x),
    median = median(x),
    n = length(x))})

DummyApply(a$modelMatrix, z$ths_per, function(x) x, simplify = FALSE)
```

---

`DummyDuplicated` *Duplicated columns in dummy matrix*

---

## Description

The algorithm is based on `crossprod(x)` or `crossprod(x, u)` where `u` is a vector of random numbers

## Usage

```
DummyDuplicated(x, idx = FALSE, rows = FALSE, rnd = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | A matrix |
| `idx` | Indices returned when TRUE |
| `rows` | Duplicated rows instead when TRUE |
| `rnd` | Algorithm based on cross product with random numbers when TRUE (dummy matrix not required) |

## Details

The efficiency of the default algorithm depends on the sparsity of `crossprod(x)`. The random values are generated locally within the function without affecting the random value stream in R.

## Value

Logical vectors specifying duplicated columns or vector of indices (first match)

## Author(s)

Øyvind Langsrud

## Examples

```
x <- cbind(1, rbind(diag(2), diag(2)), diag(4)[, 1:2])
z <- Matrix::Matrix(x[c(1:4, 2:3), c(1, 2, 1:5, 5, 2)])

DummyDuplicated(z)
which(DummyDuplicated(z, rows = TRUE))

# Four ways to obtain the same result
DummyDuplicated(z, idx = TRUE)
DummyDuplicated(z, idx = TRUE, rnd = TRUE)
DummyDuplicated(Matrix::t(z), idx = TRUE, rows = TRUE)
DummyDuplicated(Matrix::t(z), idx = TRUE, rows = TRUE, rnd = TRUE)

# The unique values in four ways
which(!DummyDuplicated(z), )
which(!DummyDuplicated(z, rnd = TRUE))
which(!DummyDuplicated(Matrix::t(z), rows = TRUE))
which(!DummyDuplicated(Matrix::t(z), rows = TRUE, rnd = TRUE))
```

---

# DummyHierarchy *Converting hierarchy specifications to a (signed) dummy matrix*

---

## Description

A matrix for mapping input codes (columns) to output codes (rows) are created. The elements of the matrix specify how columns contribute to rows.

## Usage

```
DummyHierarchy(
  mapsFrom,
  mapsTo,
  sign,
  level,
  mapsInput = NULL,
  inputInOutput = FALSE,
  keepCodes = mapsFrom[integer(0)],
  unionComplement = FALSE,
  reOrder = FALSE
)

DummyHierarchies(
  hierarchies,
  data = NULL,
  inputInOutput = FALSE,
  unionComplement = FALSE,
  reOrder = FALSE
)
```

## Arguments

| | |
|---|---|
| mapsFrom | Character vector from hierarchy table |
| mapsTo | Character vector from hierarchy table |
| sign | Numeric vector of either 1 or -1 from hierarchy table |
| level | Numeric vector from hierarchy table |
| mapsInput | All codes in mapsFrom not in mapsTo (created automatically when NULL) and possibly other codes in input data. |
| inputInOutput | When FALSE all output rows represent codes in mapsTo |
| keepCodes | To prevent some codes to be removed when inputInOutput = FALSE |
| unionComplement | When TRUE, sign means union and complement instead of addition or subtraction (see note) |
| reOrder | When TRUE (FALSE is default) output codes are ordered differently, more similar to a usual model matrix ordering. |
| hierarchies | List of hierarchies |
| data | data |

## Details

DummyHierarchies is a user-friendly wrapper for the original function DummyHierarchy. Then, the logical input parameters are vectors (possibly recycled). mapsInput and keepCodes can be supplied as attributes. mapsInput will be generated when data is non-NULL.

## Value

A sparse matrix with row and column and names

## Note

With unionComplement = FALSE (default), the sign of each mapping specifies the contribution as addition or subtraction. Thus, values above one and negative values in output can occur. With unionComplement = TRUE, positive is treated as union and negative as complement. Then 0 and 1 are the only possible elements in the output matrix.

## Author(s)

Øyvind Langsrud

## Examples

```
# A hierarchy table
h <- SSBtoolsData("FIFA2018ABCD")

DummyHierarchy(h$mapsFrom, h$mapsTo, h$sign, h$level)
DummyHierarchy(h$mapsFrom, h$mapsTo, h$sign, h$level, inputInOutput = TRUE)
DummyHierarchy(h$mapsFrom, h$mapsTo, h$sign, h$level, keepCodes = c("Portugal", "Spain"))
```

```
# Extend the hierarchy table to illustrate the effect of unionComplement
h2 <- rbind(data.frame(mapsFrom = c("EU", "Schengen"), mapsTo = "EUandSchengen",
                       sign = 1, level = 3), h)

DummyHierarchy(h2$mapsFrom, h2$mapsTo, h2$sign, h2$level)
DummyHierarchy(h2$mapsFrom, h2$mapsTo, h2$sign, h2$level, unionComplement = TRUE)

# Extend mapsInput - leading to zero columns.
DummyHierarchy(h$mapsFrom, h$mapsTo, h$sign, h$level,
            mapsInput = c(h$mapsFrom[!(h$mapsFrom %in% h$mapsTo)], "Norway", "Finland"))

# DummyHierarchies
DummyHierarchies(FindHierarchies(SSBtoolsData("sprt_emp_withEU")[, c("geo", "eu", "age")]),
                 inputInOutput = c(FALSE, TRUE))
```

---

`dummy_aggregate` `aggregate_multiple_fun` *using a dummy matrix*

---

## Description

Wrapper to `aggregate_multiple_fun` that uses a dummy matrix instead of the by parameter. Functionality for non-dummy matrices as well.

## Usage

```
dummy_aggregate(
  data,
  x,
  vars,
  fun = NULL,
  dummy = TRUE,
  when_non_dummy = warning,
  keep_names = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing data to be aggregated |
| `x` | A (sparse) dummy matrix |
| `vars` | A named vector or list of variable names in `data`. The elements are named by the names of `fun`. All the pairs of variable names and function names thus define all the result variables to be generated.<br>• Parameter `vars` will converted to an internal standard by the function `fix_vars_amf`. Thus, function names and also output variable names can be coded in different ways. Multiple output variable names can be coded using `multi_sep`. See examples and examples in `fix_vars_amf`. Indices instead of variable names are allowed. |

- Omission of (some) names is possible since names can be omitted for one function (see fun below).
- A special possible feature is the combination of a single unnamed variable and all functions named. In this case, all functions are run and output variable names will be identical to the function names.

| | |
|---|---|
| fun | A named list of functions. These names will be used as suffixes in output variable names. Name can be omitted for one function. A vector of function as strings is also possible. When unnamed, these function names will be used directly. See the examples of fix_fun_amf, which is the function used to convert fun. Without specifying fun, the functions, as strings, are taken from the function names coded in vars. |
| dummy | When TRUE, only 0s and 1s are assumed in x. When FALSE, non-0s in x are passed as an additional first input parameter to the fun functions. Thus, the same result as matrix multiplication is achieved with fun = function(x, y) sum(x * y). In this case, the data will not be subjected to unlist. See aggregate_multiple_fun. |
| when_non_dummy | Function to be called when dummy is TRUE and when x is non-dummy. Supply NULL to do nothing. |
| keep_names | When TRUE, output row names are inherited from column names in x. |
| ... | Further arguments passed to aggregate_multiple_fun |

## Details

Internally this function make use of the ind parameter to aggregate_multiple_fun

## Value

data frame

## See Also

aggregate_multiple_fun

## Examples

```
# Code that generates output similar to the
# last example in aggregate_multiple_fun

d2 <- SSBtoolsData("d2")
set.seed(12)
d2$y <- round(rnorm(nrow(d2)), 2)
d <- d2[sample.int(nrow(d2), size = 20), ]

x <- ModelMatrix(d, formula = ~main_income:k_group - 1)

# with specified output variable names
my_range <- function(x) c(min = min(x), max = max(x))
dummy_aggregate(
   data = d,
   x = x,
```

```
   vars = list("freq", "y",
               `freqmin,freqmax` = list(ra = "freq"),
                yWmean  = list(wmean  = c("y", "freq"))),
   fun = c(sum, ra = my_range, wmean = weighted.mean))


# Make a non-dummy matrix
x2 <- x
x2[17, 2:5] <- c(-1, 3, 0, 10)
x2[, 4] <- 0

# Now warning
# Result is not same as t(x2) %*% d[["freq"]]
dummy_aggregate(data = d, x = x2, vars = "freq", fun = sum)

# Now same as t(x2) %*% d[["freq"]]
dummy_aggregate(data = d, x = x2,
                vars = "freq", dummy = FALSE,
                fun = function(x, y) sum(x * y))


# Same as t(x2) %*% d[["freq"]]  + t(x2^2) %*% d[["y"]]
dummy_aggregate(data = d, x = x2,
                vars = list(c("freq", "y")), dummy = FALSE,
                fun = function(x, y1, y2) {sum(x * y1) + sum(x^2 * y2)})
```

---

`Extend0` *Add zero frequency rows*

---

## Description

Microdata or tabular frequency data is extended to contain all combinations of unique rows of (hierarchical) groups of dimensional variables. Extra variables are extended by `NA`'s or `0`'s.

## Usage

```
Extend0(
  data,
  freqName = "freq",
  hierarchical = TRUE,
  varGroups = NULL,
  dimVar = NULL,
  extraVar = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | data frame |

| | |
|---|---|
| freqName | Name of (existing) frequency variable |
| hierarchical | Hierarchical variables treated atomatically when TRUE |
| varGroups | List of variable groups, possibly with data (see details and examples). |
| dimVar | The dimensional variables |
| extraVar | Extra variables as variable names, TRUE (all remaining) or FALSE (none). |

## Details

With no frequency variable in input (microdata), the frequency variable in output consists of ones and zeros. By default, all variables, except the frequencies, are considered as dimensional variables. By default, the grouping of dimensional variables is based on hierarchical relationships (`hierarchical = TRUE`). With `varGroups = NULL` and `hierarchical = FALSE`, each dimensional variable forms a separate group (as `as.list(dimVar)`). Parameter `extraVar` can be specified as variable names. `TRUE` means all remaining variables and `FALSE` no variables.

When the contents of `varGroups[[i]]` is variable names, the data frame `unique(data[varGroups[[i]]])` will be made as a building block within the function. A possibility is to supply such a data frame instead of variable names. Then, the building block will be `unique(varGroups[[i]])`. Names and data frames can be mixed.

## Value

Extended data frame

## See Also

Advanced possibilities by varGroups-attribute. See `Extend0rnd1`.

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")[c(1, 4:6, 8, 11:15), ]
z$age[z$age == "Y15-29"] <- "young"
z$age[z$age == "Y30-64"] <- "old"

Extend0(z[, -4])
Extend0(z, hierarchical = FALSE, dimVar = c("age", "geo", "eu"))
Extend0(z, hierarchical = FALSE, dimVar = c("age", "geo", "eu"), extraVar = "year")
Extend0(z, hierarchical = FALSE, dimVar = c("age", "geo", "eu"), extraVar = FALSE)
Extend0(z, varGroups = list(c("age", "geo", "year"), "eu"))
Extend0(MakeFreq(z[c(1, 1, 1, 2, 2, 3:10), -4]))
Extend0(z, "ths_per")

# varGroups with data frames (same result as with names above)
Extend0(z, varGroups = list(z[c("age", "geo", "year")], z["eu"]))

# varGroups with both names and data frame
Extend0(z, varGroups = list(c("year", "geo", "eu"), data.frame(age = c("middle", "old"))))
```

`Extend0rnd1` *varGroups-attribute to Extend0, Example functions*

## Description

Setting `attr(varGroups, "FunctionExtend0")` to a function makes `Extend0` behave differently

## Usage

```
Extend0rnd1(data, varGroups, k = 1, rndSeed = 123)

Extend0rnd2(...)

Extend0rnd1b(...)
```

## Arguments

| | |
|---|---|
| `data` | data.frame within `Extend0` |
| `varGroups` | argument to `Extend0` |
| `k` | Number of rows generated is approx. `k*nrow(data)` |
| `rndSeed` | Internal random seed to be used |
| `...` | Extra unused parameters |

## Details

The point is to create a function that takes `data` and `varGroups` as input and that returns a data frame with a limited number of combinations of the elements in `varGroups`. The example function here is limited to two varGroups elements.

## Value

a data frame

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")[c(1, 5, 8, 14), ]
z$age[z$age == "Y15-29"] <- "young"
z$age[z$age == "Y30-64"] <- "old"

varGroups <- list(c("year", "geo", "eu"), data.frame(age = c("middle", "old")))
Extend0(z, varGroups = varGroups)

attr(varGroups, "FunctionExtend0") <- Extend0rnd1
Extend0(z, varGroups = varGroups)

attr(varGroups, "FunctionExtend0") <- Extend0rnd1b
Extend0(z, varGroups = varGroups)
```

```
attr(varGroups, "FunctionExtend0") <- Extend0rnd2
Extend0(z, varGroups = varGroups)

# To see what's going on internally. Data used only via nrow
varGroups <- list(data.frame(ab = rep(c("a", "b"), each = 4), abcd = c("a", "b", "c", "d")),
                  data.frame(AB = rep(c("A", "B"), each = 3), ABC = c("A", "B", "C")))
a <- Extend0rnd1(data.frame(1:5), varGroups)
table(a[[1]], a[[2]])
table(a[[3]], a[[4]])
a <- Extend0rnd1b(data.frame(1:5), varGroups)
table(a[[1]], a[[2]])
table(a[[3]], a[[4]])
a <- Extend0rnd2(data.frame(1:5), varGroups[2:1])
table(a[[1]], a[[2]])
table(a[[3]], a[[4]])
a <- Extend0rnd1(data.frame(1:100), varGroups)
table(a[[1]], a[[2]]) # Maybe smaller numbers than expected since duplicates were removed
table(a[[3]], a[[4]])
```

---

`FactorLevCorr` *Factor level correlation*

---

## Description

A sort of correlation matrix useful to detect (hierarchical) relationships between the levels of factor variables.

## Usage

```
FactorLevCorr(x)
```

## Arguments

| | |
|---|---|
| `x` | Input matrix or data frame containing the variables |

## Value

Output is a sort of correlation matrix.

Here we refer to ni as the number of present levels of variable i (the number of unique elements) and we refer to mij as the number of present levels obtained by crossing variable i and variable j (the number unique rows of x[,c(i,j)]).

The diagonal elements of the output matrix contains the number of present levels of each variable (=ni).

The absolute values of off-diagonal elements:

| | |
|---|---|
| `0` | when mij = ni*nj |
| 1 | when mij = max(ni,nj) |

`Other values` Computed as (ni*nj-mij)/(ni*nj-max(ni,nj))

So 0 means that all possible level combinations exist in the data and 1 means that the two variables are hierarchically related.

The sign of off-diagonal elements:

`positive` when ni<nj

`negative` when ni>nj

In cases where ni=nj elements will be positive above the diagonal and negative below.

## Author(s)

Øyvind Langsrud

## Examples

```
x <- rep(c("A","B","C"),3)
y <- rep(c(11,22,11),3)
z <- c(1,1,1,2,2,2,3,3,3)
zy <- paste(z,y,sep="")
m <- cbind(x,y,z,zy)
FactorLevCorr(m)
```

---

# filter_by_variable *Filter a List of Items or Retrieve Names by a Variable*

---

## Description

Filters a list of items, retaining only those associated with a specific variable, or retrieves the names of items associated with the variable. The association between items and variables is provided via a named list, where each element contains a vector of variables corresponding to an item in `items`.

## Usage

```
filter_by_variable(variable, items, variable_mapping)

names_by_variable(variable, variable_mapping)
```

## Arguments

| | |
|---|---|
| `variable` | A character string. The variable to filter the items by. |
| `items` | A named list of elements. These can be any type of objects (e.g., formulas, data, etc.). |
| `variable_mapping` | A named list. Each element is a character vector of variables associated with the corresponding item in `items`. The names of the list in `variable_mapping` should match the names of the list in `items`. |

### Details

`filter_by_variable()` returns the filtered list of items, whereas `names_by_variable()` is a simpler function that just returns the names of the items.

### Value

- `filter_by_variable()`: A named list containing a subset of `items` where each element is associated with the specified `variable`. If no matches are found, an empty list is returned.
- `names_by_variable()`: A character vector of names from `variable_mapping` that are associated with the specified `variable`. If no matches are found, an empty character vector is returned.

### Note

This function is written and documented by ChatGPT after some discussion. The examples have been chosen to be relevant in connection with the `tables_by_formulas` function.

### Examples

```
items <- list(
  table_1 = ~region * sector2,
  table_2 = ~region1:sector4 - 1,
  table_3 = ~region + sector4 - 1
)

variable_mapping <- list(
  table_3 = c("z", "y"),
  table_1 = c("value", "x"),
  table_2 = c("value", "x", "y")
)

filter_by_variable("value", items, variable_mapping)
filter_by_variable("y", items, variable_mapping)
filter_by_variable("nonexistent", items, variable_mapping)

names_by_variable("value", variable_mapping)
names_by_variable("y", variable_mapping)
names_by_variable("nonexistent", variable_mapping)
```

---

## FindCommonCells *Finding commonCells*

---

### Description

Finding lists defining common cells as needed for the input parameter commonCells to the function protectLinkedTables in package sdcTable. The function handles two tables based on the same main variables but possibly different aggregating variables.

