Package ‘wrMisc’

March 20, 2023

Version 1.12.0
Title Analyze Experimental High-Throughput (Omics) Data
Author Wolfgang Raffelsberger [aut, cre]
Maintainer Wolfgang Raffelsberger <w.raffelsberger@gmail.com>
Description The efficient treatment and convenient analysis of experimental high-throughput (omics) data gets facilitated through this collection of diverse functions. Several functions address advanced object-conversions, like manipulating lists of lists or lists of arrays, reorganizing lists to arrays or into separate vectors, merging of multiple entries, etc. Another set of functions provides speed-optimized calculation of standard deviation (sd), coefficient of variance (CV) or standard error of the mean (SEM) for data in matrixes or means per line with respect to additional grouping (eg n groups of replicates). Other functions facilitate dealing with non-redundant information, by indexing unique, adding counters to redundant or eliminating lines with respect redundancy in a given reference-column, etc. Help is provided to identify very closely matching numeric values to generate (partial) distance matrixes for very big data in a memory efficient manner or to reduce the complexity of large data-sets by combining very close values. Many times large experimental datasets need some additional filtering, adequate functions are provided. Batch reading (or writing) of sets of files and combining data to arrays is supported, too. Convenient data normalization is supported in various different modes, parameter estimation via permutations or boot-strap as well as flexible testing of multiple pair-wise combinations using the framework of ‘limma’ is provided, too.

VignetteBuilder knitr
Depends R (>= 3.1.0)
Imports grDevices, graphics, MASS, stats, utils
Suggests BBmisc, boot, coin, data.table, data.tree, fdrtool, flexclust, knitr, limma, markdown, mixdist, NbClust, preprocessCore, qvalue, Rcpp, RColorBrewer, readxl, rmarkdown, som, stringi, VGAM, vsn, wrGraph
License GPL-3
Encoding UTF-8
RoxygenNote 7.2.2
NeedsCompilation no
Repository CRAN
Date/Publication 2023-03-20 18:20:03 UTC

R topics documented:

.addLetterWoLast .......................................................... 7
.allRatioMatr1to2 .......................................................... 8
.allRatios ................................................................. 9
.arrLstMean ............................................................... 9
.arrLstSEM ............................................................... 10
.asDF2 ................................................................. 11
.breakInSer ............................................................. 12
.bringToCtr ............................................................. 13
.checkArgNa ............................................................. 13
.checkConsistentArrList .............................................. 14
.checkConv2Vect .......................................................... 15
.checkFactor ............................................................ 16
.checkFileNameExtensions ............................................... 17
.checkLegendLoc ........................................................ 17
.checkLmConfInt ........................................................ 18
.checkRegrArguments .................................................. 19
.chooseGrpCol ........................................................... 19
.combineListAnnot ...................................................... 20
.compareByDiff ........................................................... 21
.compareByLogRatio ..................................................... 22
.compareByPPM .......................................................... 22
.complCols ............................................................. 23
.composeCallName ....................................................... 24
.convertMatrToNum ..................................................... 25
.convertNa ............................................................. 25
.corDup1ItemsByIncr .................................................................. 26
.cutAtSearch .............................................................. 27
.cutStr ................................................................. 28
.datSlope ............................................................... 28
.extrNAneighb ........................................................... 29
.extrNumHeadingCap ..................................................... 30
.extrNumHeadingSepChar ............................................... 31
.filterNetw ............................................................ 31
.filterSw ............................................................. 32
.filtSize ............................................................... 33
.findBorderOverlaps ................................................... 34
.firstMin ............................................................. 35
.fuse2ArrBy2ndDim ............................................................... 35
R topics documented:

