Package 'SSBtools'

September 7, 2021

Type Package

Title Statistics Norway's Miscellaneous Tools

Version 1.2.2

Date 2021-09-07

Depends Matrix

Imports stringr, methods, MASS

Description Functions used by other packages from Statistics Norway are gathered. General data manipulation functions, and functions for hierarchical computations are included (Langsrud, 2020) <doi:10.13140/RG.2.2.27313.61283>. The hierarchy specification functions are useful within statistical disclosure control.

License Apache License 2.0 | file LICENSE

URL https://github.com/statisticsnorway/SSBtools

BugReports https://github.com/statisticsnorway/SSBtools/issues

RoxygenNote 7.1.1

Encoding UTF-8

Suggests testthat

NeedsCompilation no

Author Øyvind Langsrud [aut, cre],
Daniel Lupp [aut],
Bjørn-Helge Mevik [cph]

Maintainer Øyvind Langsrud <oyl@ssb.no>

Repository CRAN

Date/Publication 2021-09-07 16:10:02 UTC

R topics documented:

AddLeadingZeros 3
AutoHierarchies 4
AutoSplit 5

CbindIdMatch 6
DimList2Hierarchy 8
DimList2Hrc 9
DummyDuplicated 9
DummyHierarchy 10
Extend0 12
FactorLevCorr 14
FindCommonCells 15
FindDimLists 16
FindDisclosiveCells 17
FindTableGroup 18
FormulaSums 19
GaussIndependent 21
GaussSuppression 22
HierarchicalGroups 24
HierarchicalWildcardGlobbing 25
Hierarchies2ModelMatrix 27
HierarchiesAndFormula2ModelMatrix 30
Hierarchy2Formula 32
HierarchyCompute 33
HierarchyCompute2 36
HierarchyFix 38
LSfitNonNeg 39
MakeHierFormula 40
Match 41
matlabColon 42
Matrix2list 43
Mipf 44
ModelMatrix 48
Number 50
RbindAll 51
Reduce0exact 52
RoundWhole 54
RowGroups 55
SSBtoolsData 56
Stack 57
UniqueSeq 58
Unstack 59
WildcardGlobbing 60
WildcardGlobbingVector 61

Index **62**

`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)
```

`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
)

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

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`.
`hierarchyVarNames`	Variable names in the hierarchy tables as in `HierarchyFix`
`combineHierarchies`	Whether to combine several hierarchies for same variable into a single hierarchy
`unionComplement`	Logical vector as in `Hierarchies2ModelMatrix`. The parameter is only in use when hierarchies are combined.

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`. `FindHierarchies` wraps `FindDimLists` and `AutoHierarchies` into a single function. 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"`.

Value

List of hierarchies

Author(s)

Øyvind Langsrud

See Also

DimList2Hierarchy, Hierarchy2Formula.

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)

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

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
```

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

Examples

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

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

Examples

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

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(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(t(z), idx = TRUE, rows = TRUE)
DummyDuplicated(t(z), idx = TRUE, rows = TRUE, rnd = TRUE)

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

`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
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.

Value

Extended data frame

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")
```

`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 nij 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 nij = ni*nj
`1`	when nij = max(ni,nj)
`Other values`	Computed as (ni*nj-nij)/(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)
```

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.

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

```
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)
```

`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)
```

`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)
```

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,
  ...
)

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	String used to name totals
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.
...	Further arguments to be passed to FormulaSums

Details

The model matrix is constructed by calling fac2sparse() repeatedly. 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")
```

GaussIndependent *Linearly independent rows and columns by Gaussian elimination*

Description

The function is written primarily for large sparse matrices

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

Examples

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

GaussIndependent(x)
GaussRank(x)
GaussRank(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)
```

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,
  ...
)
```

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
hidden	Indices to be removed from the above candidates input (see details)

`singleton` Logical vector specifying inner cells for singleton handling. Normally, this means cells with 1s when 0s are non-suppressed and cells with 0s when 0s are suppressed.

`singletonMethod`
Method for handling the problem of singletons and zeros: `"anySum"` (default), `"anySumNOTprimary"`, `"subSum"`, `"subSpace"` or `"none"` (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.

`...` Extra unused parameters

Details

It is possible to specify too many (all) indices as `candidates`. Indices specified as `primary` or `hidded` will be removed. Hidden indices (not candidates or primary) refer to cells that will not be published, but do not need protection. The singleton method `"subSum"` makes new imaginary 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 imaginary 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. All the above methods assume that any published singletons are primary suppressed. When this is not the case, `"anySumNOTprimary"` must be used.

Value

Secondary suppression indices

Examples

```
# Input data
df <- data.frame(values = c(1, 1, 1, 5, 5, 9, 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
```

`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

```
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
)
```

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.

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.
crossTable	Cross table in output when TRUE
total	Vector of total codes (possibly recycled) used when running Hrc2DimList
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.
removeEmpty	When TRUE and when select=NULL, 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.

Value

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

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
t(m1) %*% ths_per  # crossprod(m1, ths_per) is equivalent and faster
t(m2) %*% ths_per
t(m3) %*% ths_per
t(m4) %*% ths_per
t(m5) %*% ths_per
t(m6) %*% ths_per


# Example using the select parameter
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]
t(m2b) %*% ths_per
```

`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) used when running Hrc2DimList
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))
t(m) %*% z$ths_per

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

`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)
```

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 +.

Value

See Arguments

Author(s)

Øyvind Langsrud

Examples

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

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")
```