### Usage

```
FindCommonCells(dimList1, dimList2)
```

### Arguments

| | |
|---|---|
| dimList1 | As input parameter dimList to the function makeProblem in package sdcTable. |
| dimList2 | Another dimList with the same names and using the same level names. |

### Value

Output is a list according to the specifications in sdcTable.

### Author(s)

Øyvind Langsrud

### Examples

```
x <- rep(c('A','B','C'),3)
y <- rep(c(11,22,11),3)
z <- c(1,1,1,2,2,2,3,3,3)
zy <- paste(z,y,sep='')
m <- cbind(x,y,z,zy)
fg <- FindTableGroup(m,findLinked=TRUE)
dimLists <- FindDimLists(m,fg$groupVarInd)
# Using table1 and table2 in this example cause error,
# but in other cases this may work well
try(FindCommonCells(dimLists[fg$table$table1],dimLists[fg$table$table2]))
FindCommonCells(dimLists[c(1,2)],dimLists[c(1,3)])
```

---

## FindDimLists *Finding dimList*

---

### Description

Finding lists of level-hierarchy as needed for the input parameter dimList to the function makeProblem in package sdcTable

### Usage

```
FindDimLists(
  x,
  groupVarInd = HierarchicalGroups(x = x),
  addName = FALSE,
  sep = ".",
  xReturn = FALSE,
  total = "Total"
)
```

### Arguments

| | |
|---|---|
| x | Matrix or data frame containing the variables (micro data or cell counts data). |
| groupVarInd | List of vectors of indices defining the hierarchical variable groups. |
| addName | When TRUE the variable name is added to the level names, except for variables with most levels. |
| sep | A character string to separate when addName apply. |
| xReturn | When TRUE x is also in output, possibly changed according to addName. |
| total | String used to name totals. Can also be a vector of length ncol(x) or a named vector/list. If named, the dimension names used in the output must be present among the names in total. |

### Value

Output is a list according to the specifications in sdcTable. When xReturn is TRUE output has an extra list level and x is the first element.

### Author(s)

Øyvind Langsrud

### Examples

```
dataset <- SSBtoolsData("example1")
FindDimLists(dataset[1:2])
FindDimLists(dataset[2:3])
FindDimLists(dataset[1:4])

FindDimLists(SSBtoolsData("magnitude1")[1:4],
                total = c("TOTAL", "unused1", "Europe", "unused2"))

 x <- rep(c('A','B','C'),3)
 y <- rep(c(11,22,11),3)
 z <- c(1,1,1,2,2,2,3,3,3)
 zy <- paste(z,y,sep='')
 m <- cbind(x,y,z,zy)
 FindDimLists(m)
 FindDimLists(m, total = paste0("A", 1:4))
```

---

## FindDisclosiveCells *Find directly disclosive cells*

---

### Description

Function for determining which cells in a frequency table can lead to direct disclosure of an identifiable individual, assuming an attacker has the background knowledge to place themselves (or a coalition) in the table.

## Usage

```
FindDisclosiveCells(
  data,
  freq,
  crossTable,
  primaryDims = names(crossTable),
  unknowns = rep(NA, length(primaryDims)),
  total = rep("Total", length(primaryDims)),
  unknown.threshold = 0,
  coalition = 1,
  suppressSmallCells = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | the data set |
| `freq` | vector containing frequencies |
| `crossTable` | cross table of key variables produced by ModelMatrix in parent function |
| `primaryDims` | dimensions to be considered for direct disclosure. |
| `unknowns` | vector of unknown values for each of the primary dimensions. If a primary dimension does not contain unknown values, NA should be passed. |
| `total` | string name for marginal values |
| `unknown.threshold` | numeric for specifying a percentage for calculating safety of cells. A cell is "safe" in a row if the number of unknowns exceeds `unknown.threshold` percent of the row total. |
| `coalition` | maximum number of units in a possible coalition, default 1 |
| `suppressSmallCells` | logical variable which determines whether small cells (<= coalition) or large cells should be suppressed. Default FALSE. |
| `...` | parameters from main suppression method |

## Details

This function does not work on data containing hierarchical variables.

## Value

list with two named elements, the first ($primary) being a logical vector marking directly disclosive cells, the second ($numExtra) a data.frame containing information regarding the dimensions in which the cells are directly disclosive.

## Examples

```
extable <- data.frame(v1 = rep(c('a', 'b', 'c'), times = 4),
            v2 = c('i','i', 'i','h','h','h','i','i','i','h','h','h'),
            v3 = c('y', 'y', 'y', 'y', 'y', 'y','z','z', 'z', 'z', 'z', 'z'),
            freq = c(0,0,5,0,2,3,1,0,3,1,1,2))
ex_freq <- c(18,10,8,9,5,4,9,5,4,2,0,2,1,0,1,1,0,1,3,2,1,3,2,1,0,0,0,13,8,5,
             5,3,2,8,5,3)
cross <- ModelMatrix(extable,
                     dimVar = 1:3,
                     crossTable = TRUE)$crossTable

FindDisclosiveCells(extable, ex_freq, cross)
```

---

`FindHierarchies` *Finding hierarchies automatically from data*

---

## Description

`FindDimLists` and `AutoHierarchies` wrapped into a single function.

## Usage

```
FindHierarchies(data, total = "Total")
```

## Arguments

| | |
|---|---|
| `data` | Matrix or data frame containing the variables (micro data or cell counts data). |
| `total` | String used to name totals. A vector of length `ncol(data)` is also possible (see examples). |

## Value

List of hierarchies

## Author(s)

Øyvind Langsrud

## Examples

```
dataset <- SSBtoolsData("example1")
FindHierarchies(dataset[1:2])
FindHierarchies(dataset[2:3])
FindHierarchies(dataset[1:4])

FindHierarchies(SSBtoolsData("magnitude1")[1:4],
                total = c("TOTAL", "unused1", "Europe", "unused2"))

x <- rep(c("A", "B", "C"), 3)
```

```
y <- rep(c(11, 22, 11), 3)
z <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
zy <- paste(z, y, sep = "")
m <- cbind(x, y, z, zy)
FindHierarchies(m)
FindHierarchies(m, total = paste0("A", 1:4))
```

---

`FindTableGroup` *Finding table(s) of hierarchical variable groups*

---

## Description

A single table or two linked tables are found

## Usage

```
FindTableGroup(
  x = NULL,
  findLinked = FALSE,
  mainName = TRUE,
  fCorr = FactorLevCorr(x),
  CheckHandling = warning
)
```

## Arguments

| | |
|---|---|
| `x` | Matrix or data frame containing the variables |
| `findLinked` | When TRUE, two linked tables can be in output |
| `mainName` | When TRUE the groupVarInd ouput is named according to first variable in group. |
| `fCorr` | When non-null x is not needed as input. |
| `CheckHandling` | Function (warning or stop) to be used in problematic situations. |

## Value

Output is a list with items

| | |
|---|---|
| `groupVarInd` | List defining the hierarchical variable groups. First variable has most levels. |
| `table` | List containing one or two tables. These tables are coded as indices referring to elements of groupVarInd. |

## Author(s)

Øyvind Langsrud

## Examples

```
x <- rep(c('A','B','C'),3)
y <- rep(c(11,22,11),3)
z <- c(1,1,1,2,2,2,3,3,3)
zy <- paste(z,y,sep='')
m <- cbind(x,y,z,zy)
FindTableGroup(m)
FindTableGroup(m,findLinked=TRUE)
```

---

`FormulaSelection.default`

*Limit matrix or data frame to selected model terms*

---

## Description

For use with output from `ModelMatrix` or data frames derived from such output. It is a generic function which means that methods for other input objects can be added.

## Usage

```
## Default S3 method:
FormulaSelection(x, formula, intercept = NA, logical = FALSE)

FormulaSelection(x, formula, intercept = NA, logical = FALSE)

formula_selection(x, formula, intercept = NA, logical = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Model matrix or a data frame |
| `formula` | Formula representing the limitation or character string(s) to be converted to a formula (see details) |
| `intercept` | Parameter that specifies whether a possible intercept term (overall total) should be included in the output. Default is `TRUE` when a formula is input. Otherwise, see details. |
| `logical` | When `TRUE`, the logical selection vector is returned. |

## Details

The selection is based on `startCol` or `startRow` attribute in input `x`.

With **formula as character**:

- `~` is included: Input is converted by `as.formula` and default intercept is `TRUE`.
- `~` is not included: Internally, input data is converted to a formula by adding `~` and possibly `+`'s when the length is `>1`. Default intercept is `FALSE` unless `"1"` or `"(Intercept)"` (is changed internally to `"1"`) is included.

## Value

Limited model matrix or a data frame

## Note

`formula_selection` and `FormulaSelection` are identical

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")
z$age[z$age == "Y15-29"] <- "young"
z$age[z$age == "Y30-64"] <- "old"

x <- ModelMatrix(z, formula = ~age * year)

FormulaSelection(x, "age")
FormulaSelection(x, ~year)
FormulaSelection(x, ~year:age)

# x1, x2, x3, x4 and x4 are identical
x1 <- FormulaSelection(x, ~age)
x2 <- FormulaSelection(x, "~age")
x3 <- FormulaSelection(x, "age", intercept = TRUE)
x4 <- FormulaSelection(x, c("1", "age"))
x5 <- FormulaSelection(x, c("(Intercept)", "age"))


a <- ModelMatrix(z, formula = ~age * geo + year, crossTable = TRUE)
b <- cbind(as.data.frame(a$crossTable),
           sum = (Matrix::t(a$modelMatrix) %*% z$ths_per)[, 1],
           max = DummyApply(a$modelMatrix,
           z$ths_per, max))
rownames(b) <- NULL
attr(b, "startRow") <- attr(a$modelMatrix, "startCol", exact = TRUE)

FormulaSelection(b, ~geo * age)
FormulaSelection(b, "age:geo")
FormulaSelection(b, ~year - 1)
FormulaSelection(b, ~geo:age, logical = TRUE)
```

---

# FormulaSums *Sums (aggregates) and/or sparse model matrix with possible cross table*

---

## Description

By default this function return sums if the formula contains a response part and a model matrix otherwise

## Usage

```
FormulaSums(
  data,
  formula,
  makeNames = TRUE,
  crossTable = FALSE,
  total = "Total",
  printInc = FALSE,
  dropResponse = FALSE,
  makeModelMatrix = NULL,
  sep = "-",
  sepCross = ":",
  avoidHierarchical = FALSE,
  includeEmpty = FALSE,
  NAomit = TRUE,
  rowGroupsPackage = "base",
  viaSparseMatrix = TRUE,
  ...
)

Formula2ModelMatrix(data, formula, dropResponse = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame |
| `formula` | A model formula |
| `makeNames` | Column/row names made when TRUE |
| `crossTable` | Cross table in output when TRUE |
| `total` | Total code(s) which can be provided in several forms:<br>• A single string (e.g. `"TOTAL"`) to be applied to all variables.<br>• A **named vector** or **named list** giving one total code per variable, e.g. `c(var1 = "ALL_A", var2 = "ALL_B")` or `list(var1 = "ALL_A", var2 = "ALL_B")`. |
| `printInc` | Printing "..." to console when TRUE |
| `dropResponse` | When TRUE response part of formula ignored. |
| `makeModelMatrix` | Make model matrix when TRUE. NULL means automatic. |
| `sep` | String to separate when creating column names |
| `sepCross` | String to separate when creating column names involving crossing |
| `avoidHierarchical` | Whether to avoid treating of hierarchical variables. Instead of logical, variables can be specified. |
| `includeEmpty` | When `TRUE`, empty columns of the model matrix (only zeros) are included. This is not implemented when a response term is included in the formula and `dropResponse = FALSE` (error will be produced). |

NAomit — When TRUE, NAs in the grouping variables are omitted in output and not included as a separate category. Technically, this parameter is utilized through the function RowGroups.

rowGroupsPackage — Parameter pkg to the function RowGroups. Default is "base". Setting this parameter to "data.table" can improve speed.

viaSparseMatrix — When TRUE, the model matrix is constructed by a single call to sparseMatrix. Setting it to FALSE reverts to the previous behavior. This parameter is included for testing purposes and will likely be removed in future versions.

... — Further arguments to be passed to FormulaSums

## Details

In the original version of the function the model matrix was constructed by calling fac2sparse repeatedly. Now this is replaced by a single call to sparseMatrix. The sums are computed by calling aggregate repeatedly. Hierarchical variables handled when constructing cross table. Column names constructed from the cross table. The returned model matrix includes the attribute startCol (see last example line).

## Value

A matrix of sums, a sparse model matrix or a list of two or three elements (model matrix and cross table and sums when relevant).

## Author(s)

Øyvind Langsrud

## See Also

ModelMatrix

## Examples

```
x <- SSBtoolsData("sprt_emp_withEU")

FormulaSums(x, ths_per ~ year*geo + year*eu)
FormulaSums(x, ~ year*age*eu)
FormulaSums(x, ths_per ~ year*age*geo + year*age*eu, crossTable = TRUE, makeModelMatrix = TRUE)
FormulaSums(x, ths_per ~ year:age:geo -1)
m <- Formula2ModelMatrix(x, ~ year*geo + year*eu)
print(m[1:3, ], col.names = TRUE)
attr(m, "startCol")
```

## formula_utils *Functions for formula manipulation*

### Description

Functions for formula manipulation

### Details

- combine_formulas: Combine formulas
- formula_from_vars: Generate model formula by specifying which variables have totals or not
- substitute_formula_vars: Replace variables in formula with sum of other variables
- formula_term_labels: Retrieve term labels from a formula

## GaussIndependent *Linearly independent rows and columns by Gaussian elimination*

### Description

The function is written primarily for large sparse matrices with integers and even more correctly it is primarily written for dummy matrices (0s and 1s in input matrix).

### Usage

```
GaussIndependent(
  x,
  printInc = FALSE,
  tolGauss = (.Machine$double.eps)^(1/2),
  testMaxInt = 0,
  allNumeric = FALSE
)

GaussRank(x, printInc = FALSE)
```

### Arguments

| | |
|---|---|
| x | A (sparse) matrix |
| printInc | Printing "..." to console when TRUE |
| tolGauss | A tolerance parameter for sparse Gaussian elimination and linear dependency. This parameter is used only in cases where integer calculation cannot be used. |
| testMaxInt | Parameter for testing: The Integer overflow situation will be forced when testMaxInt is exceeded |
| allNumeric | Parameter for testing: All calculations use numeric algorithm (as integer overflow) when TRUE |

## Details

GaussRank returns the rank

## Value

List of logical vectors specifying independent rows and columns

## Note

The main algorithm is based on integers and exact calculations. When integers cannot be used (because of input or overflow), the algorithm switches. With `printInc = TRUE` as a parameter, `.....` change to `-----` when switching to numeric algorithm. With numeric algorithm, a kind of tolerance for linear dependency is included. This tolerance is designed having in mind that the input matrix is a dummy matrix.

## Examples

```
x <- ModelMatrix(SSBtoolsData("z2"), formula = ~fylke + kostragr * hovedint - 1)

GaussIndependent(x)
GaussRank(x)
GaussRank(Matrix::t(x))

## Not run:
# For comparison, qr-based rank may not work
rankMatrix(x, method = "qr")

# Dense qr works
qr(as.matrix(x))$rank

## End(Not run)
```

---

# GaussIterationFunction

*An* `iFunction` *argument to* GaussSuppression

---

## Description

Use this function as `iFunction` or write your own using the same seven first parameters and also using `...`.

## Usage

```
GaussIterationFunction(i, I, j, J, true, false, na, filename = NULL, ...)
```

### Arguments

| | |
|---|---|
| i | Number of candidates processed (columns of x) |
| I | Total number of candidates to be processed (columns of x) |
| j | Number of eliminated dimensions (rows of x) |
| J | Total number of dimensions (rows of x) |
| true | Candidates decided to be suppressed |
| false | Candidates decided to be not suppressed |
| na | Candidates not decided |
| filename | When non-NULL, the above arguments will be saved to this file. Note that GaussSuppression passes this parameter via .... |
| ... | Extra parameters |

### Details

The number of candidates decided (true and false) may differ from the number of candidates processed (i) due to parameter removeDuplicated and because the decision for some unprocessed candidates can be found due to empty columns.

### Value

NULL

---

## GaussSuppression *Secondary suppression by Gaussian elimination*

---

### Description

Sequentially the secondary suppression candidates (columns in x) are used to reduce the x-matrix by Gaussian elimination. Candidates who completely eliminate one or more primary suppressed cells (columns in x) are omitted and made secondary suppressed. This ensures that the primary suppressed cells do not depend linearly on the non-suppressed cells. How to order the input candidates is an important choice. The singleton problem and the related problem of zeros are also handled.