- `.getAmean` ......................................................... 36
- `.getAmean2` .......................................................... 37
- `.getMvalue2` .......................................................... 37
- `.growTree` ............................................................. 38
- `.insp1dimByClustering` .......................................... 39
- `.inspectHeader` ................................................... 40
- `.keepCenter1d` ..................................................... 41
- `.keepFiniteCol` ..................................................... 42
- `.mayBeNum` ............................................................ 43
- `.medianSpecGrp` ..................................................... 43
- `.mergeMatrices` ..................................................... 44
- `.minDif` ............................................................... 45
- `.neigbDis` .............................................................. 46
- `.normalize` ........................................................... 47
- `.normConstSlope` .................................................... 48
- `.offCenter` ............................................................ 49
- `.pasteCols` ............................................................ 49
- `.plotCountPie` ....................................................... 50
- `.plusLowerCaps` ...................................................... 51
- `.predRes` .............................................................. 52
- `.raiseColLowest` ..................................................... 52
- `.removeCol` ............................................................ 53
- `.removeEmptyCol` .................................................... 54
- `.replSpecChar` ........................................................ 55
- `.retain1stPart` ....................................................... 56
- `.rowGrpCV` ............................................................. 56
- `.rowGrpMeans` ........................................................ 57
- `.rowGrpSds` ............................................................. 58
- `.rowGrpSums` ........................................................... 58
- `.rowNorm` .............................................................. 59
- `.rowNormFact` ........................................................ 60
- `.scale01` ............................................................... 62
- `.scaleSpecGrp` ........................................................ 62
- `.scaleXY` .............................................................. 63
- `.seqCutStr` ............................................................. 64
- `.setLowestTo` ......................................................... 65
- `.sortMid` ............................................................... 65
- `.stackArray` ........................................................... 66
- `.summarizeCols` ...................................................... 67
- `.trimFromEnd` ........................................................ 68
- `.trimFromStart` ....................................................... 68
- `.trimLeft` ............................................................ 69
- `.trimRight` ............................................................ 70
- `.uniqueWName` ....................................................... 71
- `.vector2Matr` ........................................................ 72
- `addBeforFileExtension` ........................................... 72
- `adjBy2ptReg` .......................................................... 73
- `appendNR` ............................................................. 74
arrayCV .......................................................... 75
asSepList ......................................................... 76
buildTree ........................................................ 77
cbindNR .......................................................... 78
checkAvSd ......................................................... 79
checkGrpOrder .................................................... 80
checkGrpOrderSEM ............................................... 81
checkSimValueInSer ............................................. 82
checkStrictOrder ................................................. 83
checkVectLength ................................................ 84
cleanReplicates .................................................. 85
closeMatchMatrix ............................................... 86
coinPermTest ..................................................... 88
colMedSds ......................................................... 89
colorAccording2 .................................................. 89
colSds ............................................................ 91
combinatIntTable ............................................... 91
combineAsN ...................................................... 92
combineByEitherFactor ........................................ 94
combineOverlapInfo ............................................. 95
combineRedBasedOnCol ......................................... 96
combineReplFromListToMatr .................................... 97
combineSingleT .................................................. 98
completeArrLst ................................................ 99
confInt .......................................................... 100
contribToContigPerFrag ....................................... 101
tconv01toColNa ................................................ 102
tconvColorToTransp ............................................. 102
tconvMatr2df .................................................... 103
tconvToNum ...................................................... 104
tcoordOfFilt ...................................................... 106
tcorrectToUnique ............................................... 107
tcorrectWinPath ............................................... 108
tcountCloseToLimits ............................................ 109
tcountSameStartEnd ............................................. 110
tcutArrayInCluLike ............................................. 111
tcutAtMultSites ................................................ 111
tcutToNgrp ...................................................... 112
tdiffCombin ..................................................... 113
tdiffPPM ........................................................ 114
telimCloseCoord ................................................ 115
tequLenNumber .................................................. 116
texclExtrValues ............................................... 116
texponNormalize ................................................. 118
textr1chan ....................................................... 119
textractLast2numericParts .................................... 120
textrColsDeX .................................................... 121
textrNumericFromMatr .......................................... 122
extrSpcText ................................................................. 122
filt3dimArr ................................................................. 124
filterLiColDeList .......................................................... 125
filterList ................................................................. 126
filterNetw ................................................................. 127
filtSizeUniq ................................................................. 129
findCloseMatch ............................................................... 130
findRepeated ............................................................... 131
findSimilFrom2sets ......................................................... 132
findUsableGroupRange ....................................................... 134
firstLineOfDat ............................................................... 134
firstOfRepeated ............................................................. 135
firstOfRepLines ............................................................. 136
fuseAnnotMatr ............................................................... 137
fuseCommonListElem ......................................................... 138
fusePairs ................................................................. 139
get1stOfRepeatedByCol ..................................................... 140
getValuesByUnique .......................................................... 141
gitDataUrl ................................................................. 142
htmlSpecCharConv ........................................................... 143
keepCommonText ............................................................ 144
levIndex ................................................................. 146
linModelSelect .............................................................. 147
linRegrParamAndPVal ......................................................... 149
listBatchReplace ............................................................ 150
listGroupsByNames .......................................................... 151
lmSelClu ................................................................. 152
lrbind ................................................................. 153
makeMAList ................................................................. 154
makeNRedMatr ............................................................... 155
matchMatrixLinesToRef ....................................................... 156
matchNamesWithReverseParts .............................................. 158
matchSampToPairw .......................................................... 159
matr2list ................................................................. 160
mergeMatrices .............................................................. 161
mergeMatrixList ............................................................ 162
mergeSelCol ................................................................. 163
mergeSelCol3 ................................................................. 165
mergeVectors ............................................................... 166
mergeW2 ................................................................. 167
minDiff ................................................................. 169
moderTest2grp .............................................................. 170
moderTestXgrp ............................................................. 171
multiCharReplace ........................................................... 173
multiMatch ................................................................. 174
naOmit ................................................................. 175
nFragments ................................................................. 176
nFragments0 ................................................................. 177
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>nNonNumChar</td>
</tr>
<tr>
<td>nonAmbiguousMat</td>
</tr>
<tr>
<td>nonAmbiguousNum</td>
</tr>
<tr>
<td>nonredDataFrame</td>
</tr>
<tr>
<td>nonRedundLines</td>
</tr>
<tr>
<td>normalizeThis</td>
</tr>
<tr>
<td>numPairDeColNames</td>
</tr>
<tr>
<td>orderMatrToRef</td>
</tr>
<tr>
<td>organizeAsListOfRepl</td>
</tr>
<tr>
<td>packageDownloadStat</td>
</tr>
<tr>
<td>pairsAsPropensMatr</td>
</tr>
<tr>
<td>partialDist</td>
</tr>
<tr>
<td>partUnlist</td>
</tr>
<tr>
<td>pasteC</td>
</tr>
<tr>
<td>presenceFilt</td>
</tr>
<tr>
<td>presenceGrpFilt</td>
</tr>
<tr>
<td>protectSpecChar</td>
</tr>
<tr>
<td>pVal2lfdr</td>
</tr>
<tr>
<td>randIndFx</td>
</tr>
<tr>
<td>rankToContigTab</td>
</tr>
<tr>
<td>ratioAllComb</td>
</tr>
<tr>
<td>ratioToPpm</td>
</tr>
<tr>
<td>readCsvBatch</td>
</tr>
<tr>
<td>readTabulatedBatch</td>
</tr>
<tr>
<td>readVarColumns</td>
</tr>
<tr>
<td>readXlsxBatch</td>
</tr>
<tr>
<td>reduceTable</td>
</tr>
<tr>
<td>regrBy1or2point</td>
</tr>
<tr>
<td>regrMultBy1or2point</td>
</tr>
<tr>
<td>renameColumns</td>
</tr>
<tr>
<td>reorgByCluNo</td>
</tr>
<tr>
<td>replicateStructure</td>
</tr>
<tr>
<td>replNAbyLow</td>
</tr>
<tr>
<td>replPlateCV</td>
</tr>
<tr>
<td>rmDupl2colMatr</td>
</tr>
<tr>
<td>rmEnumerationName</td>
</tr>
<tr>
<td>rmnormM</td>
</tr>
<tr>
<td>rowCVs</td>
</tr>
<tr>
<td>rowGrpCV</td>
</tr>
<tr>
<td>rowGrpMeans</td>
</tr>
<tr>
<td>rowGrpNA</td>
</tr>
<tr>
<td>rowGrpSds</td>
</tr>
<tr>
<td>rowGrpSums</td>
</tr>
<tr>
<td>rowMedSds</td>
</tr>
<tr>
<td>rowNormalize</td>
</tr>
<tr>
<td>rowSds</td>
</tr>
<tr>
<td>rowSEMs</td>
</tr>
<tr>
<td>sampNoDeMArrayLM</td>
</tr>
</tbody>
</table>
**.addLetterWoLast**

*Add letter to all elements but not last*

**Description**

This function allows to add 'addChr' to all entries, without the last entry.

**Usage**

```
.addLetterWoLast(x, addChr)
```

**Arguments**

- `x` (character) main input
- `addChr` (character)

**Value**

This function returns a modified character vector.
Description

This function calculates ratio(s) for each column of matrix 'x' versus all/each column(s) of matrix 'y' (reference)

Usage

.allRatioMatr1to2(x, y, asLog2 = TRUE, sumMeth = "mean", callFrom = NULL)

Arguments

x (matrix or data.frame) main input

y (matrix or data.frame) main input

asLog2 (logical)

sumMeth (character) method

callFrom (character) allow easier tracking of messages produced

Value

This function returns a numeric vector or matrix in dimension of 'x' (so far summarize all ratios from mult division from mult ref cols as mean or median)

See Also

makeMAList, grep

Examples

.allRatioMatr1to2(matrix(11:14, ncol=2), matrix(21:24, ncol=2))
Description

This function extracts/cuts text-fragments out of `txt` following specific anchors defined by arguments `cutFrom` and `cutTo`.

Usage

```r
.allRatios(dat, ty = "log2", colNaSep = ",")
```

Arguments

dat  (matrix or data.frame) main input
ty   (character) type of ratio (eg 'log2')
colNaSep (character) separator

Value

This function returns a numeric vector

See Also

`makeMAList`, `grep`

Examples

```r
.allRatios(matrix(11:14, ncol=2))
```

Description

This function allows summarizing along columns of multiple arrays in list

`.arrLstMean`  Summarize along columns of multiple arrays in list
Usage

,arrLstMean(
    arrLst,
    sumType = "mean",
    arrOutp = FALSE,
    signifDig = 3,
    formatCheck = FALSE,
    silent = FALSE,
    debug = FALSE,
    callFrom = NULL
)

Arguments

arrLst (list) main input
sumType (character)
arrOutp (logical)
signifDig (integer)
formatCheck (logical)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

array (1st dim will be summary along cols, rows will be layers of 3rd array-dim)

See Also

used in cutArrayInCluLike

Examples

.datSlope(c(3:6))

---

.arrLstSEM  Summarize along columns of mult arrays in list

Description

This function allows summarizing along columns of mult arrays in list
Usage

.asDF2(z)

Description

This function allows converting anything to data.frame

Examples

.datSlope(c(3:6))
Arguments
  
z  (numeric vector, factor, matrix or list) main input

Value
  data.frame

See Also
  as.data.frame

Examples
  .asDF2(c(3:6))

---

Description
  This function aims to get series of values after last discontinuity

Usage
  .breakInSer(x, getFrom = "last")

Arguments
  
x  (numeric) main input
  getFrom  (character)

Value
  This function returns a numeric vector of reduced length

See Also
  dist

Examples
  .breakInSer(c(11:14,16:18))
Description

This function aims to bring most extreme value to center

Usage

`.bringToCtr(aa, ctr, ctrFa = 0.75)`

Arguments

- `aa` (numeric) main input
- `ctr` (numeric) ‘control’
- `ctrFa` (numeric <1) modulate amplitude of effect