HierarchyFix *Change the hierarchy table to follow the standard*

Description

Make sure that variable names and sign coding follow an internal standard. Level may be computed automatically

Usage

```
HierarchyFix(
  hierarchy,
 hierarchyVarNames = c(mapsFrom = "mapsFrom", mapsTo = "mapsTo", sign = "sign", level
    = "level"),
  autoLevel = TRUE
)
```

Arguments

hierarchy	data frame with hierarchy table
hierarchyVarNames	variable names
autoLevel	When TRUE, level is computed by automatic method

Value

data frame with hierarchy table

Author(s)

Øyvind Langsrud

Examples

```
# Make input data by changing variable names and sign coding.
h <- SSBtoolsData("FIFA2018ABCD")[, 1:3]
names(h)[1:2] <- c("from", "to")
minus <- h$sign < 0
h$sign <- "+"
h$sign[minus] <- "-"

# Run HierarchyFix - Two levels created
HierarchyFix(h, c(mapsFrom = "from", mapsTo = "to", sign = "sign"))

# Extend the hierarchy table
h2 <- rbind(data.frame(from = c("Oceania", "Asia", "Africa", "America", "Europe"),
                       to = "World", sign = "+"),
            data.frame(from = c("World", "Europe"),
                       to = "nonEurope", sign = c("+", "-")), h)

# Run HierarchyFix - Three levels created
HierarchyFix(h2, c(mapsFrom = "from", mapsTo = "to", sign = "sign"))
```

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 <- 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 <- 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)
```

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)
```

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 <- 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 <- 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 <- 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 <- 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 <- 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 <- 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 <- 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 <- 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 = FALSE,
  modelMatrix = 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 used to name totals
removeEmpty	When TRUE, empty columns (only zeros) are not included in output (relevant when hierarchies)
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.
...	Further arguments to Hierarchies2ModelMatrix, Formula2ModelMatrix or HierarchiesAndFormula2ModelMatrix

Value

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

Author(s)

Øyvind Langsrud

Examples

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

# 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, 3)])

# 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)
```

`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)
```

`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(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 <- 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
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(colSums(a$x) == 1)

# But it is the same element of y (row 18)
a$x[18, colSums(a$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)
```

Arguments

x	Data frame or matrix
returnGroups	When TRUE unique rows are returned
returnGroupsId	When TRUE Index of unique rows are returned

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)
```

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.

my_km2: Fictitious grid data.

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

Value

data frame

Author(s)

Øyvind Langsrud

Examples

```
SSBtoolsData("FIFA2018ABCD")
SSBtoolsData("sprt_emp")
SSBtoolsData("sprt_emp_geoHier")
SSBtoolsData("sprt_emp_ageHier")
SSBtoolsData("sprt_emp_withEU")
SSBtoolsData("z1w")
```

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)
```

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)

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*", "!??????*"))
```

Index

AddLeadingZeros, 3
AutoHierarchies, 4, *28*, *30*, *35*, *38*, *49*
AutoSplit, 5

CbindIdMatch, 6, *51*

DimList2Hierarchy, *5*, 8
DimList2Hrc, 9
DummyDuplicated, 9
DummyHierarchies *(*DummyHierarchy*)*, 10
DummyHierarchy, 10, *34, 35*

Extend0, 12

FactorLevCorr, 14
FindCommonCells, 15
FindDimLists, *4*, 16
FindDisclosiveCells, 17
FindHierarchies *(*AutoHierarchies*)*, 4
FindTableGroup, 18
Formula2Hierarchy *(*Hierarchy2Formula*)*, 32
Formula2ModelMatrix, *31*, *48, 49*
Formula2ModelMatrix *(*FormulaSums*)*, 19
FormulaSums, 19, *45*

GaussIndependent, 21, *40*
GaussRank *(*GaussIndependent*)*, 21
GaussSuppression, 22
ginv, *53*
glob2rx, *60*
grepl, *60*

hclust, *26*
HierarchicalGroups, 24
HierarchicalWildcardGlobbing, 25, *60*
Hierarchies2ModelMatrix, *4*, 27, *31*, *35*, *38*, *48, 49*
HierarchiesAndFormula2ModelMatrix, *29*, 30, *48, 49*
Hierarchy2Formula, *5*, 32
HierarchyCompute, *4*, *28*, 33, *37*
HierarchyCompute2, *34*, 36
HierarchyComputeDummy, *28*
HierarchyFix, *4*, *28*, *31, 32*, *34*, 38
Hrc2DimList, *4*, *28*, *31*
Hrc2DimList *(*DimList2Hrc*)*, 9

loglin, *45*
LSfitNonNeg, 39

MakeHierFormula, 40
Match, 41
match, *41*
matlabColon, 42
Matrix2list, 43
Matrix2listInt *(*Matrix2list*)*, 43
Mipf, 44
model.matrix, *49*
ModelMatrix, *20*, *29*, *31*, *45*, 48

Number, 50

qr, *39, 40*

RbindAll, *7*, 51
Reduce0exact, *45*, 52
round, *55*
RoundWhole, *52*, 54
RowGroups, 55

seq, *43*
SeqInc *(*matlabColon*)*, 42
solve, *40*
sosialFiktiv, *56*
sparse.model.matrix, *49*
SSBtoolsData, 56
Stack, 57, *60*

UniqueSeq, 58
Unstack, *58*, 59

WildcardGlobbing, *26*, 60
WildcardGlobbingVector, *60*, 61