### Usage

```
GaussSuppression(
  x,
  candidates = 1:ncol(x),
  primary = NULL,
  forced = NULL,
  hidden = NULL,
  singleton = rep(FALSE, nrow(x)),
  singletonMethod = "anySum",
  printInc = TRUE,
```

```
  tolGauss = (.Machine$double.eps)^(1/2),
  whenEmptySuppressed = warning,
  whenEmptyUnsuppressed = message,
  whenPrimaryForced = warning,
  removeDuplicated = TRUE,
  iFunction = GaussIterationFunction,
  iWait = Inf,
  xExtraPrimary = NULL,
  unsafeAsNegative = FALSE,
  printXdim = FALSE,
  cell_grouping = NULL,
  table_id = NULL,
  auto_anySumNOTprimary = TRUE,
  auto_subSumAny = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | Matrix that relates cells to be published or suppressed to inner cells. yPublish = crossprod(x,yInner) |
| candidates | Indices of candidates for secondary suppression |
| primary | Indices of primary suppressed cells |
| forced | Indices forced to be not suppressed. forced has precedence over primary. See whenPrimaryForced below. |
| hidden | Indices to be removed from the above candidates input (see details) |
| singleton | Logical or integer vector of length nrow(x) specifying inner cells for singleton handling. Normally, for frequency tables, this means cells with 1s when 0s are non-suppressed and cells with 0s when 0s are suppressed. For some singleton methods, integer values representing the unique magnitude table contributors are needed. For all other singleton methods, only the values after conversion with as.logical matter. |
| singletonMethod | Method for handling the problem of singletons and zeros: "anySum" (default), "anySum0", "anySumNOTprimary", "subSum", "subSpace", "sub2Sum", "none" or a NumSingleton method (see details). |
| printInc | Printing "..." to console when TRUE |
| tolGauss | A tolerance parameter for sparse Gaussian elimination and linear dependency. This parameter is used only in cases where integer calculation cannot be used. |
| whenEmptySuppressed | Function to be called when empty input to primary suppressed cells is problematic. Supply NULL to do nothing. |
| whenEmptyUnsuppressed | Function to be called when empty input to candidate cells may be problematic. Supply NULL to do nothing. |

whenPrimaryForced
: Function to be called if any forced cells are primary suppressed (suppression will be ignored). Supply NULL to do nothing. The same function will also be called when there are forced cells marked as singletons (will be ignored).

removeDuplicated
: Specifies whether to remove duplicated columns and rows in x before running the main algorithm. Removing duplicates results in a faster algorithm while generally maintaining the same results. In some cases, singleton handling for magnitude tables may be affected. In such cases, singleton handling will generally be improved. Singletons are considered when removing duplicate rows, so not all duplicates are removed. The available options for removeDuplicated are as follows:

  - TRUE (default): Removes both duplicate columns and rows.
  - FALSE: No removal of duplicates.
  - "cols": Removes only duplicate columns.
  - "rows": Removes only duplicate rows.
  - "rows2": Removes only duplicate non-singleton rows in a way that preserves singleton handling.
  - Combined possibilities: Variants can be combined with "_". For example, "cols_rows" is equivalent to TRUE, and "cols_rows2" represents an alternative variant. Combining "rows" and "rows2" is possible, but superfluous calculations are then performed.
  - "test": A special variant for testing purposes. The four configurations TRUE, FALSE, "cols_rows2", and "rows" are executed.

iFunction
: A function to be called during the iterations. See the default function, GaussIterationFunction, for description of parameters.

iWait
: The minimum number of seconds between each call to iFunction. Whenever iWait<Inf, iFunction will also be called after last iteration.

xExtraPrimary
: Extra x-matrix that defines extra primary suppressed cells in addition to those defined by other inputs.

unsafeAsNegative
: When TRUE, unsafe primary cells due to forced cells are included in the output vector as negative indices.

printXdim
: When set to TRUE, the printInc parameter is also automatically set to TRUE. Additionally, the dimensions of the x matrix are printed twice: first, the dimensions of the input x, potentially extended with xExtraPrimary; second, the dimensions after applying singletonMethod and removeDuplicated.

cell_grouping
: Numeric vector indicating suppression group membership. Cells with the same non-zero value belong to the same suppression group, meaning they will be suppressed or non-suppressed together. A value of 0 indicates that the cell is not a member of any suppression group.

table_id
: A parameter that can be provided in addition to cell_grouping to reduce computation time, when x is a block-diagonal matrix. Each block represents a separate table, and table_id indicates table affiliation. Note: no check is performed to verify that table_id corresponds to the block structure of x.

auto_anySumNOTprimary
: When TRUE (default), the singletonMethod "anySumNOTprimary" may be forced if a check indicates that singletons are not primary suppressed. Set this to FALSE in cases where the x matrix has already undergone duplicate row removal, as the check may then produce incorrect results.

auto_subSumAny
: When TRUE (default), and singletonMethod is "anySum", it is internally changed to "subSumAny" if there are forced cells. This is done to give information about unsafe cells.

...
: Extra unused parameters

## Details

It is possible to specify too many (all) indices as candidates. Indices specified as primary or hidden will be removed. Hidden indices (not candidates or primary) refer to cells that will not be published, but do not need protection.

- **Singleton methods for frequency tables:** All singleton methods, except "sub2Sum" and the NumSingleton methods, have been implemented with frequency tables in mind. The singleton method "subSum" makes new virtual primary suppressed cells, which are the sum of the singletons within each group. The "subSpace" method is conservative and ignores the singleton dimensions when looking for linear dependency. The default method, "anySum", is between the other two. Instead of making virtual cells of sums within groups, the aim is to handle all possible sums, also across groups. In addition, "subSumSpace" and "subSumAny" are possible methods, primarily for testing. These methods are similar to "subSpace" and "anySum", and additional cells are created as in "subSum". It is believed that the extra cells are redundant. Note that in order to give information about unsafe cells, "anySum" is internally changed to "subSumAny" when there are forced cells. All the above methods assume that any published singletons are primary suppressed. If this is not the case, either "anySumNOTprimary" or "anySum0" must be used. Notably, "anySum0" is an enhancement of "anySumNOTprimary" for situations where zeros are singletons. Using that method avoids suppressing a zero marginal along with only one of its children.
- **Singleton methods for magnitude tables:** The singleton method "sub2Sum" makes new virtual primary suppressed cells, which are the sum of two inner cells. This is done when a group contains exactly two primary suppressed inner cells provided that at least one of them is singleton. This was the first method implemented. Other magnitude methods follow the coding according to NumSingleton. The "sub2Sum" method is equivalent to "numFFT". Also note that "num", "numFFF" and "numFTF" are equivalent to "none".
- **Combined:** For advanced use, singleton can be a two-element list with names "freq" and "num". Then singletonMethod must be a corresponding named two-element vector. For example: singletonMethod = c(freq = "anySumNOTprimary", num = "sub2Sum")

## Value

Secondary suppression indices

## References

Langsrud, Ø. (2024): "Secondary Cell Suppression by Gaussian Elimination: An Algorithm Suitable for Handling Issues with Zeros and Singletons". Presented at: *Privacy in statistical databases*,

Antibes, France. September 25-27, 2024. doi:10.1007/9783031696510_6

## Examples

```
# Input data
df <- data.frame(values = c(1, 1, 1, 5, 5, 9, 9, 9, 9, 0, 0, 0, 7, 7),
                 var1 = rep(1:3, each = 5),
                 var2 = c("A", "B", "C", "D", "E"), stringsAsFactors = FALSE)

# Make output data frame and x
fs <- FormulaSums(df, values ~ var1 * var2, crossTable = TRUE, makeModelMatrix = TRUE)
x <- fs$modelMatrix
datF <- data.frame(fs$crossTable, values = as.vector(fs$allSums))

# Add primary suppression
datF$primary <- datF$values
datF$primary[datF$values < 5 & datF$values > 0] <- NA
datF$suppressedA <- datF$primary
datF$suppressedB <- datF$primary
datF$suppressedC <- datF$primary

# zero secondary suppressed
datF$suppressedA[GaussSuppression(x, primary = is.na(datF$primary))] <- NA

# zero not secondary suppressed by first in ordering
datF$suppressedB[GaussSuppression(x, c(which(datF$values == 0), which(datF$values > 0)),
                            primary = is.na(datF$primary))] <- NA

# with singleton
datF$suppressedC[GaussSuppression(x, c(which(datF$values == 0), which(datF$values > 0)),
                        primary = is.na(datF$primary), singleton = df$values == 1)] <- NA

datF



#### with cell_grouping

candidates <- c(which(datF$values == 0), which(datF$values > 0))
primary <- 10:12
cell_grouping <- rep(0, 24)

# same as without cell_grouping
GaussSuppression(x, candidates, primary, cell_grouping = cell_grouping)

cell_grouping[c(16, 20:21)] <- 1
cell_grouping[c(10, 4)] <- 2  # 10 is primary

GaussSuppression(x, candidates, primary, cell_grouping = cell_grouping)
```

get_colnames *Get column names from a data.frame, tibble, or data.table*

## Description

This helper function returns column names based on either column indices (numeric) or column names (character). It works consistently across `data.frame`, `tibble`, and `data.table` objects.

## Usage

```
get_colnames(data, cols, preserve_duplicates = FALSE, preserve_NULL = FALSE)
```

## Arguments

| | |
|---|---|
| `data` | A data frame, tibble, or data.table. |
| `cols` | Column selection, either as numeric indices, character names, or `NULL`. |
| `preserve_duplicates` | Logical, default `FALSE`. If `TRUE`, duplicates and order in `cols` are preserved. If `FALSE`, duplicates are removed while preserving order of first appearance. |
| `preserve_NULL` | Logical, default `FALSE`. If `TRUE`, returns `NULL` when `cols = NULL`. If `FALSE`, returns `character(0)` when `cols = NULL`. |

## Details

By default, `cols = NULL` returns `character(0)`, matching the behavior of `names(data[1, NULL, drop = FALSE])`. If `preserve_NULL = TRUE`, the function instead returns `NULL`.

## Value

A character vector of column names, or `NULL` if `cols = NULL` and `preserve_NULL = TRUE`.

## Note

This function is written and documented by ChatGPT after some discussion.

## Examples

```
df <- data.frame(a = 1, b = 2, c = 3)

# NULL input handling
get_colnames(df, NULL)
get_colnames(df, NULL, preserve_NULL = TRUE)

# Numeric input
get_colnames(df, c(2, 2, 1))
get_colnames(df, c(2, 2, 1), preserve_duplicates = TRUE)

# Character input
```

```
get_colnames(df, c("c", "a", "c"))
get_colnames(df, c("c", "a", "c"), preserve_duplicates = TRUE)
```

---

`HierarchicalGroups` *Finding hierarchical variable groups*

---

## Description

According to the (factor) levels of the variables

## Usage

```
HierarchicalGroups(
  x = NULL,
  mainName = TRUE,
  eachName = FALSE,
  fCorr = FactorLevCorr(x)
)
```

## Arguments

| | |
|---|---|
| `x` | Matrix or data frame containing the variables |
| `mainName` | When TRUE output list is named according to first variable in group. |
| `eachName` | When TRUE variable names in output instead of indices. |
| `fCorr` | When non-null, x is not needed as input. |

## Value

Output is a list containing the groups. First variable has most levels.

## Author(s)

Øyvind Langsrud

## Examples

```
dataset <- SSBtoolsData("example1")
HierarchicalGroups(dataset[1:2], eachName = TRUE)
HierarchicalGroups(dataset[2:3])
HierarchicalGroups(dataset[1:4], eachName = TRUE)

HierarchicalGroups(SSBtoolsData("magnitude1")[1:4])

 x <- rep(c("A","B","C"),3)
 y <- rep(c(11,22,11),3)
 z <- c(1,1,1,2,2,2,3,3,3)
 zy <- paste(z,y,sep="")
 m <- cbind(x,y,z,zy)
 HierarchicalGroups(m)
```

`HierarchicalWildcardGlobbing` *Find variable combinations by advanced wildcard/globbing specifications.*

## Description

Find combinations present in an input data frame or, when input is a list, find all possible combinations that meet the requirements.

## Usage

```
HierarchicalWildcardGlobbing(
  z,
  wg,
  useUnique = NULL,
  useFactor = FALSE,
  makeWarning = TRUE,
  printInfo = FALSE,
  useMatrixToDataFrame = TRUE,
  invert = "!"
)
```

## Arguments

| | |
|---|---|
| `z` | list or data.frame |
| `wg` | data.frame with data globbing and wildcards |
| `useUnique` | Logical variable about recoding within the algorithm. By default (NULL) an automatic decision is made. |
| `useFactor` | When TRUE, internal factor recoding is used. |
| `makeWarning` | When TRUE, warning is made in cases of unused variables. Only variables common to z and wg are used. |
| `printInfo` | When TRUE, information is printed during the process. |
| `useMatrixToDataFrame` | When TRUE, special functions (DataFrameToMatrix/MatrixToDataFrame) for improving speed and memory is utilized. |
| `invert` | Character to invert each single selection. |

## Details

The final variable combinations must meet the requirements in each positive sign group and must not match the requirements in the negative sign groups.The function is implemented by calling `WildcardGlobbing` several times within an algorithm that uses hierarchical clustering (`hclust`).

## Value

data.frame

## Author(s)

Øyvind Langsrud

## Examples

```
# useUnique=NULL betyr valg ut fra antall rader i kombinasjonsfil
data(precip)
data(mtcars)
codes <- as.character(c(100, 200, 300, 600, 700, 101, 102, 103, 104, 134, 647, 783,
                        13401, 13402, 64701, 64702))


# Create list input
zList <- list(car = rownames(mtcars), wt = as.character(1000 * mtcars$wt),
              city = names(precip), code = codes)

# Create data.frame input
m <- cbind(car = rownames(mtcars), wt = as.character(1000 * mtcars$wt))
zFrame <- data.frame(m[rep(1:NROW(m), each = 35), ],
                     city = names(precip), code = codes, stringsAsFactors = FALSE)

# Create globbing/wildcards input
wg <- data.frame(rbind(c("Merc*", ""    , ""    , "?00"  ),
                       c("F*"   , ""    , ""    , "?????"),
                       c(""     , "???0", "C*"  , ""     ),
                       c(""     , ""    , "!Co*", ""     ),
                       c(""     , ""    , "?i*" , "????2"),
                       c(""     , ""    , "?h*" , "????1")),
           sign = c("+", "+", "+", "+", "-", "-"), stringsAsFactors = FALSE)
names(wg)[1:4] <- names(zList)



# ======================================================================
#   Finding unique combinations present in the input data frame
# ======================================================================


# Using first row of wg. Combinations of car starting with Merc
# and three-digit code ending with 00
HierarchicalWildcardGlobbing(zFrame[, c(1, 4)], wg[1, c(1, 4, 5)])

# Using first row of wg. Combinations of all four variables
HierarchicalWildcardGlobbing(zFrame, wg[1, ])

# More combinations when using second row also
HierarchicalWildcardGlobbing(zFrame, wg[1:2, ])
```

```
# Less combinations when using third row also
# since last digit of wt must be 0 and only cities starting with C
HierarchicalWildcardGlobbing(zFrame, wg[1:3, ])


# Less combinations when using fourth row also since city cannot start with Co
HierarchicalWildcardGlobbing(zFrame, wg[1:4, ])

# Less combinations when using fourth row also
# since specific combinations of city and code are removed
HierarchicalWildcardGlobbing(zFrame, wg)


# ====================================================================
#  Using list input to create all possible combinations
# ====================================================================

dim(HierarchicalWildcardGlobbing(zList, wg))

# same result with as.list since same unique values of each variable
dim(HierarchicalWildcardGlobbing(as.list(zFrame), wg))
```

---

`Hierarchies2ModelMatrix` *Model matrix representing crossed hierarchies*

---

## Description

Make a model matrix, x, that corresponds to data and represents all hierarchies crossed. This means that aggregates corresponding to numerical variables can be computed as `t(x) %*% y`, where `y` is a matrix with one column for each numerical variable.

## Usage

```
Hierarchies2ModelMatrix(
  data,
  hierarchies,
  inputInOutput = TRUE,
  crossTable = FALSE,
  total = "Total",
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level =
    "level"),
  unionComplement = FALSE,
  reOrder = TRUE,
  select = NULL,
  removeEmpty = FALSE,
  selectionByMultiplicationLimit = 10^7,
  makeColnames = TRUE,
  verbose = FALSE,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| data | Matrix or data frame with data containing codes of relevant variables |
| hierarchies | List of hierarchies, which can be converted by AutoHierarchies. Thus, the variables can also be coded by "rowFactor" or "", which correspond to using the categories in the data. |
| inputInOutput | Logical vector (possibly recycled) for each element of hierarchies. TRUE means that codes from input are included in output. Values corresponding to "rowFactor" or "" are ignored. Also see note. |
| crossTable | Cross table in output when TRUE |
| total | See AutoHierarchies |
| hierarchyVarNames | Variable names in the hierarchy tables as in HierarchyFix |
| unionComplement | Logical vector (possibly recycled) for each element of hierarchies. When TRUE, sign means union and complement instead of addition or subtraction. Values corresponding to "rowFactor" and "colFactor" are ignored. |
| reOrder | When TRUE (default) output codes are ordered in a way similar to a usual model matrix ordering. |
| select | Data frame specifying variable combinations for output or a named list specifying code selections for each variable (see details). |
| removeEmpty | When TRUE and when select is not a data frame, empty columns (only zeros) are not included in output. |
| selectionByMultiplicationLimit | With non-NULL select and when the number of elements in the model matrix exceeds this limit, the computation is performed by a slower but more memory efficient algorithm. |
| makeColnames | Colnames included when TRUE (default). |
| verbose | Whether to print information during calculations. FALSE is default. |
| ... | Extra unused parameters |

## Details

This function makes use of AutoHierarchies and HierarchyCompute via HierarchyComputeDummy. Since the dummy matrix is transposed in comparison to HierarchyCompute, the parameter rowSelect is renamed to select and makeRownames is renamed to makeColnames.