Value

This function returns an adjusted numeric vector

See Also

dist

Examples

`.bringToCtr(11:14, 9)`

---

Description

This function allows checking of argument names

Usage

`.checkArgNa(x, argNa, lazyEval = TRUE)`

Arguments

- `x` (character) main input
- `argNa` (character) argument name
- `lazyEval` (logical) decide if argument should be evaluated with abbreviated names, too
.checkConsistentArrList

Check list of arrays for consistent dimensions of all arrays

Description

This function allows to check list of arrays for consistent dimensions of all arrays

Usage

.checkConsistentArrList(
  arrLst,
  arrNDim = 3,
  fxName = NULL,
  varName = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

arrLst (list) main input
arrNDim (integer) number of dimensions for arrays
fxName (character) this name will be given in message
varName (character)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

list

See Also

chartr

Examples

.checkArgNa("Abc", c("ab", "Ab", "BCD"))
.checkConv2Vect

See Also

used in cutArrayInCluLike

Examples

.datSlope(c(3:6))

---

.checkConv2Vect Convert to simple vector (similar to unlist)

Description

This function allows converting 'dat' (may be list, data.frame etc) to simple vector, more elaborate than unlist()

Usage

.checkConv2Vect(dat, toNumeric = TRUE)

Arguments

dat (list, data.frame) main input
toNumeric (logical)

Value

character (or numeric) vector

See Also

unlist; used in equLenNumber

Examples

aa <- matrix(11:14, ncol=2)
.checkConv2Vect(aa)
.checkFactor  

**Description**

This function was designed to check a factor object

**Usage**

```r
.checkFactor(  
  fac,  
  facNa = NULL,  
  minLev = 2,  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL
)
```

**Arguments**

- `fac` (factor) main input
- `facNa` (character) level-names
- `minLev` (integer) minimum number of levels
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

**Value**

This function returns a corrected/adjusted factor

**See Also**

`factor`

**Examples**

```r
.checkFactor(gl(3,2))
```
checkFileNameExtensions Function for checking file-names.

Description

checkFileNameExtensions Function for checking file-names.

Usage

.checkFileNameExtensions(fileNa, ext)

Arguments

fileNa (character) file name to be checked
ext (character) file extension

Value

modified character vector

Examples

.checkFileNameExtensions("testFile.txt","txt")

Check argument for Location of legend

Description

This function allows checking an argument for Location of legend, if value provided not found as valid, it returns `defLoc

Usage

.checkLegendLoc(
  legLoc,
  defLoc = "topright",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>legLoc</td>
<td>(character) main input</td>
</tr>
<tr>
<td>defLoc</td>
<td>(character)</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>debug</td>
<td>(logical) additional messages for debugging</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allow easier tracking of messages produced</td>
</tr>
</tbody>
</table>

Value

This function returns a character vector designing the potential location of legend

See Also

legend

Examples

```r
.checkLegendLoc("abc")
```

---

Description

This function allows to compare 'dat' to confidence interval of linear model 'lMod' (eg from \( \text{lm}() \))

Usage

```r
.checkLmConfInt(dat, lMod, level = 0.95)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat</td>
<td>matrix or data.frame, main input</td>
</tr>
<tr>
<td>lMod</td>
<td>linear model, only used to extract coefficients offset &amp; slope</td>
</tr>
<tr>
<td>level</td>
<td>(numeric) alpha threshold for linear model</td>
</tr>
</tbody>
</table>

Value

This function returns a logical vector for each value in 2nd col of 'dat' if INSIDE confidence interval

See Also

searchLinesAtGivenSlope

Examples

```r
set.seed(2016); dat1 <- matrix(c(rnorm(200)+rep(1:10,20)),ncol=10)
```
.checkRegrArguments Check regression arguments

Description
This function allows to check arguments for linear regression. Used as argument checking for `regrBy1or2point` and `regrMultBy1or2point`

Usage
`.checkRegrArguments(inData, refList, regreTo, callFrom = NULL)`

Arguments
- `inData` (numeric vector): main input
- `refList` (list)
- `regreTo` (numeric vector)
- `callFrom` (character): allow easier tracking of messages produced

Value
list

See Also
- `append`, `lrbind`

Examples
`.datSlope(c(3:6))`

.chooseGrpCol Automatic choice of colors

Description
This function allows to do automatic choice of colors: if single-> grey, if few -> RColorBrewer, if many : gradient green -> grey/red

Usage
`.chooseGrpCol(nGrp, paired = FALSE, alph = 0.2)`
Arguments

nGrp  (numeric vector) main input
paired  (logical)
alph  (numeric vector)

Value

character vector with color codes

See Also

`rgb`; `colorAccording2`

Examples

chooseGrpCol(4)

------------------------------------------------------------------------
.combineListAnnot  Combine annotation information from list of matrices
------------------------------------------------------------------------

Description

This function allows to combine information (annotation) from list of matrixes (ie replace when NA), using always the columns specified in ‘useCol’ (numeric)

Usage

.combineListAnnot(
  lst,
  useCol = 1:2,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

lst  (list) main input
useCol  (numeric vector) which columns should be used
silent  (logical) suppress messages
debug  (logical) additional messages for debugging
callFrom  (character) allow easier tracking of messages produced

Value

This function returns a single matrix of combined (non-redundant) info
.compareByDiff

See Also

used in cutArrayInCluLike

Examples

.datSlope(c(3:6))
.compareByLogRatio  
*Compare by log-ratio*

**Description**

This function allows to compare by log-ratio

**Usage**

```
.compareByLogRatio(dat, limit, distVal = FALSE)
```

**Arguments**

- `dat`: list of 2 numerical vectors
- `limit`: (numeric, length=1) threshold value for retaining values, used with distance-type specified in argument 'compTy'
- `distVal`: (logical) to toggle output as matrix of numeric (distance values above 'limit', others NA) or matrix of logical

**Value**

This function returns a list with close matches of 'x' to given 'y', the numeric value depends on 'sortMatch' (if FASLE then always value of 'y' otherwise of longest of x&y)

**See Also**

`findCloseMatch, checkSimValueInSer`, and also `.compareByDiff`, for convient output `countCloseToLimits`

**Examples**

```
cc <- list(aa=11:14, bb=c(13.1,11.5,14.3,20:21))
.compareByLogRatio(cc, 1)
```

---

.compareByPPM  
*Compare by PPM*

**Description**

This function allows to compare by ppm

**Usage**

```
.compareByPPM(dat, limit, distVal = FALSE)
```
### complCols

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat</td>
<td>list of 2 numerical vectors</td>
</tr>
<tr>
<td>limit</td>
<td>(numeric, length=1) threshold value for retaining values, used with distance-type specified in argument 'compTy'</td>
</tr>
<tr>
<td>distVal</td>
<td>(logical) to toggle output as matrix of numeric (distance values above 'limit', others NA) or matrix of logical</td>
</tr>
</tbody>
</table>

**Value**

This function returns a list with close matches of 'x' to given 'y', the numeric value depends on 'sortMatch' (if FALSE then always value of 'y' otherwise of longest of x&y)

**See Also**

`findCloseMatch`, `checkSimValueInSer`, and also `.compareByDiff`, for convenient output `countCloseToLimits`

**Examples**

```r
cc <- list(aa=11:14, bb=c(13.1,11.5,14.3,20:21))
.complCols(cc, 1)
```

---

### complCols

*Search (complementing) columns for best coverage of non-NA data for rowNormalization (main)*

**Description**

This function was designed to complete the selection of columns of sparse matrix 'dat' with sets of 'nCombin' columns at complete 'coverage' Context: In sparse matrix 'dat' search subsets of columns with some rows as complete (no NA).

**Usage**

`.complCols(x, dat, nCombin)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>(integer, length=1) column number for with other columns to combine &amp; give (some) complete non-NA lines are sought</td>
</tr>
<tr>
<td>dat</td>
<td>(matrix) .. init data, smay be parse matrix with numerous NA</td>
</tr>
<tr>
<td>nCombin</td>
<td>(integer) .. number of columns used to make complete subset</td>
</tr>
</tbody>
</table>

**Value**

This function returns a matrix of column-indexes complementing (nCombin rows)
See Also

- rowNormalize

Examples