The select parameter as a list can be partially specified in the sense that not all hierarchy names have to be included. The parameter inputInOutput will only apply to hierarchies that are not in the select list (see note).

## Value

A sparse model matrix or a list of two elements (model matrix and cross table)

## Note

The `select` as a list is run via a special coding of the `inputInOutput` parameter. This parameter is converted into a list (`as.list`) and `select` elements are inserted into this list. This is also an additional option for users of the function.

## Author(s)

Øyvind Langsrud

## See Also

ModelMatrix, HierarchiesAndFormula2ModelMatrix

## Examples

```
# Create some input
z <- SSBtoolsData("sprt_emp_withEU")
ageHier <- SSBtoolsData("sprt_emp_ageHier")
geoDimList <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]


# First example has list output
Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList), inputInOutput = FALSE,
                        crossTable = TRUE)


m1 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList), inputInOutput = FALSE)
m2 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList))
m3 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList, year = ""),
                              inputInOutput = FALSE)
m4 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList, year = "allYears"),
                              inputInOutput = c(FALSE, FALSE, TRUE))

# Illustrate the effect of unionComplement, geoHier2 as in the examples of HierarchyCompute
geoHier2 <- rbind(data.frame(mapsFrom = c("EU", "Spain"), mapsTo = "EUandSpain", sign = 1),
                  SSBtoolsData("sprt_emp_geoHier")[, -4])
m5 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoHier2, year = "allYears"),
                              inputInOutput = FALSE)  # Spain is counted twice
m6 <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoHier2, year = "allYears"),
                              inputInOutput = FALSE, unionComplement = TRUE)


# Compute aggregates
ths_per <- as.matrix(z[, "ths_per", drop = FALSE]) # matrix with the values to be aggregated
Matrix::t(m1) %*% ths_per  # Matrix::crossprod(m1, ths_per) is equivalent and faster
Matrix::t(m2) %*% ths_per
Matrix::t(m3) %*% ths_per
Matrix::t(m4) %*% ths_per
Matrix::t(m5) %*% ths_per
Matrix::t(m6) %*% ths_per
```

```
# Example using the select parameter as a data frame
select <- data.frame(age = c("Y15-64", "Y15-29", "Y30-64"), geo = c("EU", "nonEU", "Spain"))
m2a <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList), select = select)

# Same result by slower alternative
m2B <- Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList), crossTable = TRUE)
m2b <- m2B$modelMatrix[, Match(select, m2B$crossTable), drop = FALSE]
Matrix::t(m2b) %*% ths_per

# Examples using the select parameter as a list
Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList),
       inputInOutput = FALSE,
       select = list(geo = c("nonEU", "Portugal")))
Hierarchies2ModelMatrix(z, list(age = ageHier, geo = geoDimList),
       select = list(geo = c("nonEU", "Portugal"), age = c("Y15-64", "Y15-29")))
```

---

HierarchiesAndFormula2ModelMatrix
*Model matrix representing crossed hierarchies according to a formula*

---

## Description

How to cross the hierarchies are defined by a formula. The formula is automatically simplified when totals are involved.

## Usage

```
HierarchiesAndFormula2ModelMatrix(
  data,
  hierarchies,
  formula,
  inputInOutput = TRUE,
  makeColNames = TRUE,
  crossTable = FALSE,
  total = "Total",
  simplify = TRUE,
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level =
    "level"),
  unionComplement = FALSE,
  removeEmpty = FALSE,
  reOrder = TRUE,
  sep = "-",
  ...
)
```

## Arguments

| | |
|---|---|
| data | Matrix or data frame with data containing codes of relevant variables |
| hierarchies | List of hierarchies, which can be converted by AutoHierarchies. Thus, the variables can also be coded by "rowFactor" or "", which correspond to using the categories in the data. |
| formula | A model formula |
| inputInOutput | Logical vector (possibly recycled) for each element of hierarchies. TRUE means that codes from input are included in output. Values corresponding to "rowFactor" or "" are ignored. |
| makeColNames | Colnames included when TRUE (default). |
| crossTable | Cross table in output when TRUE |
| total | Vector of total codes (possibly recycled) passed to AutoHierarchies, with optional named vector/list support. |
| simplify | When TRUE (default) the model can be simplified when total codes are found in the hierarchies (see examples). |
| hierarchyVarNames | Variable names in the hierarchy tables as in HierarchyFix |
| unionComplement | Logical vector (possibly recycled) for each element of hierarchies. When TRUE, sign means union and complement instead of addition or subtraction. Values corresponding to "rowFactor" and "colFactor" are ignored. |
| removeEmpty | When TRUE, empty columns (only zeros) are not included in output. |
| reOrder | When TRUE (default) output codes are ordered in a way similar to a usual model matrix ordering. |
| sep | String to separate when creating column names |
| ... | Extra unused parameters |

## Value

A sparse model matrix or a list of two elements (model matrix and cross table)

## Author(s)

Øyvind Langsrud

## See Also

ModelMatrix, Hierarchies2ModelMatrix, Formula2ModelMatrix.

## Examples

```
# Create some input
z <- SSBtoolsData("sprt_emp_withEU")
ageHier <- SSBtoolsData("sprt_emp_ageHier")
geoDimList <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]
```

```
# Shorter function name
H <- HierarchiesAndFormula2ModelMatrix

# Small dataset example. Two dimensions.
s <- z[z$geo == "Spain", ]
geoYear <- list(geo = geoDimList, year = "")
m <- H(s, geoYear, ~geo * year, inputInOutput = c(FALSE, TRUE))
print(m, col.names = TRUE)
attr(m, "total")     # Total code 'Europe' is found
attr(m, "startCol")  # Two model terms needed

# Another model and with crossTable in output
H(s, geoYear, ~geo + year, crossTable = TRUE)

# Without empty columns
H(s, geoYear, ~geo + year, crossTable = TRUE, removeEmpty = TRUE)

# Three dimensions
ageGeoYear <- list(age = ageHier, geo = geoDimList, year = "allYears")
m <- H(z, ageGeoYear, ~age * geo + geo * year)
head(colnames(m))
attr(m, "total")
attr(m, "startCol")

# With simplify = FALSE
m <- H(z, ageGeoYear, ~age * geo + geo * year, simplify = FALSE)
head(colnames(m))
attr(m, "total")
attr(m, "startCol")

# Compute aggregates
m <- H(z, ageGeoYear, ~geo * age, inputInOutput = c(TRUE, FALSE, TRUE))
Matrix::t(m) %*% z$ths_per

# Without hierarchies. Only factors.
ageGeoYearFactor <- list(age = "", geo = "", year = "")
Matrix::t(H(z, ageGeoYearFactor, ~geo * age + year:geo))
```

---

`hierarchies_as_vars` *Hierarchies coded as variables*

---

## Description

The hierarchical relations are stored as minimal datasets

## Usage

```
hierarchies_as_vars(
  hierarchies,
```

```
  name_function = function(name, level) paste0(name, "_level_", level),
  single_vars = FALSE,
  from_dummy = NA,
  dummy_reorder = TRUE,
  combine_vars = TRUE,
  drop_codes = NULL,
  include_codes = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| hierarchies | List of hierarchies in the same format as input to AutoHierarchies |
| name_function | A function defining how to name all columns except the first. The input consists of the hierarchy name (identical to the first column's name, name) and the column number minus 1 (level). |
| single_vars | When TRUE, a single variable is created for all codes except the input codes. |
| from_dummy | Logical value indicating the method for handling hierarchies.<br>• When TRUE, the algorithm uses dummy-coded hierarchies.<br>• When FALSE, the algorithm works directly on hierarchies standardized by AutoHierarchies, often resulting in well-structured output variables.<br>• When NA (default), the algorithm first attempts the FALSE method; if not feasible, it falls back to the TRUE method. |
| dummy_reorder | When TRUE, dummy-coded hierarchies are reordered to potentially improve the structure of output variables. |
| combine_vars | When TRUE, an algorithm is applied to potentially reduce the number of output variables needed. |
| drop_codes | A named list of codes (except the input codes) to be dropped from the output. The list must have the same names as the hierarchies, but not all names/elements need to be included. |
| include_codes | Similar to drop_codes, but specifies the codes to be included instead. |
| ... | Additional parameters passed to AutoHierarchies |

## Value

Named list of data frames

## See Also

vars_to_hierarchies

## Examples

```
# Examples based on those from AutoHierarchies
# You may also try converting other examples from AutoHierarchies

z <- SSBtoolsData("sprt_emp_withEU")
```

```
year_formula <- c("y_14 = 2014", "y_15_16 = y_all - y_14", "y_all = 2014 + 2015 + 2016")
geo_dim_list <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]
age_hierarchy <- SSBtoolsData("sprt_emp_ageHier")

hierarchies_as_vars(list(age = age_hierarchy, geo = geo_dim_list, year = year_formula))
hierarchies_as_vars(list(age = age_hierarchy, geo = geo_dim_list, year = year_formula),
                    singleVars = TRUE)

# NAs are included in data when necessary
hierarchies_as_vars(list(f = c("AB = A + B", "AC = A + C", "CD = C + D", "ABCD = AB + CD")))

# drop_codes and include_codes
hierarchies_as_vars(list(age = age_hierarchy, geo = geo_dim_list, year = year_formula),
                    drop_codes = list(geo = "nonEU", year = c("y_14", "y_all")))
hierarchies_as_vars(list(age = age_hierarchy, geo = geo_dim_list, year = year_formula),
                    include_codes = list(year = c("y_14", "y_all")))
```

---

`Hierarchy2Formula` *Hierarchy2Formula*

---

## Description

Conversion between to-from coded hierarchy and formulas written with =, - and +.

## Usage

```
Hierarchy2Formula(
  x,
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level =
    "level")
)

Formula2Hierarchy(s)

Hierarchies2Formulas(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | Data frame with to-from coded hierarchy |
| `hierarchyVarNames` | Variable names in the hierarchy tables as in `HierarchyFix`. |
| `s` | Character vector of formulas written with =, - and +. |
| `...` | Extra parameters. Only `hierarchyVarNames` is relevant. |

## Value

See Arguments

## Note

Hierarchies2Formulas is a wrapper for lapply(x, Hierarchy2Formula, ...)

## Author(s)

Øyvind Langsrud

## See Also

DimList2Hierarchy, DimList2Hrc, AutoHierarchies.

## Examples

```
x <- SSBtoolsData("sprt_emp_geoHier")
s <- Hierarchy2Formula(x)
s
Formula2Hierarchy(s)

# Demonstrate Hierarchies2Formulas and problems
hi <- FindHierarchies(SSBtoolsData("sprt_emp_withEU")[, c("geo", "eu", "age")])
hi
Hierarchies2Formulas(hi) # problematic formula since minus sign in coding
AutoHierarchies(Hierarchies2Formulas(hi)) # Not same as hi because of problems

# Change coding to avoid problems
hi$age$mapsFrom <- gsub("-", "_", hi$age$mapsFrom)
hi
Hierarchies2Formulas(hi)
AutoHierarchies(Hierarchies2Formulas(hi))
```

---

# HierarchyCompute *Hierarchical Computations*

---

## Description

This function computes aggregates by crossing several hierarchical specifications and factorial variables.

## Usage

```
HierarchyCompute(
  data,
  hierarchies,
  valueVar,
  colVar = NULL,
  rowSelect = NULL,
  colSelect = NULL,
```

```
  select = NULL,
  inputInOutput = FALSE,
  output = "data.frame",
  autoLevel = TRUE,
  unionComplement = FALSE,
  constantsInOutput = NULL,
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level =
    "level"),
  selectionByMultiplicationLimit = 10^7,
  colNotInDataWarning = TRUE,
  useMatrixToDataFrame = TRUE,
  handleDuplicated = "sum",
  asInput = FALSE,
  verbose = FALSE,
  reOrder = FALSE,
  reduceData = TRUE,
  makeRownames = NULL
)
```

## Arguments

| | |
|---|---|
| data | The input data frame |
| hierarchies | A named (names in data) list with hierarchies. Variables can also be coded by "rowFactor" and "colFactor". |
| valueVar | Name of the variable(s) to be aggregated. |
| colVar | When non-NULL, the function HierarchyCompute2 is called. See its documentation for more information. |
| rowSelect | Data frame specifying variable combinations for output. The colFactor variable is not included. In addition rowSelect="removeEmpty" removes combinations corresponding to empty rows (only zeros) of dataDummyHierarchy. |
| colSelect | Vector specifying categories of the colFactor variable for output. |
| select | Data frame specifying variable combinations for output. The colFactor variable is included. |
| inputInOutput | Logical vector (possibly recycled) for each element of hierarchies. TRUE means that codes from input are included in output. Values corresponding to "rowFactor" and "colFactor" are ignored. |
| output | One of "data.frame" (default), "dummyHierarchies", "outputMatrix", "dataDummyHierarchy", "valueMatrix", "fromCrossCode", "toCrossCode", "crossCode" (as toCrossCode), "outputMatrixWithCrossCode", "matrixComponents", "dataDummyHierarchyWithCodeFrame", "dataDummyHierarchyQuick". The latter two do not require valueVar (reduceData set to FALSE). |
| autoLevel | Logical vector (possibly recycled) for each element of hierarchies. When TRUE, level is computed by automatic method as in HierarchyFix. Values corresponding to "rowFactor" and "colFactor" are ignored. |

`unionComplement`
: Logical vector (possibly recycled) for each element of hierarchies. When TRUE, sign means union and complement instead of addition or subtraction as in `DummyHierarchy`. Values corresponding to `"rowFactor"` and `"colFactor"` are ignored.

`constantsInOutput`
: A single row data frame to be combine by the other output.

`hierarchyVarNames`
: Variable names in the hierarchy tables as in `HierarchyFix`.

`selectionByMultiplicationLimit`
: With non-NULL `rowSelect` and when the number of elements in `dataDummyHierarchy` exceeds this limit, the computation is performed by a slower but more memory efficient algorithm.

`colNotInDataWarning`
: When TRUE, warning produced when elements of `colSelect` are not in data.

`useMatrixToDataFrame`
: When TRUE (default) special functionality for saving time and memory is used.

`handleDuplicated`
: Handling of duplicated code rows in data. One of: "sum" (default), "sumByAggregate", "sumWithWarning", "stop" (error), "single" or "singleWithWarning". With no colFactor sum and sumByAggregate/sumWithWarning are different (original values or aggregates in "valueMatrix"). When single, only one of the values is used (by matrix subsetting).

`asInput`
: When TRUE (FALSE is default) output matrices match input data. Thus `valueMatrix = Matrix(data[, valueVar],ncol=1)`. Only possible when no colFactor.

`verbose`
: Whether to print information during calculations. FALSE is default.

`reOrder`
: When TRUE (FALSE is default) output codes are ordered differently, more similar to a usual model matrix ordering.

`reduceData`
: When TRUE (default) unnecessary (for the aggregated result) rows of `valueMatrix` are allowed to be removed.

`makeRownames`
: When TRUE `dataDummyHierarchy` contains rownames. By default, this is decided based on the parameter `output`.

## Details

A key element of this function is the matrix multiplication: `outputMatrix = dataDummyHierarchy %*% valueMatrix`. The matrix, `valueMatrix` is a re-organized version of the valueVar vector from input. In particular, if a variable is selected as `colFactor`, there is one column for each level of that variable. The matrix, `dataDummyHierarchy` is constructed by crossing dummy coding of hierarchies (`DummyHierarchy`) and factorial variables in a way that matches `valueMatrix`. The code combinations corresponding to rows and columns of `dataDummyHierarchy` can be obtained as `toCrossCode` and `fromCrossCode`. In the default data frame output, the `outputMatrix` is stacked to one column and combined with the code combinations of all variables.

## Value

As specified by the parameter `output`

## Author(s)

Øyvind Langsrud

## See Also

Hierarchies2ModelMatrix, AutoHierarchies.

## Examples

```
# Data and hierarchies used in the examples
x <- SSBtoolsData("sprt_emp") # Employment in sport in thousand persons from Eurostat database
geoHier <- SSBtoolsData("sprt_emp_geoHier")
ageHier <- SSBtoolsData("sprt_emp_ageHier")

# Two hierarchies and year as rowFactor
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per")

# Same result with year as colFactor (but columns ordered differently)
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per")

# Internally the computations are different as seen when output='matrixComponents'
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per",
                 output = "matrixComponents")
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
                 output = "matrixComponents")


# Include input age groups by setting inputInOutput = TRUE for this variable
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
                 inputInOutput = c(TRUE, FALSE))

# Only input age groups by switching to rowFactor
HierarchyCompute(x, list(age = "rowFactor", geo = geoHier, year = "colFactor"), "ths_per")

# Select some years (colFactor) including a year not in input data (zeros produced)
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
                 colSelect = c("2014", "2016", "2018"))

# Select combinations of geo and age including a code not in data or hierarchy (zeros produced)
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
               rowSelect = data.frame(geo = "EU", age = c("Y0-100", "Y15-64", "Y15-29")))

# Select combinations of geo, age and year
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
    select = data.frame(geo = c("EU", "Spain"), age = c("Y15-64", "Y15-29"), year = 2015))

# Extend the hierarchy table to illustrate the effect of unionComplement
# Omit level since this is handled by autoLevel
geoHier2 <- rbind(data.frame(mapsFrom = c("EU", "Spain"), mapsTo = "EUandSpain", sign = 1),
                  geoHier[, -4])

# Spain is counted twice
```

```
HierarchyCompute(x, list(age = ageHier, geo = geoHier2, year = "colFactor"), "ths_per")

# Can be seen in the dataDummyHierarchy matrix
HierarchyCompute(x, list(age = ageHier, geo = geoHier2, year = "colFactor"), "ths_per",
                 output = "matrixComponents")

# With unionComplement=TRUE Spain is not counted twice
HierarchyCompute(x, list(age = ageHier, geo = geoHier2, year = "colFactor"), "ths_per",
                 unionComplement = TRUE)

# With constantsInOutput
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "colFactor"), "ths_per",
                 constantsInOutput = data.frame(c1 = "AB", c2 = "CD"))

# More that one valueVar
x$y <- 10*x$ths_per
HierarchyCompute(x, list(age = ageHier, geo = geoHier), c("y", "ths_per"))
```

---

`HierarchyCompute2` *Extended Hierarchical Computations*

---

## Description

Extended variant of `HierarchyCompute` with several column variables (not just `"colFactor"`). Parameter colVar splits the hierarchy variables in two groups and this variable overrides the difference between `"rowFactor"` and `"colFactor"`.

## Usage

```
HierarchyCompute2(
  data,
  hierarchies,
  valueVar,
  colVar,
  rowSelect = NULL,
  colSelect = NULL,
  select = NULL,
  output = "data.frame",
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | The input data frame |
| `hierarchies` | A named list with hierarchies |
| `valueVar` | Name of the variable(s) to be aggregated |
| `colVar` | Name of the column variable(s) |

| | |
|---|---|
| rowSelect | Data frame specifying variable combinations for output |
| colSelect | Data frame specifying variable combinations for output |
| select | Data frame specifying variable combinations for output |
| output | One of "data.frame" (default), "outputMatrix", "matrixComponents". |
| ... | Further parameters sent to HierarchyCompute |

## Details

Within this function, HierarchyCompute is called two times. By specifying output as "matrixComponents", output from the two runs are retuned as a list with elements hcRow and hcCol. The matrix multiplication in HierarchyCompute is extended to outputMatrix = hcRow$dataDummyHierarchy %*% hcRow$valueMatrix %*% t(hcCol$dataDummyHierarchy). This is modified in cases with more than a single valueVar.

## Value

As specified by the parameter output

## Note

There is no need to call HierarchyCompute2 directly. The main function HierarchyCompute can be used instead.

## Author(s)

Øyvind Langsrud

## See Also

Hierarchies2ModelMatrix, AutoHierarchies.

## Examples

```
x <- SSBtoolsData("sprt_emp")
geoHier <- SSBtoolsData("sprt_emp_geoHier")
ageHier <- SSBtoolsData("sprt_emp_ageHier")

HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per",
                 colVar = c("age", "year"))
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per",
                 colVar = c("age", "geo"))
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per",
                 colVar = c("age", "year"), output = "matrixComponents")
HierarchyCompute(x, list(age = ageHier, geo = geoHier, year = "rowFactor"), "ths_per",
                 colVar = c("age", "geo"), output = "matrixComponents")
```

LSfitNonNeg *Non-negative regression fits with a sparse overparameterized model matrix*

## Description

Assuming `z = t(x) %*% y + noise`, a non-negatively modified least squares estimate of `t(x) %*% y` is made.

## Usage

```
LSfitNonNeg(x, z, limit = 1e-10, viaQR = FALSE, printInc = TRUE)
```

## Arguments

| | |
|---|---|
| x | A matrix |
| z | A single column matrix |
| limit | Lower limit for non-zero fits. Set to NULL or -Inf to avoid the non-zero restriction. |
| viaQR | Least squares fits obtained using qr when TRUE. |
| printInc | Printing "..." to console when TRUE. |

## Details

The problem is first reduced by elimination some rows of x (elements of y) using GaussIndependent. Thereafter least squares fits are obtained using solve or qr. Possible negative fits will be forced to zero in the next estimation iteration(s).

## Value

A fitted version of z

## Author(s)

Øyvind Langsrud

## Examples

```
set.seed(123)
data2 <- SSBtoolsData("z2")
x <- ModelMatrix(data2, formula = ~fylke + kostragr * hovedint - 1)
z <- Matrix::t(x) %*% data2$ant + rnorm(ncol(x), sd = 3)
LSfitNonNeg(x, z)
LSfitNonNeg(x, z, limit = NULL)

## Not run:
mf <- ~region*mnd + hovedint*mnd + fylke*hovedint*mnd + kostragr*hovedint*mnd
```

```
data4 <- SSBtoolsData("sosialFiktiv")
x <- ModelMatrix(data4, formula = mf)
z <- Matrix::t(x) %*% data4$ant + rnorm(ncol(x), sd = 3)
zFit <- LSfitNonNeg(x, z)

## End(Not run)
```

---

`MakeHierFormula` *Make model formula from data taking into account hierarchical variables*

---

## Description

Make model formula from data taking into account hierarchical variables

## Usage

```
MakeHierFormula(
  data = NULL,
  hGroups = HierarchicalGroups2(data),
  n = length(hGroups),
  sim = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | data frame |
| `hGroups` | Output from HierarchicalGroups2() |
| `n` | Interaction level or 0 (all levels) |
| `sim` | Include "~" when TRUE |

## Value

Formula as character string

## Author(s)

Øyvind Langsrud

## Examples

```
x <- SSBtoolsData("sprt_emp_withEU")[, -4]
MakeHierFormula(x)
MakeHierFormula(x, n = 2)
MakeHierFormula(x, n = 0)
```

`map_hierarchies_to_data` *Add variables to dataset based on hierarchies*

## Description

Uses `hierarchies_as_vars` to transform hierarchies, followed by mapping to the dataset.

## Usage

```
map_hierarchies_to_data(
  data,
  hierarchies,
  when_overwritten = warning,
  add_comment = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing variables with names matching the names of the hierarchies. |
| `hierarchies` | List of hierarchies in the same format as input to `AutoHierarchies` |
| `when_overwritten` | A function to be called when existing column(s) are overwritten. Supply `stop` to invoke an error, `warning` for a warning (default), `message` to display an informational message, or `NULL` to do nothing. |
| `add_comment` | Logical. When `TRUE` (default), a comment attribute will be added to the output data frame, containing the names of the variables that were added. |
| `...` | Further parameters sent to `hierarchies_as_vars` |

## Value

Input `data` with extra Variables

## Examples

```
# Examples similar those from hierarchies_as_vars

z <- SSBtoolsData("sprt_emp_withEU")
year_formula <- c("y_14 = 2014", "y_15_16 = y_all - y_14", "y_all = 2014 + 2015 + 2016")
geo_dim_list <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]
age_hierarchy <- SSBtoolsData("sprt_emp_ageHier")

map_hierarchies_to_data(z, list(age = age_hierarchy, geo = geo_dim_list,
                                year = year_formula))
```

```
map_hierarchies_to_data(data.frame(f = c("A", "B", "C", "D", "E", "A")), list(f =
       c("AB = A + B", "AC = A + C", "CD = C + D", "ABCD = AB + CD")))


# Examples demonstrating when_overwritten and add_comment

a <- map_hierarchies_to_data(z, list(age = age_hierarchy, geo = geo_dim_list))
comment(a)

b <- map_hierarchies_to_data(a[-7], list(age = age_hierarchy, geo = geo_dim_list),
                             when_overwritten = message, add_comment = FALSE)
comment(b)
```

---

`Match` *Matching rows in data frames*

---

## Description

The algorithm is based on converting variable combinations to whole numbers. The final matching is performed using `match`.

## Usage

```
Match(x, y)
```

## Arguments

| | |
|---|---|
| x | data frame |
| y | data frame |

## Details

When the result of multiplying together the number of unique values in each column of x exceeds 9E15 (largest value stored exactly by the numeric data type), the algorithm is recursive.

## Value

An integer vector giving the position in y of the first match if there is a match, otherwise NA.

## Author(s)

Øyvind Langsrud

## Examples

```
a <- data.frame(x = c("a", "b", "c"), y = c("A", "B"), z = 1:6)
b <- data.frame(x = c("b", "c"), y = c("B", "K", "A", "B"), z = c(2, 3, 5, 6))

Match(a, b)
Match(b, a)

# Slower alternative
match(data.frame(t(a), stringsAsFactors = FALSE), data.frame(t(b), stringsAsFactors = FALSE))
match(data.frame(t(b), stringsAsFactors = FALSE), data.frame(t(a), stringsAsFactors = FALSE))

# More comprehensive example (n, m and k may be changed)
n <- 10^4
m <- 10^3
k <- 10^2
data(precip)
data(mtcars)
y <- data.frame(car = sample(rownames(mtcars), n, replace = TRUE),
                city = sample(names(precip), n, replace = TRUE),
                n = rep_len(1:k, n), a = rep_len(c("A", "B", "C", "D"), n),
                b = rep_len(as.character(rnorm(1000)), n),
                d = sample.int(k + 10, n, replace = TRUE),
                e = paste(sample.int(k * 2, n, replace = TRUE),
                          rep_len(c("Green", "Red", "Blue"), n), sep = "_"),
                r = rnorm(k)^99)
x <- y[sample.int(n, m), ]
row.names(x) <- NULL
ix <- Match(x, y)
```

---

`matlabColon` *Simulate Matlab's ':'*

---

## Description

Functions to generate increasing sequences

## Usage

```
matlabColon(from, to)

SeqInc(from, to)
```

## Arguments

| | |
|---|---|
| `from` | numeric. The start value |
| `to` | numeric. The end value. |

## Details

matlabColon(a,b) returns a:b (R's version) unless a > b, in which case it returns integer(0). SeqInc(a,b) is similar, but results in error when the calculated length of the sequence (1+to-from) is negative.

## Value

A numeric vector, possibly empty.

## Author(s)

Bjørn-Helge Mevik (matlabColon) and Øyvind Langsrud (SeqInc)

## See Also

seq

## Examples

```
identical(3:5, matlabColon(3, 5)) ## => TRUE
3:1 ## => 3 2 1
matlabColon(3, 1) ## => integer(0)
try(SeqInc(3, 1)) ## => Error
SeqInc(3, 2)      ## => integer(0)
```

---

# Matrix2list *Convert matrix to sparse list*

---

## Description

Convert matrix to sparse list

## Usage

```
Matrix2list(x)

Matrix2listInt(x)
```

## Arguments

| | |
|---|---|
| x | Input matrix |

## Details

Within the function, the input matrix is first converted to a dgTMatrix matrix (Matrix package).

## Value

A two-element list: List of row numbers (r) and a list of numeric or integer values (x)

## Note

`Matrix2listInt` convers the values to integers by `as.integer` and no checking is performed. Thus, zeros are possible.

## Author(s)

Øyvind Langsrud

## Examples

```
m = matrix(c(0.5, 1.1, 3.14, 0, 0, 0, 0, 4, 5), 3, 3)
Matrix2list(m)
Matrix2listInt(m)
```

---

# max_contribution *Find Major Contributions to Aggregates and Count Contributors*

---

## Description

These functions analyze contributions to aggregates, assuming that the aggregates are calculated using a dummy matrix with the formula: `z = t(x) %*% y`.

## Usage

```
max_contribution(
  x,
  y,
  n = 1,
  id = NULL,
  output = "y",
  drop = TRUE,
  decreasing = TRUE,
  remove_fraction = NULL,
  do_abs = TRUE
)

n_contributors(x, y = rep(1L, nrow(x)), id = NULL, output = "n_contr", ...)
```

## Arguments

| | |
|---|---|
| `x` | A (sparse) dummy matrix |
| `y` | A numeric vector of input values (contributions). |
| `n` | Integer. The number of largest contributors to identify for each aggregate. Default is 1. |
| `id` | An optional vector for grouping. When non-NULL, major contributions are found after aggregation within each group specified by `id`. Aggregates with missing `id` values are excluded. |

`output` A character vector specifying the desired output. Possible values:

- `"y"`: A matrix with the largest contributions in the first column, the second largest in the second column, and so on.
- `"id"`: A matrix of IDs associated with the largest contributions. If an `id` vector is provided, it returns these IDs; otherwise, it returns indices.
- `"n_contr"`: An integer vector indicating the number of contributors to each aggregate.
- `"n_0_contr"`: An integer vector indicating the number of contributors that contribute a value of 0 to each aggregate.
- `"n_non0_contr"`: An integer vector indicating the number of contributors that contribute a nonzero value to each aggregate.
- `"sums"`: A numeric vector containing the aggregate sums of `y`.
- `"n_contr_all"`, `"n_0_contr_all"`, `"n_non0_contr_all"`, `"sums_all"`: Same as the corresponding outputs above, but without applying the `remove_fraction` parameter.

`drop` Logical. If TRUE (default) and `output` has length 1, the function returns the single list element directly instead of a list containing one element.

`decreasing` Logical. If TRUE (default), finds the largest contributors. If FALSE, finds the smallest contributors.

`remove_fraction`
A numeric vector containing values in the interval `[0, 1]`, specifying contributors to be removed when identifying the largest contributions.

- If an `id` vector is provided, `remove_fraction` must be named according to the IDs of the contributors to be removed.
- If no `id` vector is provided, the length of `remove_fraction` must match the length of `y`. In this case, contributors not to be removed should have a value of `NA` in `remove_fraction`.
- The actual values in `remove_fraction` are used for calculating `"sums"` (see description above).

`do_abs` Logical. If TRUE (default), uses the absolute values of the summed contributions. The summation is performed for all contributions from the same contributor, within each aggregate being computed.

`...` Further arguments to `max_contribution` (used by `n_contributors`).

## Details

The `max_contribution` function identifies the largest contributions to these aggregates, while the wrapper function `n_contributors` is designed specifically to count the number of contributors for each aggregate.

## Value

A list or a single element, depending on the values of the `output` and `drop` parameters.

## Examples

```
z <- SSBtoolsData("magnitude1")
a <- ModelMatrix(z, formula = ~sector4 + geo, crossTable = TRUE)

cbind(a$crossTable,
      y =  max_contribution(x = a$modelMatrix, y = z$value, n = 2),
      id = max_contribution(x = a$modelMatrix, y = z$value, n = 2, output = "id"),
      n =  n_contributors(  x = a$modelMatrix, y = z$value, n = 2))

cbind(a$crossTable,
      y = max_contribution(x = a$modelMatrix, y = z$value, n = 3, id = z$company),
      id = max_contribution(a$modelMatrix, z$value, 3, id = z$company, output = "id"))

max_contribution(x = a$modelMatrix,
                 y = z$value,
                 n = 3,
                 id = z$company,
                 output = c("y", "id", "n_contr", "sums"))

as.data.frame(
  max_contribution(x = a$modelMatrix,
                   y = z$value,
                   n = 3,
                   id = z$company,
                   output = c("y", "id", "n_contr", "sums", "n_contr_all", "sums_all"),
                   remove_fraction = c(B = 1)))
```

---

# Mipf *Iterative proportional fitting from matrix input*

---

## Description

The linear equation, `z = t(x) %*% y`, is (hopefully) solved for `y` by iterative proportional fitting

## Usage

```
Mipf(
  x,
  z = NULL,
  iter = 100,
  yStart = matrix(1, nrow(x), 1),
  eps = 0.01,
  tol = 1e-10,
  reduceBy0 = FALSE,
  reduceByColSums = FALSE,
  reduceByLeverage = FALSE,
  returnDetails = FALSE,
  y = NULL
)
```

## Arguments

| | |
|---|---|
| x | a matrix |
| z | a single column matrix |
| iter | maximum number of iterations |
| yStart | a starting estimate of y |
| eps | stopping criterion. Maximum allowed value of max(abs(z - t(x) %*% yHat)) |
| tol | Another stopping criterion. Maximum absolute difference between two iterations. |
| reduceBy0 | When TRUE, Reduce0exact used within the function |
| reduceByColSums | Parameter to Reduce0exact (when TRUE) |
| reduceByLeverage | Parameter to Reduce0exact (when TRUE) |
| returnDetails | More output when TRUE. |
| y | It is possible to set z to NULL and supply original y instead (z = t(x) %*% y) |

## Details

The algorithm will work similar to loglin when the input x-matrix is a overparameterized model matrix – as can be created by ModelMatrix and FormulaSums. See Examples.

## Value

yHat, the estimate of y

## Author(s)

Øyvind Langsrud

## Examples

```
## Not run:
data2 <- SSBtoolsData("z2")
x <- ModelMatrix(data2, formula = ~fylke + kostragr * hovedint - 1)
z <- Matrix::t(x) %*% data2$ant  # same as FormulaSums(data2, ant~fylke + kostragr * hovedint -1)
yHat <- Mipf(x, z)

############################
# loglm comparison
############################

if (require(MASS)){

# Increase accuracy
yHat <- Mipf(x, z, eps = 1e-04)

# Run loglm and store fitted values in a data frame
```

```
outLoglm <- loglm(ant ~ fylke + kostragr * hovedint, data2, eps = 1e-04, iter = 100)
dfLoglm <- as.data.frame.table(fitted(outLoglm))

# Problem 1: Variable region not in output, but instead the variable .Within.
# Problem 2: Extra zeros since hierarchy not treated. Impossible combinations in output.