```r
.complCols(3, dat=matrix(c(NA,12:17,NA,19),ncol=3), nCombin=3)
```

---

### .composeCallName

#### Compose sequence of (function-)calls

**Description**

This function was designed for tracing the hierarchy of function-calls. It allows to remove any tailing space or '-' from `callFrom` (character vector) and return with added `newNa` (+ `add2Tail`)

**Usage**

```r
.composeCallName(newNa, add2Head = "", add2Tail = " : ", callFrom = NULL)
```

**Arguments**

- `newNa` (character vector) main input
- `add2Head` (character)
- `add2Tail` (character)
- `callFrom` (character) may also contain multiple separate names (ie length >1), will be concatenated using `->'

**Value**

- character vector (history of who called whom)

**See Also**

- paste

**Examples**

```r
.composeCallName("newFunction", callFrom="initFunction")
```
**.convertMatrToNum**  
*Convert numeric matrix to numeric*

**Description**  
Take matrix and return vector

**Usage**  
```r  
.convertMatrToNum(matr, useCol = NULL)  
```

**Arguments**  
- **matr**  
  (matrix) main input  
- **useCol**  
  (integer) design the columns to be used

**Value**  
numeric vector

**See Also**  
`matrix`

**Examples**  
```r  
.convertMatrToNum(matrix(1:6, ncol=2))  
```

---

**.convertNa**  
*Convert/standardize names of 'query' to standard names from 'ref'*

**Description**  
This function converts/standardizes names of 'query' to standard names from 'ref' (list of possible names (char vect) where names define standardized name). It takes 'query' as character vector and return character vector (same length as 'query') with 'converted/corrected' names

**Usage**  
```r  
.convertNa(query, ref, partMatch = TRUE)  
```

**Arguments**  
- **query**  
  (matrix or data.frame, min 2 columns) main input  
- **ref**  
  (list) list of multiple possible names associated to given group, reference name for each group is name of list  
- **partMatch**  
  (logical) allows partial matching (ie name of 'ref' must be in head of 'query')
Value

This function returns a character vector

Examples

daPa <- matrix(c(1:5,8,2:6,9), ncol=2)

d.corDuplItemsByIncrem(letters[1:6], letters[8:4])

Description

This function aims to avoid duplicating items between `curNa` and `newNa` by incrementing digits after `extPref` (in newNa)

Usage

`.corDuplItemsByIncrem(newNa, curNa, extPref = "_s")`

Arguments

- `newNa` (character) main input 1
- `curNa` (character) main input 2
- `extPref` (character) extension

Value

This function returns the corrected input vector `newNa`

See Also

duplicated

Examples

`.corDuplItemsByIncrem(letters[1:6], letters[8:4])`
.cutAtSearch

Search character-string and cut either before or after

Description

This function extracts/cuts text-fragments out of txt following specific anchors defined by arguments cutFrom and cutTo.

Usage

`.cutAtSearch(
  x,
  searchChar,
  after = TRUE,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

x character vector to be treated
searchChar (character) text to look for
after (logical)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a modified character vector

See Also

grep

Examples

`.cutAtSearch("abcdefg","de")`
.cutStr  

Cut string to get all variants from given start with min and max length

Description

This function allows truncating character vector to all variants from given start, with min and optional max length. Used to evaluate argument calls without giving full length of argument.

Usage

.cutStr(txt, startFr = 1, minLe = 1, maxLe = NULL, reverse = TRUE)

Arguments

txt     (character) main input, may be length >1
startFr (integer) where to start
minLe   (integer) minimum length of output
maxLe   (integer) maximum length of output
reverse (logical) return longest text-fragments at beginning of vector

Value

This function returns a character vector.

See Also

used in pasteC; substr

Examples

.cutStr("abcdefg", minLe=2)

.datSlope  

Model linear regression and optional plot

Description

This function allows to model a linear regression and optionally to plot the results.
Usage

.datSlope(
  dat,
  typeOfPlot = "sort",
  toNinX = FALSE,
  plotData = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dat (vector or matrix) main input
typeOfPlot (character)
toNinX (logical)
plotData (logical)
silent (logical) suppress messages
debug (logical) display additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

numeric vector with intercept and slope, optional plot

See Also

append; lrbind

Examples

.datSlope(c(3:6))

Description

This function allows extracting NA-neighbour value

Usage

.extrNAneighb(x, grp)
Arguments

- **x**: initial matrix to treat
- **grp**: (factor) grouping of replicates

Value

numeric vector

See Also

unique, nonAmbiguousNum, faster than firstOfRepeated which gives more detail in output (lines/elements/indexes of omitted)

Examples

```
.extrNAneighb(c(11:14,NA), rep(1,5))
```

Description

This function aims to extract number(s) before capital character

Usage

```
.extrNumHeadingCap(x)
```

Arguments

- **x**: character vector to be treated

Value

This function returns a numeric vector

See Also

grep, nchar

Examples

```
.extrNumHeadingCap(" 1B ")
```
.extrNumHeadingSepChar

Extract numbers before separator followed by alphabetic character

Description
This function aims to extract number(s) before separator followed by alphabetic character (return named numeric vector, NAs when no numeric part found)

Usage
.extrNumHeadingSepChar(x, sep = "_")

Arguments
x character vector to be treated
sep (character) separator

Value
This function returns a numeric vector

See Also
nchar

Examples
.extrNumHeadingSepChar("1B")

.filterNetw

Filter nodes & edges for extracting networks (main) This function allows extracting and filtering network-data based on fixed threshold (limInt) and add sandwich-nodes (nodes inter-connecting initial nodes) out of node-based queries.

Description
Filter nodes & edges for extracting networks (main)
This function allows extracting and filtering network-data based on fixed threshold (limInt) and add sandwich-nodes (nodes inter-connecting initial nodes) out of node-based queries.
Usage

.filterSw(
  lst,
  remOrphans = TRUE,
  reverseCheck = TRUE,
  filtCol = 2,
  callFrom = NULL,
  silent = FALSE,
  debug = FALSE
)

Arguments

lst (list, composed of multiple matrix or data.frames) main input (each list-element should have same number of columns)
remOrphans (logical) remove networks consisting only of 2 connected edges
reverseCheck (logical)
filtCol (integer, length=1) which column of lst should be used to filter using thresholds limInt and sandwLim
callFrom (character) allow easier tracking of message(s) produced
silent (logical) suppress messages
debug (logical) display additional messages for debugging

Value

This function returns a matrix or data.frame

See Also

.filterNetw and other CRAN package dedicated to networks

Examples

ab <- 1:10

$filterSw Filter 3-dim array of numeric data (main)

Description

Filtering of matrix or array x (may be 3-dim array) according to fiTy and checkVa

Usage

$filterSw(x, fiTy, checkVa, indexRet = TRUE)
Arguments

x array (3-dim) of numeric data
fiTy (character) which type of testing to perform ('eq', 'inf', 'infeq', 'sup', 'supeq', '>', '<', '>=', '<=', '==')
checkVa (logical) s
indexRet (logical) if TRUE (default) rather return index numbers than filtered values

Value

This function returns either index (position within 'x') or concrete (filtered) result

See Also

filt3dimArr; filterList; filterLiColDeList;

Examples

arr1 <- array(11:34, dim=c(4,3,2), dimnames=list(c(LETTERS[1:4]), paste("col",1:3,sep=""),c("ch1","ch2")))
filt3dimArr(arr1, displCrit=c("col1","col2"), filtCrit="col2", filtVal=7)
.filterSw(arr1, fiTy="inf", checkVa=7)

Description

This function aims to filter for size

Usage

.filtSize(x, minSize = 5, maxSize = 36)

Arguments

x main input
minSize (integer) minimum number of characters, if NULL set to 0
maxSize (integer) maximum number of characters

Value

list of filtered input

See Also

filtSizeUniq; correctToUnique, unique, duplicated
.findBorderOverlaps

Examples

```r
aa <- 1:10
```

Description

This function aims to find overlap instances among range of values in lines of `x` (typically give just min & max)

Usage

```r
.findBorderOverlaps(x, rmRedund = FALSE, callFrom = NULL)
```

Arguments

- `x` (matrix of numeric values or all-numeric data.frame) main input
- `rmRedund` (logical) report overlaps only in 1st instance (will show up twice otherwise)
- `callFrom` (character) allow easier tracking of message(s) produced

Value

This function returns a matrix with line for each overlap found, cols 'refLi' (line no), 'targLi' (line no), 'targCol' (col no)

See Also

- `nchar`

Examples

```r
aa <- 11:15
```
.firstMin

Description
This function allows to find the first minimum of a numeric vector.

Usage
.firstMin(x, positionOnly = FALSE)

Arguments
x (numeric vector) main input
positionOnly (logical)

Value
numeric vector

See Also
which.min

Examples
.firstMin(c(4,3:6))

.fuse2ArrBy2ndDim

Description
This function allows fusing 2 instances of 3dim arr as mult cols in 3dim array (ie fuse along 2nd dim, increase cols)

Usage
.fuse2ArrBy2ndDim(arr1, arr2, silent = FALSE, debug = FALSE, callFrom = NULL)

Arguments
arr1 (array)
arr2 (array)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced
Value

This function returns a numeric vector with number of non-numeric characters (i.e., not '.', 0-9).

See Also

array

Examples

aa <- 11:15

---

Description

This function calculates the 'A' value (i.e., group mean) for each group of replicates (e.g., for MA-plot).

Usage

.getAmean(dat, grp)

Arguments

dat (matrix or data.frame) main input
grp (factor) grouping of replicates

Value

This function returns a numeric vector

See Also

makeMAList

Examples

.getAmean(matrix(11:18, ncol=4), gl(2,2))
.getAmean2

Get A value for each group of replicates based on comp

Description

This function calculates the 'A' value (ie group mean) for each group of replicates (eg for MA-plot) comp is matrix telling which groups to use/compare, assuming that dat are already group-means

Usage

.getAmean2(dat, comp)

Arguments

dat (matrix or data.frame) main input
comp (matrix) tells which groups to use/compare, assuming that dat are already group-means

Value

This function returns a numeric vector

See Also

makeMAList

Examples

.getAmean(matrix(11:18, ncol=4), gl(2,2))

.getMvalue2

Get M value for each group of replicates based on comp

Description

This function calculates the 'M' value (ie log-ratio) for each group of replicates based on comp (eg for MA-plot) comp is matrix telling which groups to use/compare, assuming that dat are already group-means

Usage

.getMvalue2(dat, comp)
Arguments

dat (matrix or data.frame) main input
comp (matrix) tells which groups to use/compare, assuming that dat are already group-means

Value

This function returns a numeric vector

See Also

makeMAList

Examples

.getAmean(matrix(11:18, ncol=4), gl(2,2))

---

Description

This function allows growing tree-like structures (data.tree objects)

Usage

.growTree(tm, setX, addToObj = NULL)

Arguments

tm (list) main input, $disDat .. matrix with integer start & end sites for fragments; $lo (logical) which fragments may be grown; $start (integer) index for which line of $disDat to start; $it numeric version of $lo; $preN for previous tree objects towards root; $iter for iterator (starting at 1))
setX .. data.tree object (main obj from root)
addToObj .. data.tree object (branch on which to add new branches/nodes)

Value

list

See Also

buildTree

Examples

.datSlope(c(3:6))
Segment (1-dim vector) 'dat' into clusters

Description

This function allows segmenting (1-dim vector) 'dat' into clusters. If 'automClu=TRUE ..' first try automatic clustering, if too few clusters, run km with length(dat)^0.3 clusters This function requires the package NbClust to be installed.

Usage

.insp1dimByClustering(
  dat,
  automClu = TRUE,
  cluChar = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dat      matrix or data.frame, main input
automClu  (logical) run automatic clustering
cluChar   (logical) to display cluster characteristics
silent    (logical) suppress messages
debug     (logical) additional messages for debugging
callFrom  (character) allow easier tracking of messages produced

Value

This function returns clustering (class index) or (if ’cluChar’=TRUE) list with clustering and cluster-characteristics

See Also

searchLinesAtGivenSlope

Examples

set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
Description

This function inspects 'matr' and check if 1st line can be used/converted as header. If colnames of 'matr' are either NULL or 'V1', etc the 1st row will be tested if it contains any of the elements (if not, 1st line won't be used as new colnames) If 'numericCheck'=TRUE, all columns will be tested if they can be converted to numeric

Usage

```r
.inspectHeader(
  matr,
  headNames = c("Plate", "Well", "StainA"),
  numericCheck = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- `matr` (matrix or data.frame) main input to be instected
- `headNames` (character) column-names t look for
- `numericCheck` (logical) allows reducing complexity by drawing for very long x or y
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Value

This function returns a matrix vector or data.frame similar to input

See Also

- `head` for looking at first few lines

Examples

```r
ma1 <- matrix(letters[1:6], ncol=3, dimnames=list(NULL,c("ab","Plate","Well")))
.inspectHeader(ma1)
```
Refine/filter 'dat1' (1dim dataset, eg cluster) with aim of keeping center of data

Description
This function allows to refine/filter 'dat1' (1dim dataset, eg cluster) with aim of keeping center of data. It is done based on most freq class of histogramm keep/filter data if 'core' (  

Usage
.keepCenter1d(
  dat1,
  core = NULL,
  keepOnly = TRUE,
  displPlot = FALSE,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)

Arguments
  dat1 simple numeric vector
  core numeric vector (betw 0 and 1) for fraction of data to keep; if null trimmed-
               Mean/max hist occurrence will be used, limited within 30-70 percent; may also be
               'high' or 'low' for forcing low (20-60percent) or high (75-99) percent of data to
               retain
  keepOnly (logical)
  displPlot (logical) show plot of hist & boundaries
  silent (logical) suppress messages
  debug (logical) additional messages for debugging
  callFrom (character) allow easier tracking of messages produced

Value
This function returns the index of values retained or if 'keepOnly' return list with 'keep' index and 'drop' index

See Also
  searchLinesAtGivenSlope

Examples
  set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
Description

This function aims to remove all columns where all data are not finite

Usage

```r
.keepFiniteCol(
  dat,  
  msgStart = NULL, 
  silent = FALSE, 
  debug = FALSE, 
  callFrom = NULL
)
```

Arguments

dat (matrix or data.frame) main input
msgStart (character)
silent (logical) suppres messages
debug (logical) additional messages for debugging
callFrom (character) allows easier tracking of messages produced

Value

This function returns a corrected matrix or data.frame

See Also

renameColumns; is.finite

Examples

```r
ma1 <- matrix(c(1:5, Inf), ncol=2)
.keepFiniteCol(ma1)
```
mightBeNum

Check if vector may be numeric content

Description
This function allows to checking if a given vector may be numeric content

Usage
.mightBeNum(x, pattern = NULL)

Arguments
x (numeric vector) main input
pattern (character) custom pattern to check

Value
This functions returns a logical/boolean vector for each of the elements of 'x'

See Also
numeric; convMatr2df

Examples
.mightBeNum(c(3:6))

medianSpecGrp

Rescale respective to specific group

Description
This function allows to rescale data 'x' so that specific group 'grpNum' gets normalized to predefined value 'grpVal'. In normal case x will be multiplied by 'grpVal' and devided by value obtained from 'grpNum'. If summary of 'grpNum-positions' or 'grpVal' is 0, then grpVal will be attained by subtraction of summary & adding grpVal

Usage
.medianSpecGrp(x, grpNum, grpVal, sumMeth = "median", callFrom = NULL)
.mergeMatrices

Arguments

x (numeric vector) main input
grpNum (numeric)
grpVal (numeric)
sumMeth (character) method for summarizing
callFrom (character) allow easier tracking of messages produced

Value

numeric vector

See Also

which.min

Examples

.firstMin(c(4,3:6))

Description

This function allows merging of multiple matrix-like objects from an initial list.