# By sorting data, it becomes clear that the fitted values are the same.
max(abs(sort(dfLoglm$Freq, decreasing = TRUE)[1:nrow(data2)] - sort(yHat, decreasing = TRUE)))

# Modify so that region is in output. Problem 1 avoided.
x <- ModelMatrix(data2, formula = ~region + kostragr * hovedint - 1)
z <- Matrix::t(x) %*% data2$ant  # same as FormulaSums(data2, ant~fylke + kostragr * hovedint -1)
yHat <- Mipf(x, z, eps = 1e-04)
outLoglm <- loglm(ant ~ region + kostragr * hovedint, data2, eps = 1e-04, iter = 100)
dfLoglm <- as.data.frame.table(fitted(outLoglm))

# Now it is possible to merge data
merg <- merge(cbind(data2, yHat), dfLoglm)

# Identical output
max(abs(merg$yHat - merg$Freq))

}

## End(Not run)

############################
# loglin comparison
############################


# Generate input data for loglin
n <- 5:9
tab <- array(sample(1:prod(n)), n)

# Input parameters
iter <- 20
eps <- 1e-05

# Estimate yHat by loglin
out <- loglin(tab, list(c(1, 2), c(1, 3), c(1, 4), c(1, 5), c(2, 3, 4), c(3, 4, 5)),
              fit = TRUE, iter = iter, eps = eps)
yHatLoglin <- matrix(((out$fit)), ncol = 1)

# Transform the data for input to Mipf
df <- as.data.frame.table(tab)
names(df)[1:5] <- c("A", "B", "C", "D", "E")
x <- ModelMatrix(df, formula = ~A:B + A:C + A:D + A:E + B:C:D + C:D:E - 1)
z <- Matrix::t(x) %*% df$Freq

# Estimate yHat by Mipf
yHatPMipf <- Mipf(x, z, iter = iter, eps = eps)
```

```
# Maximal absolute difference
max(abs(yHatPMipf - yHatLoglin))

# Note: loglin reports one iteration extra

# Another example. Only one iteration needed.
max(abs(Mipf(x = FormulaSums(df, ~A:B + C - 1),
             z = FormulaSums(df, Freq ~ A:B + C -1))
             - matrix(loglin(tab, list(1:2, 3), fit = TRUE)$fit, ncol = 1)))


########################################
# Examples utilizing Reduce0exact
########################################

z3 <- SSBtoolsData("z3")
x <- ModelMatrix(z3, formula = ~region + kostragr * hovedint + region * mnd2 + fylke * mnd +
                     mnd * hovedint + mnd2 * fylke * hovedint - 1)

# reduceBy0, but no iteration improvement. Identical results.
t <- 360
y <- z3$ant
y[round((1:t) * 432/t)] <- 0
z <- Matrix::t(x) %*% y
a1 <- Mipf(x, z, eps = 0.1)
a2 <- Mipf(x, z, reduceBy0 = TRUE, eps = 0.1)
a3 <- Mipf(x, z, reduceByColSums = TRUE, eps = 0.1)
max(abs(a1 - a2))
max(abs(a1 - a3))


## Not run:
# Improvement by reduceByColSums. Changing eps and iter give more similar results.
t <- 402
y <- z3$ant
y[round((1:t) * 432/t)] <- 0
z <- Matrix::t(x) %*% y
a1 <- Mipf(x, z, eps = 1)
a2 <- Mipf(x, z, reduceBy0 = TRUE, eps = 1)
a3 <- Mipf(x, z, reduceByColSums = TRUE, eps = 1)
max(abs(a1 - a2))
max(abs(a1 - a3))


# Improvement by ReduceByLeverage. Changing eps and iter give more similar results.
t <- 378
y <- z3$ant
y[round((1:t) * 432/t)] <- 0
z <- Matrix::t(x) %*% y
a1 <- Mipf(x, z, eps = 1)
a2 <- Mipf(x, z, reduceBy0 = TRUE, eps = 1)
a3 <- Mipf(x, z, reduceByColSums = TRUE, eps = 1)
a4 <- Mipf(x, z, reduceByLeverage = TRUE, eps = 1)
```

```
max(abs(a1 - a2))
max(abs(a1 - a3))
max(abs(a1 - a4))


# Example with small eps and "Iteration stopped since tol reached"
t <- 384
y <- z3$ant
y[round((1:t) * 432/t)] <- 0
z <- Matrix::t(x) %*% y
a1 <- Mipf(x, z, eps = 1e-14)
a2 <- Mipf(x, z, reduceBy0 = TRUE, eps = 1e-14)
a3 <- Mipf(x, z, reduceByColSums = TRUE, eps = 1e-14)
max(abs(a1 - a2))
max(abs(a1 - a3))

## End(Not run)

# All y-data found by reduceByColSums (0 iterations).
t <- 411
y <- z3$ant
y[round((1:t) * 432/t)] <- 0
z <- Matrix::t(x) %*% y
a1 <- Mipf(x, z)
a2 <- Mipf(x, z, reduceBy0 = TRUE)
a3 <- Mipf(x, z, reduceByColSums = TRUE)
max(abs(a1 - y))
max(abs(a2 - y))
max(abs(a3 - y))
```

---

ModelMatrix *Model matrix from hierarchies and/or a formula*

---

## Description

A common interface to Hierarchies2ModelMatrix, Formula2ModelMatrix and HierarchiesAndFormula2ModelMatrix

## Usage

```
ModelMatrix(
  data,
  hierarchies = NULL,
  formula = NULL,
  inputInOutput = TRUE,
  crossTable = FALSE,
  sparse = TRUE,
  viaOrdinary = FALSE,
  total = "Total",
  removeEmpty = !is.null(formula) & is.null(hierarchies),
```

```
  modelMatrix = NULL,
  dimVar = NULL,
  select = NULL,
  ...
)

NamesFromModelMatrixInput(
  data = NULL,
  hierarchies = NULL,
  formula = NULL,
  dimVar = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Matrix or data frame with data containing codes of relevant variables |
| hierarchies | List of hierarchies, which can be converted by AutoHierarchies. Thus, the variables can also be coded by "rowFactor" or "", which correspond to using the categories in the data. |
| formula | A model formula |
| inputInOutput | Logical vector (possibly recycled) for each element of hierarchies. TRUE means that codes from input are included in output. Values corresponding to "rowFactor" or "" are ignored. |
| crossTable | Cross table in output when TRUE |
| sparse | Sparse matrix in output when TRUE (default) |
| viaOrdinary | When TRUE, output is generated by model.matrix or sparse.model.matrix. Since these functions omit a factor level, an empty factor level is first added. |
| total | String(s) used to name totals, passed to underlying functions. Named vectors or lists are supported. See FindDimLists(), Formula2ModelMatrix() and AutoHierarchies(). |
| removeEmpty | When TRUE, empty columns (only zeros) are not included in output. Default is TRUE with formula input without hierarchy and otherwise FALSE (see details). |
| modelMatrix | The model matrix as input (same as output) |
| dimVar | The main dimensional variables and additional aggregating variables. This parameter can be useful when hierarchies and formula are unspecified. |
| select | Data frame specifying variable combinations for output or a named list specifying code selections for each variable (see details). |
| ... | Further arguments to Hierarchies2ModelMatrix, Formula2ModelMatrix or HierarchiesAndFormula2ModelMatrix |

## Details

The default value of removeEmpty corresponds to the default settings of the underlying functions. The functions Hierarchies2ModelMatrix and HierarchiesAndFormula2ModelMatrix

have removeEmpty as an explicit parameter with FALSE as default. The function Formula2ModelMatrix is a wrapper for FormulaSums, which has a parameter includeEmpty with FALSE as default. Thus, ModelMatrix makes a call to Formula2ModelMatrix with includeEmpty = !removeEmpty.

NamesFromModelMatrixInput returns the names of the data columns involved in creating the model matrix. Note that data must be non-NULL to convert dimVar as indices to names.

The select parameter is forwarded to Hierarchies2ModelMatrix unless removeEmpty = TRUE is combined with select as a data frame. In all other cases, select is handled outside the underlying functions by making selections in the result. Empty columns can be added to the model matrix when removeEmpty = FALSE (with warning).

## Value

A (sparse) model matrix or a list of two elements (model matrix and cross table)

## Author(s)

Øyvind Langsrud

## See Also

formula_utils

## Examples

```
# Create some input
z <- SSBtoolsData("sp_emp_withEU")
ageHier <- data.frame(mapsFrom = c("young", "old"), mapsTo = "Total", sign = 1)
geoDimList <- FindDimLists(z[, c("geo", "eu")], total = "Europe")[[1]]

# Small dataset example. Two dimensions.
s <- z[z$geo == "Spain" & z$year != 2016, ]
rownames(s) <- NULL
s

# via Hierarchies2ModelMatrix() and converted to ordinary matrix (not sparse)
ModelMatrix(s, list(age = ageHier, year = ""), sparse = FALSE)

# Hierarchies generated automatically. Then via Hierarchies2ModelMatrix()
ModelMatrix(s[, c(1, 4)])

# via Formula2ModelMatrix()
ModelMatrix(s, formula = ~age + year)

# via model.matrix() after adding empty factor levels
ModelMatrix(s, formula = ~age + year, sparse = FALSE, viaOrdinary = TRUE)

# via sparse.model.matrix() after adding empty factor levels
ModelMatrix(s, formula = ~age + year, viaOrdinary = TRUE)

# via HierarchiesAndFormula2ModelMatrix() and using different data and parameter settings
ModelMatrix(s, list(age = ageHier, geo = geoDimList, year = ""), formula = ~age * geo + year,
```

```
            inputInOutput = FALSE, removeEmpty = TRUE, crossTable = TRUE)
ModelMatrix(s, list(age = ageHier, geo = geoDimList, year = ""), formula = ~age * geo + year,
            inputInOutput = c(TRUE, FALSE), removeEmpty = FALSE, crossTable = TRUE)
ModelMatrix(z, list(age = ageHier, geo = geoDimList, year = ""), formula = ~age * year + geo,
            inputInOutput = c(FALSE, TRUE), crossTable = TRUE)

# via Hierarchies2ModelMatrix() using unnamed list element. See AutoHierarchies.
colnames(ModelMatrix(z, list(age = ageHier, c(Europe = "geo", Allyears = "year", "eu"))))
colnames(ModelMatrix(z, list(age = ageHier, c("geo", "year", "eu")), total = c("t1", "t2")))

# Example using the select parameter as a data frame
select <- data.frame(age = c("Total", "young", "old"), geo = c("EU", "nonEU", "Spain"))
ModelMatrix(z, list(age = ageHier, geo = geoDimList),
            select = select, crossTable = TRUE)$crossTable

# Examples using the select parameter as a list
ModelMatrix(z, list(age = ageHier, geo = geoDimList), inputInOutput = FALSE,
            select = list(geo = c("nonEU", "Portugal")), crossTable = TRUE)$crossTable
ModelMatrix(z, list(age = ageHier, geo = geoDimList),
            select = list(geo = c("nonEU", "Portugal"), age = c("Total", "young")),
            crossTable = TRUE)$crossTable

# Using NAomit parameter avalable in Formula2ModelMatrix()
s$age[1] <- NA
ModelMatrix(s, formula = ~age + year)
ModelMatrix(s, formula = ~age + year, NAomit = FALSE)
```

---

`model_aggregate` *Hierarchical aggregation via model specification*

---

## Description

Internally a dummy/model matrix is created according to the model specification. This model matrix is used in the aggregation process via matrix multiplication and/or the function `aggregate_multiple_fun`.

## Usage

```
model_aggregate(
  data,
  sum_vars = NULL,
  fun_vars = NULL,
  fun = NULL,
  hierarchies = NULL,
  formula = NULL,
  dim_var = NULL,
  total = NULL,
  input_in_output = NULL,
  remove_empty = NULL,
```

```
  avoid_hierarchical = NULL,
  preagg_var = NULL,
  dummy = TRUE,
  pre_aggregate = dummy,
  aggregate_pkg = "base",
  aggregate_na = TRUE,
  aggregate_base_order = FALSE,
  list_return = FALSE,
  pre_return = FALSE,
  verbose = TRUE,
  mm_args = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Input data containing data to be aggregated, typically a data frame, tibble, or data.table. If data is not a classic data frame, it will be coerced to one internally. |
| sum_vars | Variables to be summed. This will be done via matrix multiplication. |
| fun_vars | Variables to be aggregated by supplied functions. This will be done via aggregate_multiple_fun and dummy_aggregate and fun_vars is specified as the parameter vars. |
| fun | The fun parameter to aggregate_multiple_fun |
| hierarchies | The hierarchies parameter to ModelMatrix |
| formula | The formula parameter to ModelMatrix |
| dim_var | The dimVar parameter to ModelMatrix |
| total | When non-NULL, the total parameter to ModelMatrix. Thus, the actual default value is "Total". |
| input_in_output | When non-NULL, the inputInOutput parameter to ModelMatrix. Thus, the actual default value is TRUE. |
| remove_empty | When non-NULL, the removeEmpty parameter to ModelMatrix. Thus, the actual default value is TRUE with formula input without hierarchy and otherwise FALSE (see ModelMatrix). |
| avoid_hierarchical | When non-NULL, the avoidHierarchical parameter to Formula2ModelMatrix, which is an underlying function of ModelMatrix. |
| preagg_var | Extra variables to be used as grouping elements in the pre-aggregate step |
| dummy | The dummy parameter to dummy_aggregate. When TRUE, only 0s and 1s are assumed in the generated model matrix. When FALSE, non-0s in this matrix are passed as an additional first input parameter to the fun functions. |
| pre_aggregate | Whether to pre-aggregate data to reduce the dimension of the model matrix. Note that all original fun_vars observations are retained in the aggregated dataset and pre_aggregate does not affect the final result. However, pre_aggregate must be set to FALSE when the dummy_aggregate parameter dummy is set to FALSE since then unlist will not be run. An exception to this is if the fun functions are written to handle list data. |

| | |
|---|---|
| aggregate_pkg | Package used to pre-aggregate. Parameter pkg to aggregate_by_pkg. |
| aggregate_na | Whether to include NAs in the grouping variables while preAggregate. Parameter include_na to aggregate_by_pkg. |
| aggregate_base_order | Parameter base_order to aggregate_by_pkg, used when pre-aggregate. The default is set to FALSE to avoid unnecessary sorting operations. When TRUE, an attempt is made to return the same result with data.table as with base R. This cannot be guaranteed due to potential variations in sorting behavior across different systems. |
| list_return | Whether to return a list of separate components including the model matrix x. |
| pre_return | Whether to return the pre-aggregate data as a two-component list. Can also be combined with list_return (see examples). |
| verbose | Whether to print information during calculations. |
| mm_args | List of further arguments passed to ModelMatrix. |
| ... | Further arguments passed to dummy_aggregate. |

## Details

With formula input, limited output can be achieved by formula_selection (see example). An attribute called startCol has been added to the output data frame to make this functionality work.

## Value

A data frame or a list.

## Examples