Usage

.mergeMatrices(
  inpL,
  mode = "intersect",
  useColumn = 1,
  extrRowNames = FALSE,
  na.rm = TRUE,
  argL = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

inpL: (list containing matrices or data.frames) main input (multiple matrix or data.frame objects)
mode: (character) allows choosing restricting to all common elements (mode='intersect') or union (mode='union')
useColumn: (integer, character or list) the column(s) to consider, may be 'all' to use all, integer to select specific indexes or list of indexes or colnames for custom-selection per matrix
extrRowNames: (logical) decide whether columns with all values different (ie no replicates or max divergency) should be excluded
na.rm: (logical) suppress NAs
argL: (list of arguments)
silent: (logical) suppress messages
debug: (logical) additional messages for debugging
callFrom: (character) allow easier tracking of messages produced

Value

This function returns a matrix containing all selected columns of the input matrices to fuse

See Also

mergeMatrixList, merge, mergeMatrices for separate entries

Examples

mat1 <- matrix(11:18, ncol=2, dimnames=list(letters[3:6],LETTERS[1:2]))

.minDif: find closest neighbour to numeric vector

Description

This function aims to find closest neighbour to numeric vector

Usage

.minDif(z, initOrder = TRUE, rat = TRUE)

Arguments

z: (numeric) vector to search minimum difference
initOrder: (logical) return matrix so that 'x' matches exactly 2nd col of output
rat: (logical) express result as ratio
Value

This function returns a matrix with index, value, dif, best

See Also

dist

Examples

e.g. minDif(c(11:15,17))

Description

This function returns distances between sorted points of 2-column matrix 'x'

Usage

.neigbDis(x, asSum = TRUE)

Arguments

x              (matrix or data.frame, min 2 columns) main input
asSum          (logical) if TRUE (default) the sum of all distances will be returned, otherwise the individual distances

Value

This function returns a numeric vector with distances

Examples

daPa <- matrix(c(1:5,8,2:6,9), ncol=2)
.neigbDis(daPa)
.normalize

Main Normalization function

Description

This function aims to normalize a matrix or data.frame by columns. It assumes all checks have been done before calling this function.

Usage

.normalize(
  dat,
  meth,
  mode,
  param,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dat          matrix or data.frame of data to get normalized
meth         (character) may be "mean","median","NULL","none", "trimMean", "rowNormalize", "slope", "exponent", "slope2Sections", "vsn"; When NULL or 'none' is chosen the input will be returned
mode         (character) may be "proportional", "additive"; decide if normalization factors will be applied as multiplicative (proportional) or additive; for log2-omics data mode="additive" is suggested
param       (list) additional parameters
silent   (logical) suppress messages
debug    (logical) additional messages for debugging
callFrom  (character) allows easier tracking of messages produced

Value

This function returns a numeric vector

See Also

normalizeThis

Examples

aa <- matrix(1:12, ncol=3)
.normalize(aa,"median",mode="proportional",param=NULL)
.normConstSlope

Normalize columns of 2dim matrix to common linear regression fit

Description

This function aims to normalize columns of 2dim matrix to common linear regression fit within range of 'useQuant'

Usage

.normConstSlope(
  mat, 
  useQuant = c(0.2, 0.8), 
  refLines = NULL, 
  diagPlot = TRUE, 
  plotLog = "", 
  datName = NULL, 
  silent = FALSE, 
  debug = FALSE, 
  callFrom = NULL
)

Arguments

mat matrix or data.frame of data to get normalized
useQuant (numeric) quantiles to use
refLines (NULL or numeric) allows to consider only specific lines of 'dat' when determining normalization factors (all data will be normalized)
diagPlot (logical) draw diagnostic plot
plotLog (character) indicate which axis should be displayed on log-scale, may be 'x', 'xy' or 'y'
datName (character) use as title in diag plot
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allows easier tracking of messages produced

Value

This function returns a numeric vector

See Also

normalizeThis

Examples

aa <- matrix(1:12, ncol=3)
.offCenter

Return position of 'di' (numeric vector) which is most excentric (distant to 0), starts with NAs as most excentric

Description

This function aims to return position of 'di' (numeric vector) which is most excentric (distant to 0), starts with NAs as most excentric It is used for identifying/removing (potential) outliers. Note : this fx doesn’t consider reference distributions, even with "perfect data" ‘nMost’ points will ba tagged !

Usage

.offCenter(di, nMost = 1)

Arguments

di (numeric) main input
nMost (integer)

Value

This function returns a integer/numeric vector (indicating index)

See Also

use in presenceFilt; diff

Examples

.offCenter(11:14)

.pasteCols

Paste-concatenate all columns of matrix

Description

This function allows paste columns

Usage

.pasteCols(mat, sep = "")

Arguments

mat initial matrix
sep (character) separator
Value

simplified/non-redundant vector/matrix (ie fewer lines for matrix), or respective index

See Also

unique, nonAmbiguousNum, faster than firstOfRepeated which gives more detail in output (lines/elements/indexes of omitted)

Examples

`.pasteCols(matrix(11:16,ncol=2), sep=" ")`

Description

This function allows to inspect results of table or uniqCountReport on a pie-plot Note: fairly slow for long vectors !!

Usage

```r
.plotCountPie(count, tit = NULL, col = NULL, radius = 0.9, sizeTo = NULL, clockwise = FALSE, silent = FALSE, debug = FALSE, callFrom = NULL)
```

Arguments

count (integer vector) counting result
tit (character) optional title in plot
col (character) custom colors in pie
radius (numeric) radius passed to pie
sizeTo (numeric or character) optional reference group for size-population relative adjusting overall surface of pie
clockwise (logical) argument passed to pie
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced
Value

vector with counts of n (total), nUnique (wo any repeated), nHasRepeated (first of repeated), nRedundant, optional figure

See Also

uniqCountReport, correctToUnique, unique

Examples

.plotCountPie(table(c(1:5,4:2)))

---

.plusLowerCaps  

Add lower caps to character vector

Description

This function allows adding all content as lower caps to/of character vector

Usage

.plusLowerCaps(x)

Arguments

x  
(character) main input

Value

This function returns a elongated character vector

See Also

chartr

Examples

.plusLowerCaps(c("Abc","BCD"))
### .predRes

**Calculate residues of (2-dim) linear model 'lMod'-prediction of/for 'dat'**

#### Description

This function calculates residues of (2-dim) linear model 'lMod'-prediction of/for 'dat' (using 2nd col of 'useCol') (indexing in 'dat', matrix or data.frame with min 2 cols), using 1st col of 'useCol' as 'x'. It may be used for comparing/identifying data close to regression (eg re-finding data on autoregression line in FT-ICR)

#### Usage

```
.predRes(dat, lMod, regTy = "lin", useCol = 1:2)
```

#### Arguments

- **dat**: matrix or data.frame, main input
- **lMod**: linear model, only used to extract coefficients offset & slope
- **regTy**: (character) type of regression model
- **useCol**: (integer) columns to use

#### Value

This function returns a numeric vector of residues (for each line of dat)