```
z <- SSBtoolsData("sprt_emp_withEU")
z$age[z$age == "Y15-29"] <- "young"
z$age[z$age == "Y30-64"] <- "old"
names(z)[names(z) == "ths_per"] <- "ths"
z$y <- 1:18

my_range <- function(x) c(min = min(x), max = max(x))

out <- model_aggregate(z,
   formula = ~age:year + geo,
   sum_vars = c("y", "ths"),
   fun_vars = c(sum = "ths", mean = "y", med = "y", ra = "ths"),
   fun = c(sum = sum, mean = mean, med = median, ra = my_range))

out

# Limited output can be achieved by formula_selection
formula_selection(out, ~geo)


# Using the single unnamed variable feature.
model_aggregate(z, formula = ~age, fun_vars = "y",
```

```
                fun = c(sum = sum, mean = mean, med = median, n = length))


# To illustrate list_return and pre_return
for (pre_return in c(FALSE, TRUE)) for (list_return in c(FALSE, TRUE)) {
  cat("\n=======================================\n")
  cat("list_return =", list_return, ", pre_return =", pre_return, "\n\n")
  out <- model_aggregate(z, formula = ~age:year,
                         sum_vars = c("ths", "y"),
                         fun_vars = c(mean = "y", ra = "y"),
                         fun = c(mean = mean, ra = my_range),
                         list_return = list_return,
                         pre_return = pre_return)
  cat("\n")
  print(out)
}


# To illustrate preagg_var
model_aggregate(z, formula = ~age:year,
sum_vars = c("ths", "y"),
fun_vars = c(mean = "y", ra = "y"),
fun = c(mean = mean, ra = my_range),
preagg_var = "eu",
pre_return = TRUE)[["pre_data"]]


# To illustrate hierarchies
geo_hier <- SSBtoolsData("sprt_emp_geoHier")
model_aggregate(z, hierarchies = list(age = "All", geo = geo_hier),
                sum_vars = "y",
                fun_vars = c(sum = "y"))

####  Special non-dummy cases illustrated below  ####

# Extend the hierarchy to make non-dummy model matrix
geo_hier2 <- rbind(data.frame(mapsFrom = c("EU", "Spain"),
                              mapsTo = "EUandSpain", sign = 1), geo_hier[, -4])

# Warning since non-dummy
# y and y_sum are different
model_aggregate(z, hierarchies = list(age = "All", geo = geo_hier2),
                sum_vars = "y",
                fun_vars = c(sum = "y"))

# No warning since dummy since unionComplement = TRUE (see ?HierarchyCompute)
# y and y_sum are equal
model_aggregate(z, hierarchies = list(age = "All", geo = geo_hier2),
                sum_vars = "y",
                fun_vars = c(sum = "y"),
                mm_args = list(unionComplement = TRUE))

# Non-dummy again, but no warning since dummy = FALSE
```

```
# Then pre_aggregate is by default set to FALSE (error when TRUE)
# fun with extra argument needed (see ?dummy_aggregate)
# y and y_sum2 are equal
model_aggregate(z, hierarchies = list(age = "All", geo = geo_hier2),
                sum_vars = "y",
                fun_vars = c(sum2 = "y"),
                fun = c(sum2 = function(x, y) sum(x * y)),
                dummy = FALSE)
```

---

`Number` *Adding leading zeros*

---

## Description

Adding leading zeros

## Usage

```
Number(n, width = 3)
```

## Arguments

| | |
|---|---|
| `n` | numeric vector of whole numbers |
| `width` | width |

## Value

Character vector

## Author(s)

Øyvind Langsrud

## Examples

```
Number(1:3)
```

`NumSingleton` *Decoding of* `singletonMethod`

## Description

A `GaussSuppression` `singletonMethod` starting with `"num"` is decoded into separate characters. Part of the theory for interpreting the 3rd, 4th, and 5th characters is discussed in Langsrud (2024). To utilize possibly duplicated contributor IDs, the 2nd character must be `"T"`.

## Usage

```
NumSingleton(singletonMethod)
```

## Arguments

`singletonMethod`
: String to be decoded. If necessary, the input string is extended with `F`'s.

## Details

Any `F` means the feature is turned off. Other characters have the following meaning:

1. `singleton2Primary` (1st character):
   - `T`: All singletons are forced to be primary suppressed.
   - `t`: Non-published singletons are primary suppressed.
2. `integerUnique` (2nd character):
   - `T`: Integer values representing the unique contributors are utilized. Error if `singleton` not supplied as integer.
   - `t`: As `T` above, but instead of error, the feature is turned off (as `F`) if `singleton` is not supplied as integer.
3. `sum2` (3rd character):
   - `T`: Virtual primary suppressed cells are made, which are the sum of some suppressed inner cells and which can be divided into two components. At least one component is singleton contributor. The other component may be an inner cell.
   - `H`: As `T` above. And in addition, the other component can be any primary suppressed published cell. This method may be computationally demanding for big data.
4. `elimination` (4th character):
   - `t`: The singleton problem will be handled by methodology implemented as a part of the Gaussian elimination algorithm.
   - `m`: As `t` above. And in addition, a message will be printed to inform about problematic singletons. Actual reveals will be calculated when `singleton2Primary = T` (1st character) and when `singleton2Primary = t` yield the same result as `singleton2Primary = T`. Problematic singletons can appear since the algorithm is not perfect in the sense that the elimination of rows may cause problems. Such problems can be a reason not to switch off `sum2`.

- `w`: As `m` above, but `warning` instead of `message`.
- `T`, `M` and `W`: As `t`, `m` and `w` above. In addition, the gauss elimination routine is allowed to run in parallel with different sortings so that the problem of eliminated singleton rows is reduced.
- `f`: As `F`, which means that the elimination feature is turned off. However, when possible, a message will provide information about actual reveals, similar to `m` above.

5. `combinations` (5th character):
   - `T`: This is a sort of extension of `singleton2Primary` which is relevant when both `integerUnique` and `elimination` are used. For each unique singleton contributor, the method seeks to protect all linear combinations of singleton cells from the unique contributor. Instead of construction new primary cells, protection is achieved as a part of the elimination procedure. Technically this is implemented by extending the above `elimination` method. It cannot be guaranteed that all problems are solved, and this is a reason not to turn off `singleton2Primary`. Best performance is achieved when `elimination` is `T`, `M` or `W`.
   - `t`: As `T`, but without the added singleton protection. This means that protected linear combinations cannot be calculated linearly from non-suppressed cells. However, other contributors may still be able to recalculate these combinations using their own suppressed values.

## Value

A character vector or `NULL`

## Note

Note an update made in SSBtools version 1.7.5, which relates to this sentence in Langsrud (2024): *"The remaining non-zero rows for columns corresponding to primary cells must not originate solely from one contributor."* Due to speed and memory considerations, the algorithm does not first perform elimination and then check the result. Instead, it first verifies whether elimination is permissible before executing it. This approach allows for a more thorough validation process compared to performing elimination with a fixed row order. Specifically, this means that the singleton procedure, denoted as `elimination` (4th character), can take into account different row orders. Such an improvement was introduced in SSBtools version 1.7.5 after the publication of Langsrud (2024). As a result, more potential issues can now be detected during a single elimination sequence (`t` as the 4th character), slightly reducing the need for double elimination (`T` as the 4th character). However, since the row order influences subsequent elimination steps, double elimination remains an important safeguard.

## References

Langsrud, Ø. (2024): "Secondary Cell Suppression by Gaussian Elimination: An Algorithm Suitable for Handling Issues with Zeros and Singletons". Presented at: *Privacy in statistical databases*, Antibes, France. September 25-27, 2024. doi:10.1007/9783031696510_6

## Examples

```
NumSingleton("numTFF")
NumSingleton("numFtT")
NumSingleton("numttH")
```

```
NumSingleton("numTTFTT")
```

---

`output_term_labels` *Extract vector of term labels from a data.frame*

---

## Description

The data.frame is assumed to be have been constructed with `ModelMatrix()` functionality using the `formula` parameter.

## Usage

```
output_term_labels(x)
```

## Arguments

| | |
|---|---|
| `x` | data.frame |

## Value

vector of term labels

## See Also

`formula_term_labels()`, `tables_by_formulas()`

## Examples

```
out <- model_aggregate(SSBtoolsData("magnitude1"),
                       formula = ~eu:sector4 + geo * sector2,
                       sum_vars = "value",
                       avoid_hierarchical = TRUE)
out
term_labels <- output_term_labels(out)
term_labels
cbind(term_labels, out)
```

quantile_weighted *Weighted quantiles*

## Description

The default method (type=2) corresponds to weighted percentiles in SAS.

## Usage

```
quantile_weighted(
  x,
  probs = (0:4)/4,
  weights = rep(1, length(x)),
  type = 2,
  eps = 1e-09
)
```

## Arguments

| | |
|---|---|
| x | Numeric vector |
| probs | Numeric vector of probabilities |
| weights | Numeric vector of weights of the same length as x |
| type | An integer, 2 (default) or 5. Similar to types 2 and 5 in quantile. |
| eps | Precision parameter used when type=2 so that numerical inaccuracy is accepted (see details) |

## Details

When type=2, averaging is used in case of equal of probabilities. Equal probabilities (p[j]==probs[i]) is determined by abs(1-p[j]/probs[i])<eps with p=cumsum(w)/sum(w) where w=weights[order(x)].

With zero length of x, NAs are returned.

When all weights are zero and when when all x's are not equal, NaNs are returned except for the 0% and 100% quantiles.

## Value

Quantiles as a named numeric vector.

## Note

Type 2 similar to type 5 in DescTools::Quantile

## Examples

```
x <- rnorm(27)/5 + 1:27
w <- (1:27)/27

quantile_weighted(x, (0:5)/5, weights = w)
quantile_weighted(x, (0:5)/5, weights = w, type = 5)

quantile_weighted(x) - quantile(x, type = 2)
quantile_weighted(x, type = 5) - quantile(x, type = 5)
```

---

`RbindAll` *Combining several data frames when the columns don't match*

---

## Description

Combining several data frames when the columns don't match

## Usage

```
RbindAll(...)
```

## Arguments

| | |
|---|---|
| `...` | Several data frames as several input parameters or a list of data frames |

## Value

A single data frame

## Note

The function is an extended version of rbind.all.columns at https://amywhiteheadresearch.wordpress.com/2013/05/13/combining-dataframes-when-the-columns-dont-match/

## Author(s)

Øyvind Langsrud

## See Also

`CbindIdMatch` (same example data)

## Examples

```
zA <- data.frame(idA = 1:10, idB = rep(10 * (1:5), 2), idC = rep(c(100, 200), 5),
                 idC2 = c(100, rep(200, 9)), idC3 = rep(100, 10),
                 idD = 99, x = round(rnorm(10), 3), xA = round(runif(10), 2))
zB <- data.frame(idB = 10 * (1:5), x = round(rnorm(5), 3), xB = round(runif(5), 2))
zC <- data.frame(idC = c(100, 200), x = round(rnorm(2), 3), xC = round(runif(2), 2))
zD <- data.frame(idD = 99, x = round(rnorm(1), 3), xD = round(runif(1), 2))
RbindAll(zA, zB, zC, zD)
RbindAll(list(zA, zB, zC, zD))
```

---

# Reduce0exact *Reducing a non-negative regression problem*

---

## Description

The linear equation problem, `z = t(x) %*% y` with y non-negative and x as a design (dummy) matrix, is reduced to a smaller problem by identifying elements of y that can be found exactly from `x` and `z`.

## Usage

```
Reduce0exact(
  x,
  z = NULL,
  reduceByColSums = FALSE,
  reduceByLeverage = FALSE,
  leverageLimit = 0.999999,
  digitsRoundWhole = 9,
  y = NULL,
  yStart = NULL,
  printInc = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | A matrix |
| `z` | A single column matrix |
| `reduceByColSums` | See Details |
| `reduceByLeverage` | See Details |
| `leverageLimit` | Limit to determine perfect fit |
| `digitsRoundWhole` | `RoundWhole` parameter for fitted values (when `leverageLimit` and `y` not in input) |

| | |
|---|---|
| y | A single column matrix. With y in input, z in input can be omitted and estimating y (when leverageLimit) is avoided. |
| yStart | A starting estimate when this function is combined with iterative proportional fitting. Zeros in yStart will be used to reduce the problem. |
| printInc | Printing iteration information to console when TRUE |

## Details

Exact elements can be identified in three ways in an iterative manner:

1. By zeros in z. This is always done.
2. By columns in x with a singe nonzero value. Done when reduceByColSums or reduceByLeverage is TRUE.
3. By exact linear regression fit (when leverage is one). Done when reduceByLeverage is TRUE. The leverages are computed by hat(as.matrix(x), intercept = FALSE), which can be very time and memory consuming. Furthermore, without y in input, known values will be computed by ginv.

## Value

A list of five elements:

- x: A reduced version of input x
- z: Corresponding reduced z
- yKnown: Logical, specifying known values of y
- y: A version of y with known values correct and others zero
- zSkipped: Logical, specifying omitted columns of x

## Author(s)

Øyvind Langsrud

## Examples

```
# Make a special data set
d <- SSBtoolsData("sprt_emp")
d$ths_per <- round(d$ths_per)
d <- rbind(d, d)
d$year <- as.character(rep(2014:2019, each = 6))
to0 <- rep(TRUE, 36)
to0[c(6, 14, 17, 18, 25, 27, 30, 34, 36)] <- FALSE
d$ths_per[to0] <- 0

# Values as a single column matrix
y <- Matrix::Matrix(d$ths_per, ncol = 1)

# A model matrix using a special year hierarchy
x <- Hierarchies2ModelMatrix(d, hierarchies = list(geo = "", age = "", year =
    c("y1418 = 2014+2015+2016+2017+2018", "y1519 = 2015+2016+2017+2018+2019",
```

```
      "y151719 = 2015+2017+2019", "yTotal = 2014+2015+2016+2017+2018+2019")),
      inputInOutput = FALSE)

# Aggregates
z <- Matrix::t(x) %*% y
sum(z == 0)  # 5 zeros

# From zeros in z
a <- Reduce0exact(x, z)
sum(a$yKnown)   # 17 zeros in y is known
dim(a$x)        # Reduced x, without known y and z with zeros
dim(a$z)        # Corresponding reduced z
sum(a$zSkipped) # 5 elements skipped
Matrix::t(a$y)          # Just zeros (known are 0 and unknown set to 0)

# It seems that three additional y-values can be found directly from z
sum(Matrix::colSums(a$x) == 1)

# But it is the same element of y (row 18)
a$x[18, Matrix::colSums(a$x) == 1]

# Make use of ones in colSums
a2 <- Reduce0exact(x, z, reduceByColSums = TRUE)
sum(a2$yKnown)          # 18 values in y is known
dim(a2$x)               # Reduced x
dim(a2$z)               # Corresponding reduced z
a2$y[which(a2$yKnown)]  # The known values of y (unknown set to 0)

# Six ones in leverage values
# Thus six extra elements in y can be found by linear estimation
hat(as.matrix(a2$x), intercept = FALSE)

# Make use of ones in leverages (hat-values)
a3 <- Reduce0exact(x, z, reduceByLeverage = TRUE)
sum(a3$yKnown)          # 26 values in y is known (more than 6 extra)
dim(a3$x)               # Reduced x
dim(a3$z)               # Corresponding reduced z
a3$y[which(a3$yKnown)]  # The known values of y (unknown set to 0)

# More than 6 extra is caused by iteration
# Extra checking of zeros in z after reduction by leverages
# Similar checking performed also after reduction by colSums
```

---

RoundWhole *Round values that are close two whole numbers*

---

## Description

Round values that are close two whole numbers

## Usage

```
RoundWhole(x, digits = 9, onlyZeros = FALSE)
```

## Arguments

| | |
|---|---|
| x | vector or matrix |
| digits | parameter to round |
| onlyZeros | Only round values close to zero |

## Details

When digits is NA, Inf or NULL, input is returned unmodified. When there is more than one element in digits or onlyZeros, rounding is performed column-wise.

## Value

Modified x

## Author(s)

Øyvind Langsrud

## Examples

```
x <- c(0.0002, 1.00003, 3.00014)
RoundWhole(x)     # No values rounded
RoundWhole(x, 4)  # One value rounded
RoundWhole(x, 3)  # All values rounded
RoundWhole(x, NA) # No values rounded (always)
RoundWhole(x, 3, TRUE)  # One value rounded
RoundWhole(cbind(x, x, x), digits = c(3, 4, NA))
RoundWhole(cbind(x, x), digits = 3, onlyZeros = c(FALSE, TRUE))
```

---

# RowGroups *Create numbering according to unique rows*

---

## Description

Create numbering according to unique rows

## Usage

```
RowGroups(
  x,
  returnGroups = FALSE,
  returnGroupsId = FALSE,
  NAomit = FALSE,
  pkg = "base"
)
```

### Arguments

| | |
|---|---|
| x | Data frame or matrix |
| returnGroups | When TRUE unique rows are returned |
| returnGroupsId | When TRUE Index of unique rows are returned |
| NAomit | When TRUE, rows containing NAs are omitted, and the corresponding index numbers are set to NA. |
| pkg | A character string indicating which package to use. Must be either "base" for base R or "data.table" for data.table. Default is "base". |

### Value

A vector with the numbering or, according to the arguments, a list with more output.