#### See Also

- `searchLinesAtGivenSlope`

#### Examples

```
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
```


### .raiseColLowest

**Raise all values close to lowest value**

#### Description

This function aims to raise all values close to lowest value to end up as at value of 'raiseTo'. This is done independently for each col of mat. This function sets all data to common raiseTo (which is min among all cols)
Usage

```r
.raiseColLowest(
  mat,
  raiseTo = NULL,
  minFa = 0.1,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- `mat` (matrix of numeric values) main input
- `raiseTo` (numeric)
- `minFa` (numeric) minimum factor
- `silent` (logical) suppress messages
- `debug` (logical) display additional messages for debugging
- `callFrom` (character) allow easier tracking of message(s) produced

Value

This function returns a numeric vector with number of non-numeric characters (ie not '.' or 0-9))

See Also

- `nchar`

Examples

```r
aa <- 11:15
```

Description

This function aims to remove columns indicated by col-number

Usage

```r
.removeCol(matr, rmCol)
```

Arguments

- `matr` (matrix or data.frame) main input
- `rmCol` (integer) column index for removing
Value

This function returns a corrected matrix or data.frame

See Also

dist

Examples

```r
aa <- matrix(1:6, ncol=3)
.removeCol(aa, 2)
```

Description

This function aims to search for (empty) columns containing only entries defined in 'searchFields' and remove such columns. If 'fromBackOnly' = TRUE .. only tailing empty columns will be removed (other columns with "empty" entries in middle will be kept). If "=TRUE columns containing all NAs will be excluded as well. This function will also remove columns containing (exclusively) mixtures of the various 'searchFields'.

Usage

```r
.removeEmptyCol(
  dat,
  fromBackOnly = TRUE,
  searchFields = c("", " ", "NA.", NA),
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

dat (matrix or data.frame) main input
fromBackOnly (logical)
searchFields (character)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allows easier tracking of messages produced

Value

This function returns a corrected matrix or data.frame
.replSpecChar

See Also

renameColumns; is.finite

Examples

ma1 <- matrix(c(1:5, NA), ncol=2)
.removeEmptyCol(ma1)

. replSpecChar

Replace Special Characters

Description

This function allows replacing special characters Note that (most) special characters must be presented with protection for grep and sub.

Usage

.replSpecChar(x, findSp = c("\(", "\)", "\$"), replBy = ".")

Arguments

x (character) main input
findSp (character) special characters to replace (may have to be given as protected)
replBy (character) replace by

Value

This function returns a corrected/adjusted factor

See Also

factor

Examples

.replSpecChar(c("jhjh(ab)"."abc"))
.retain1stPart

Trim character string: keep only text before 'sep'

Description

Trim character string: keep only text before 'sep' (length=1 !)

Usage

.retain1stPart(chr, sep = " = ", offSet = 1)

Arguments

chr character vector to be treated
sep (character) separator
offSet (integer) offset

Value

This function returns a modified character vector

See Also

substr

Examples

.retain1stPart("abc = def")

.rowGrpCV

row group CV (main)

Description

This function calculates CVs for matrix with multiple groups of data, i.e., one CV for each group of data.

Usage

.rowGrpCV(x, grp, means)

Arguments

x numeric matrix where replicates are organized into separate columns
grp (factor) defining which columns should be grouped (considered as replicates)
means (numeric) alternative values instead of means by .rowGrpMeans()
.rowGrpMeans

**Value**

This function returns a matrix of CV values

**See Also**

rowGrpCV, rowCVs, arrayCV, replPlateCV

**Examples**

```r
grun <- makeGrid(20, 10, height = 5, site = 4)  # (in addition to set.seed)

data <- matrix(rnorm(1000), ncol = 10)

g <- rowGrpMeans(data, gl(4, 20, labels = LETTERS[1:4])[-1])
```

---

### .rowGrpMeans

row group mean (main)

**Description**

This function calculates CVs for matrix with multiple groups of data, ie one CV for each group of data.

**Usage**

```r
=rowGrpMeans(x, grp, na.replVa = NULL, na.rm = TRUE)
```

**Arguments**

- `x`: numeric matrix where replicates are organized into separate columns
- `grp` (factor) defining which columns should be grouped (considered as replicates)
- `na.replVa` (numeric) value to replace NA values
- `na.rm` (logical) remove all NA values

**Value**

This function returns a matrix of mean values per row and group of replicates

**See Also**

rowGrpCV, rowCVs, arrayCV, replPlateCV

**Examples**

```r
grun <- makeGrid(20, 10, height = 5, site = 4)  # (in addition to set.seed)

data <- matrix(rnorm(1000), ncol = 10)

g <- rowGrpMeans(data, gl(4, 20, labels = LETTERS[1:4])[-1])
```
.rowGrpSds

### Description

This function calculates sd for matrix with multiple groups of data, ie one sd for each group of data.

### Usage

```
.rowGrpSds(x, grp)
```

### Arguments

- **x**: numeric matrix where replicates are organized into separate columns
- **grp**: (factor) defining which columns should be grouped (considered as replicates)

### Value

This function returns a matrix of sd values per row and group of replicates

### See Also

- `rowGrpCV`, `rowCVs`, `arrayCV`, `replPlateCV`

### Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
grp1 <- gl(4,3,labels=LETTERS[1:4])[2:11]
head(.rowGrpSds(dat1, grp1))
```

---

.rowGrpSums

### Description

This function calculates row-sums for matrix with multiple groups of data, with multiple groups of data, ie one sd for each group of data.

### Usage

```
.rowGrpSums(x, grp, na.replVa = NULL, na.rm = TRUE)
```

### See Also

- `rowGrpCV`, `rowCVs`, `arrayCV`, `replPlateCV`
.rowNorm

Arguments

- x: numeric matrix where replicates are organized into separate columns
- grp: (factor) defining which columns should be grouped (considered as replicates)
- na.replVa: (numeric) value to replace NA values
- na.rm: (logical) remove all NA values

Value

This function returns a matrix of row-sums for matrix with multiple groups of data

See Also

rowGrpCV, rowCVs, arrayCV, replPlateCV

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
grp1 <- gl(4,3,labels=LETTERS[1:4])[2:11]
head(.rowGrpSums(dat1, grp1))
```

Description

This function was performs a row-normalization procedure on matrix or data.frame 'dat'

Usage

```r
.rowNorm(
  dat,
  refLi,
  method,
  proportMode,
  maxFact = 10,
  fact0val = 10,
  retFact = FALSE,
  callFrom = NULL,
  debug = FALSE,
  silent = FALSE
)
```
Arguments

dat (matrix) .. init data, smay be parse matrix with numerous NA
refLi (NULL or numeric) allows to consider only specific lines of 'dat' when determining normalization factors (all data will be normalized)
method (character) may be "mean","median" (plus "NULL","none"); When NULL or 'none' is chosen the input will be returned as is
proportMode (logical) decide if normalization should be done by multiplicative or additive factor
maxFact (numeric, length=2) max normalization factor
fact0val (integer)
retFact (logical)
callFrom (character) This function allows easier tracking of messages produced
debug (logical) additional messages for debugging
silent (logical) suppress messages

Value

This function returns a matrix of normalized data same dimensions as 'dat'

See Also

rowNormalize

Examples

.rowNorm(matrix(11:31, ncol=3), refLi=1, method="mean", proportMode=TRUE)

.rowNormFact

\begin{itemize}
  \item \textit{Obtain normalization factor (main)}
\end{itemize}

Description

This function was designed to obtain normalization factors.