### Author(s)

Øyvind Langsrud

### Examples

```
a <- data.frame(x = c("a", "b"), y = c("A", "B", "A"), z = rep(1:4, 3))
RowGroups(a)
RowGroups(a, TRUE)
RowGroups(a[, 1:2], TRUE, TRUE)
RowGroups(a[, 1, drop = FALSE], TRUE)
```

---

## SortRows *Sorting rows of a matrix or data frame*

---

### Description

Sorting rows of a matrix or data frame

### Usage

```
SortRows(m, cols = 1:dim(m)[2], index.return = FALSE)
```

### Arguments

| | |
|---|---|
| m | matrix or data frame |
| cols | Indexes of columns, in the desired order, used for sorting. |
| index.return | logical indicating if the ordering index vector should be returned instead of sorted input. |

### Value

sorted m or a row index vector

### Author(s)

Øyvind Langsrud

### Examples

```
d <- SSBtoolsData("d2w")
SortRows(d[4:7])
SortRows(d, cols = 4:7)
SortRows(d, cols = c(2, 4))

SortRows(matrix(sample(1:3,15,TRUE),5,3))
```

---

## SSBtoolsData *Function that returns a dataset*

---

### Description

Function that returns a dataset

### Usage

```
SSBtoolsData(dataset)
```

### Arguments

| | |
|---|---|
| dataset | Name of data set within the SSBtools package |

### Details

**FIFA2018ABCD:** A hierarchy table based on countries within groups A-D in the football championship, 2018 FIFA World Cup.

**sprt_emp:** Employment in sport in thousand persons. Data from Eurostat database.

**sprt_emp_geoHier:** Country hierarchy for the employment in sport data.

**sprt_emp_ageHier:** Age hierarchy for the employment in sport data.

**sprt_emp_withEU:** The data set sprt_emp extended with a EU variable.

**sp_emp_withEU:** As `sprt_emp_withEU`, but coded differently.

**example1** Example data similar to `sp_emp_withEU`.

**magnitude1:** Example data for magnitude tabulation. Same countries as above.

**my_km2:** Fictitious grid data.

**mun_accidents:** Fictitious traffic accident by municipality data.

**sosialFiktiv, z1, z1w, z2, z2w, z3, z3w, z3wb:** See `sosialFiktiv`.

**d4, d1, d1w, d2, d2w, d3, d3w, d3wb:** English translation of the datasets above.

**d2s, d2ws:** `d2` and `d2w` modified to smaller/easier data.

**power10to1, power10to2, ...:** `power10to`$i$ is hierarchical data with $10^i$ rows and $2 * i$ columns. Tip: Try `FindDimLists(SSBtoolsData("power10to3"))`

**code_pairs:** Example dataset with two code columns illustrating paired categorical codes.

**barcelona2025:** Example data in poster at expert meeting in Barcelona 2025.

**paris2025_freq:** Example frequency data for INSEE Statistical Methodology Days (JMS 2025)

**paris2025_micro:** Example microdata for INSEE Statistical Methodology Days (JMS 2025)

## Value

data frame

## Author(s)

Øyvind Langsrud and Daniel Lupp

## Examples

```
SSBtoolsData("FIFA2018ABCD")
SSBtoolsData("sprt_emp")
SSBtoolsData("sprt_emp_geoHier")
SSBtoolsData("sprt_emp_ageHier")
SSBtoolsData("sprt_emp_withEU")
SSBtoolsData("d1w")
SSBtoolsData("barcelona2025")
SSBtoolsData("paris2025_freq")
```

---

# Stack *Stack columns from a data frame and include variables.*

---

## Description

Stack columns from a data frame and include variables.

## Usage

```
Stack(
  data,
  stackVar = 1:NCOL(data),
  blockVar = integer(0),
  rowData = data.frame(stackVar)[, integer(0), drop = FALSE],
  valueName = "values",
  indName = "ind"
)
```

### Arguments

| | |
|---|---|
| data | A data frame |
| stackVar | Indices of variables to be stacked |
| blockVar | Indices of variables to be replicated |
| rowData | A separate data frame where NROW(rowData)=length(stackVar) such that each row may contain multiple information of each stackVar variable. The output data frame will contain an extended variant of rowData. |
| valueName | Name of the stacked/concatenated output variable |
| indName | Name of the output variable with information of which vector in x the observation originated. When indName is NULL this variable is not included in output. |

### Value

A data frame where the variable ordering corresponds to: blockVar, rowData, valueName, indName

### Author(s)

Øyvind Langsrud

### See Also

Unstack

### Examples

```
z <- data.frame(n=c(10,20,30), ssb=c('S','S','B'),
Ayes=1:3,Ano=4:6,Byes=7:9,Bno=10:12)
zRow <- data.frame(letter=c('A','A','B','B'),answer=c('yes','no','yes','no') )

x <- Stack(z,3:6,1:2,zRow)

Unstack(x,6,3:4,numeric(0),1:2)
Unstack(x,6,5,numeric(0),1:2)
Unstack(x,6,3:4,5,1:2)
```

---

## tables_by_formulas *Tabular Statistics Based on Formulas*

---

### Description

This function acts as an overlay for functions that produce tabular statistics through an interface utilizing the ModelMatrix() function and its formula parameter. Each table (individual statistic) is defined by a formula. The output is a single data.frame that contains the results for all tables.

## Usage

```
tables_by_formulas(
  data,
  table_fun,
  ...,
  table_formulas,
  substitute_vars = NULL,
  auto_collapse = TRUE,
  collapse_vars = NULL,
  total = "Total",
  hierarchical_extend0 = TRUE,
  term_labels = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The input data to be processed by table_fun. |
| table_fun | The table-producing function to be used. |
| ... | Additional arguments passed to table_fun. |
| table_formulas | A named list of formulas, where each entry defines a specific table. |
| substitute_vars | Allows formulas in table_formulas to be written in a simplified way. If substitute_vars is specified, the final formulas are generated using substitute_formula_vars() with substitute_vars as input. |
| auto_collapse | Logical. If TRUE, variables are collapsed using total_collapse() with the variables parameter according to substitute_vars. |
| collapse_vars | When specified, total_collapse() is called with collapse_vars as the variables parameter, after any call triggered by the auto_collapse parameter. |
| total | string(s) used to name totals. Passed to both table_fun and total_collapse(). When total is named, the names are handled in a way that accounts for both substitute_vars with auto_collapse and collapse_vars. You may supply names corresponding to either input or output variables. Inconsistent or incompatible naming will result in an error. |
| hierarchical_extend0 | Controls automatic hierarchy generation for Extend0(). See "Details" for more information. |
| term_labels | Logical. If TRUE, a term_labels column (as constructed by output_term_labels()) is included as the first column of the output. |

## Details

To ensure full control over the generated output variables, table_fun is called with avoid_hierarchical or avoidHierarchical set to TRUE. Desired variables in the output are achieved using substitute_vars, auto_collapse, and collapse_vars.

If table_fun automatically uses Extend0(), the parameter hierarchical_extend0 specifies the hierarchical parameter in Extend0() via Extend0fromModelMatrixInput(). When hierarchical_extend0

is TRUE, hierarchies are generated automatically. By default, it is set to TRUE, preventing excessive data extension and aligning with the default behavior of Formula2ModelMatrix(), where avoidHierarchical = FALSE.

An attribute table_formulas is added to formula before table_fun() is called. This attribute contains the version of table_formulas after applying substitute_vars. This allows for special use in the function table_fun().

Note: The use of total_collapse internally allows handling of variable names not present in the data. This ensures flexibility when modifying the table_formulas parameter.

## Value

A single data.frame containing results for all tables defined in table_formulas.

## See Also

filter_by_variable

## Examples

```
tables_by_formulas(SSBtoolsData("magnitude1"),
                   table_fun = model_aggregate,
                   table_formulas = list(table_1 = ~region * sector2,
                                         table_2 = ~region1:sector4 - 1,
                                         table_3 = ~region + sector4 - 1),
                   substitute_vars = list(region = c("geo", "eu"), region1 = "eu"),
                   collapse_vars = list(sector = c("sector2", "sector4")),
                   sum_vars = "value",
                   total = c(region = "E", sector = "T"),
                   term_labels = TRUE)
```

---

# table_all_integers *Table all integers from 1 to n*

---

## Description

Table all integers from 1 to n

## Usage

```
table_all_integers(x, n)
```

## Arguments

| | |
|---|---|
| x | A vector of integers. |
| n | The maximum integer value. |

### Value

A 1D array of class "table" representing the frequency of each integer from 1 to n.

### Examples

```
table_all_integers(c(2, 3, 5, 3, 5, 3), 7)
```

---

## total_collapse *Collapse variables to single representation*

---

### Description

Simplify a data frame by collapsing specified variables, according to the location of total codes, into a single vector or by consolidating groups of variables into new columns.

### Usage

```
total_collapse(data, variables, total = "Total", include_names = NULL)
```

### Arguments

| | |
|---|---|
| data | A data frame containing the variables to be collapsed. |
| variables | A vector of variable names or a named list of variable names.<br>• If variables is a vector, the specified variables in data are collapsed into a single character vector.<br>• If variables is a named list, each element in the list defines a group of variables to consolidate into a new column. Each list name will be used as the new column name in the resulting data frame. |
| total | A total code or vector of total codes to use in the result.<br>• If variables is a vector, total specifies the code to represent collapsed values.<br>• If variables is a named list, total may contain one code per group. |
| include_names | A character string or NULL (default).<br>• If variables is a vector, whether the resulting output vector is named depends on whether include_names is NULL or not. The actual value of include_names is ignored in this case.<br>• If variables is a named list, include_names specifies a suffix to append to each group name, creating one additional column per group. If NULL, no additional columns with variable names are included in the result. |

### Value

A character vector (if variables is a vector) or a modified data frame (if variables is a named list).

## Examples

```
# Creates data that can act as input
magnitude1 <- SSBtoolsData("magnitude1")
a <- model_aggregate(magnitude1,
                     formula = ~geo + eu + sector2 + sector4,
                     sum_vars = "value",
                     avoid_hierarchical = TRUE)
a

b <- total_collapse(a, list(GEO = c("geo", "eu"), SECTOR = c("sector2", "sector4")))
b

total_collapse(a, c("geo", "eu"))
total_collapse(a, c("sector2", "sector4"))


# Similar examples with both `total` and `include_names` parameters
aa <- a
aa[1:2][aa[1:2] == "Total"] <- "Europe"
aa[3:4][aa[3:4] == "Total"] <- ""
aa

bb <- total_collapse(data = aa,
                     variables = list(GEO = c("geo", "eu"),
                                      SECTOR = c("sector2", "sector4")),
                     total = c("Europe", ""),
                     include_names = "_Vars")
bb

total_collapse(aa, c("geo", "eu"), total = "Europe", include_names = "_Vars")
total_collapse(aa, c("sector2", "sector4"), total = "", include_names = "_Vars")


# All four variables can be collapsed
total_collapse(a,
               list(ALL = c("geo", "eu", "sector2", "sector4")),
               include_names = "_Vars")
```

---

`UniqueSeq` *Sequence within unique values*

---

## Description

Sequence within unique values

## Usage

```
UniqueSeq(x, sortdata = matrix(1L, length(x), 0))
```

### Arguments

| | |
|---|---|
| x | vector |
| sortdata | matrix or vector to determine sequence order |

### Value

integer vector

### Author(s)

Øyvind Langsrud

### Examples

```
# 1:4 within A and 1:2 within B
UniqueSeq(c("A", "A", "B", "B", "A", "A"))

# Ordered differently
UniqueSeq(c("A", "A", "B", "B", "A", "A"), c(4, 5, 20, 10, 3, 0))
```

---

## Unstack *Unstack a column from a data frame and include additional variables.*

---

### Description

Unstack a column from a data frame and include additional variables.

### Usage

```
Unstack(
  data,
  mainVar = 1,
  stackVar = (1:NCOL(data))[-mainVar],
  extraVar = integer(0),
  blockVar = integer(0),
  sep = "_",
  returnRowData = TRUE,
  sorted = FALSE
)
```

### Arguments

| | |
|---|---|
| data | A data frame |
| mainVar | Index of the variable to be unstacked |
| stackVar | Index of variables defining the unstack grouping |
| extraVar | Indices of within-replicated variables to be added to the rowData output |

| | |
|---|---|
| blockVar | Indices of between-replicated variables to be added to the data output |
| sep | A character string to separate when creating variable names |
| returnRowData | When FALSE output is no list, but only data |
| sorted | When TRUE the created variables is in sorted order. Otherwise input order is used. |

## Value

When returnRowData=TRUE output is list of two elements.

| | |
|---|---|
| data | Unstacked data |
| rowData | A separate data frame with one row for each unstack grouping composed of the stackVar variables |

## Author(s)

Øyvind Langsrud

## See Also

Stack (examples)

---

# vars_to_hierarchies *Transform hierarchies coded as Variables to "to-from" format*

---

## Description

A kind of reverse operation of hierarchies_as_vars

## Usage

```
vars_to_hierarchies(var_hierarchies)
```

## Arguments

| | |
|---|---|
| var_hierarchies | As output from hierarchies_as_vars |

## Value

List of hierarchies

## Examples

```
a <- hierarchies_as_vars(list(f =
       c("AB = A + B", "CD = C + D", "AC = A + C", "ABCD = AB + CD")))
a

vars_to_hierarchies(a)
```

`WildcardGlobbing` *Row selection by wildcard/globbing*

## Description

The selected rows match combined requirements for all variables.

## Usage

```
WildcardGlobbing(x, wg, sign = TRUE, invert = "!")
```

## Arguments

| | |
|---|---|
| `x` | data.frame with character data |
| `wg` | data.frame with wildcard/globbing |
| `sign` | When FALSE, the result is inverted. |
| `invert` | Character to invert each single selection. |

## Details

This function is used by `HierarchicalWildcardGlobbing` and `WildcardGlobbingVector` and make use of `grepl` and `glob2rx`.

## Value

Logical vector defining subset of rows.

## Author(s)

Øyvind Langsrud

## Examples

```
# Create data input
data(precip)
data(mtcars)
x <- data.frame(car = rownames(mtcars)[rep(1:NROW(mtcars), each = 35)], city = names(precip),
                stringsAsFactors = FALSE)

# Create globbing/wildcards input
wg <- data.frame(rbind(c("Merc*", "C*"), c("F*", "??????"), c("!??????????*", "!???????*")),
                 stringsAsFactors = FALSE)
names(wg) <- names(x)

# Select the following combinations:
# - Cars starting with Merc and cities starting with C
# - Cars starting with F and six-letter cities
# - Cars with less than nine letters and cities with less than seven letters
x[WildcardGlobbing(x, wg), ]
```

WildcardGlobbingVector *Selection of elements by wildcard/globbing*

## Description

Selection of elements by wildcard/globbing

## Usage

```
WildcardGlobbingVector(x, wg, negSign = "-", invert = "!")
```

## Arguments

| | |
|---|---|
| x | Character vector |
| wg | Character vector with wildcard/globbing |
| negSign | Character representing selection to be removed |
| invert | Character to invert each single selection. |

## Value

vector with selected elements of x

## Author(s)

Øyvind Langsrud

## Examples

```
data(precip)
x <- names(precip)

# Select the cities starting with B, C and Sa.
WildcardGlobbingVector(x, c("B*", "C*", "Sa*"))

# Remove from the selection cities with o and t in position 2 and 4, respectively.
WildcardGlobbingVector(x, c("B*", "C*", "Sa*", "-?o*", "-???t*"))

# Add to the selection cities not having six or more letters.
WildcardGlobbingVector(x, c("B*", "C*", "Sa*", "-?o*", "-???t*", "!??????*"))
```

zero_col *Check for empty matrix columns (or rows)*

## Description

More generally, checks that both row/col sums and sums of absolute values equal a target. For `value = 0`, this means all entries are zero. `single_col()` is a wrapper with `value = 1`, often used to check for dummy columns/rows with exactly one element that is 1.

## Usage

```
zero_col(x, rows = FALSE, value = 0)

single_col(..., value = 1)
```

## Arguments

| | |
|---|---|
| `x` | Numeric matrix. Sparse matrices from the Matrix package are also supported. |
| `rows` | Logical; if `TRUE` check rows, else columns. |
| `value` | Numeric target (default `0`). |
| `...` | Passed to `zero_col()`. |

## Details

Memory usage is reduced by applying `abs()` checks only to rows/columns whose total sum is already the target.

## Value

Logical vector.

## Examples

```
m <- matrix(c(
  0,  0, 0, 0,
  1, -1, 0, 0,
  0,  0, 1, 0
), nrow = 3, byrow = TRUE)

zero_col(m)
zero_col(m, rows = TRUE)
single_col(m)
single_col(m, rows = TRUE)
```

# Index