Usage

.rowNormFact(
  dat,
  combOfN,
  comUse,
  method = "median",
  refLi = NULL,
  refGrp = NULL,
  proportMode = TRUE,
Arguments

- **dat** (matrix): .. init data, smay be parse matrix with numerous NA
- **combOfN** (matrix): .. # matrix of index for all sub-groups (assumed as sorted)
- **comUse** (list): .. index of complete lines for each col of combOfN
- **method** (character): may be "mean","median" (plus "NULL","none"); When NULL or 'none' is chosen the input will be returned as is
- **refLi** (NULL or numeric): allows to consider only specific lines of 'dat' when determining normalization factors (all data will be normalized)
- **refGrp** (integer): Only the columns indicated will be used as reference, default all columns (integer or colnames)
- **proportMode** (logical): decide if normalization should be done by multiplicative or additive factor
- **minQuant** (numeric): optional filter to set all values below given value as NA
- **maxFact** (numeric, length=2): max normalization factor
- **omitNonAlignable** (logical): allow omitting all columns which can’t get aligned due to sparseness
- **silent** (logical): suppress messages
- **debug** (logical): additional messages for debugging
- **callFrom** (character): This function allows easier tracking of messages produced

Value

This function returns a matrix of column-indexes complementing (nCombin rows)

See Also

- rowNormalize

Examples

```r
ma1 <- matrix(11:41, ncol=3)
```
.scale01  Scale between 0 and 1 (main)

Description

This function rescales between 0 and 1

Usage

.scale01(x)

Arguments

x  numeric vector to be re-scaled

Value

This function returns a numeric vector of same length with re-scaled values

See Also

scaleXY, scale

Examples

.scale01(11:15)

.scaleSpecGrp  Rescale respective to specific group

Description

This function allows to rescale data 'x' so that 2 specific groups get normalized to predefined values (and all other values follow proportionally) 'grp1Num' and 'grp2Num' should be either numeric for positions in 'x' or character for names of 'x'; if 'grp1Num' and/or 'grp2Num' design multiple locations: perform median or mean summarization, according to 'sumMeth'

Usage

.scaleSpecGrp(
  x,
  grp1Num,
  grp1Val,
  grp2Num = NULL,
  grp2Val = NULL,
  sumMeth = "mean",
  callFrom = NULL
)
.scaleXY

Arguments
- **x** (numeric vector) main input
- **grp1Num** (numeric)
- **grp1Val** (numeric)
- **grp2Num** (numeric)
- **grp2Val** (numeric)
- **sumMeth** (character) method for summarizing
- **callFrom** (character) allow easier tracking of messages produced

Value
- numeric vector

See Also
- which.min

Examples
- `.firstMin(c(4,3:6))`

---

**Scale between min and max value (main)**

Description
This function rescales between user-defined min and max values

Usage
- `.scaleXY(x, minim = 2, maxim = 3)`

Arguments
- **x** numeric vector to be re-scaled
- **minim** (numeric) minimum value for resultant vector
- **maxim** (numeric) minimum value for resultant vector

Value
This function returns a matrix of CV values

See Also
- scaleXY, scale
**Examples**

```r
c.scaleXY(11:15, min=1, max=100)
```

---

**Description**

This function is depreciated, please use `cutStr` instead! This function allows truncating character vector to all variants from given start, with min and optional max length. Used to evaluate argument calls without giving full length of argument.

**Usage**

```r
.seqCutStr(txt, startFr = 1, minLe = 1, reverse = TRUE)
```

**Arguments**

- `txt` (character) main input, may be length >1
- `startFr` (integer) where to start
- `minLe` (integer) minimum length of output
- `reverse` (logical) return longest text-fragments at beginning of vector

**Value**

This function returns a character vector

**See Also**

`pasteC`; `substr`

**Examples**

```r
.seqCutStr("abcdefg", minLe=2)
```
.setLowestTo

*Set lowest value to given value*

**Description**

This function aims to set lowest value of `x` to value 'setTo'.

**Usage**

```
.setLowestTo(x, setTo)
```

**Arguments**

- `x` (numeric): main vector to be treated
- `setTo` (numeric): replacement value

**Value**

This function returns a numeric vector.

**See Also**

`nchar`

**Examples**

```
.setLowestTo(9:4, 6)
```

.sortMid

*Choose most frequent or middle of sorted vector*

**Description**

This function chooses the (first) most frequent or middle of sorted vector.

**Usage**

```
.sortMid(x, retVal = TRUE)
```

**Arguments**

- `x` (numeric): main input
- `retVal` (logical): return value of most frequent, if FALSE return index of (1st) 'x' for most frequent

**Examples**

```
.sortMid(9:4, TRUE)
```
Value

This function returns a numeric vector

See Also

Simple/partial functionality in `summarizeCols, checkSimValueInSer`

Examples

```r
.sortMid(11:14)
```

---

**Description**

This function aims to reorganize an array by reducing dimension `byDim` (similar to `stack()` for data-frames) It returns an array/matrix of 1 dimension less than `arr`, 1st dim has more lines (names as paste with `_`)

**Usage**

```r
.stackArray(arr, byDim = 3)
```

**Arguments**

<table>
<thead>
<tr>
<th>arg</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr</td>
<td>(array) main input</td>
</tr>
<tr>
<td>byDim</td>
<td>(integer)</td>
</tr>
</tbody>
</table>

**Value**

This function returns an array/matrix of 1 dimension less than `arr`, 1st dim has more lines (names as paste with `_`)

**See Also**

`dist`

**Examples**

```r
(arr1 <- array(11:37, dim=c(3,3,3)))
.stackArray(arr1, 3)
```
.summarizeCols

Summarize columns of matrix (or data.frame) 'x' using apply (main)

Description

This function summarizes columns of matrix (or data.frame) 'x' using apply. Note, it cannot handle character entries! (all results will be NA)

Usage

.summarizeCols(
  x,
  me = c("median", "medianComplete", "mean", "meanComplete", "aver", "average", "min", "max", "maxOfRef", "minOfRef", "maxAbsOfRef", "lastLi", "last", "firstComplete", "first", "firstLi", "summary"),
  vectAs1row = TRUE
)

Arguments

  x          data.frame matrix of data to be summarized by column
  me         (character, length=1) summarization method (eg 'maxLast', 'minLast', 'maxLast', 'maxAbsLast', 'minLast', 'medianComplete' or 'meanComplete')
  vectAs1row (logical) if TRUE will interpret non-matrix 'x' as matrix with 1 row (correct effect of automatic conversion when extracting 1 line)

Value

  vector with summary for each column

See Also

  summarizeCols

Examples

t1 <- matrix(round(runif(30,1,9)), nc=3); rownames(t1) <- letters[c(1:5,3:4,6:4)]
.trimFromEnd

Trim from end

Description

This function allows trimming/removing redundant text-fragments from end

Usage

.trimFromEnd(x, ..., callFrom = NULL, debug = FALSE, silent = TRUE)

Arguments

x character vector to be treated
... more vectors to be treated
callFrom (character) allow easier tracking of messages produced
debug (logical) display additional messages for debugging
silent (logical) suppress messages

Value

This function returns a modified character vector

See Also

.trimRedundText; Inverse : Find/keep common text keepCommonText; you may also look for related functions in package stringr

Examples

txt1 <- c("abcd_ccc","bcd_ccc","cde_ccc")
.trimFromEnd(txt1)

.trimFromStart

Trim from start

Description

This function allows trimming/removing redundant text-fragments from start
Usage

.trimFromStart(
  x,
  ..., 
  minNchar = 1,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

x character vector to be treated
... more vectors to be treated
minNchar (integer) minimum number of characters that must remain
silent (logical) suppress messages
debug (logical) display additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a modified character vector

See Also

.trimRedundText; Inverse: Find/keep common text keepCommonText; you may also look for related functions in package stringr

Examples

txt1 <- c("abcd_ccc","bcd_ccc","cde_ccc")
.trimFromStart(txt1)

.trimLeft

Trim from Left

Description

This function allows trimming/removing redundant text-fragments from left side

Usage

.trimLeft(x, minNchar = 1, silent = TRUE, debug = FALSE, callFrom = NULL)
Arguments

- `x`: character vector to be treated
- `minNchar`: (integer) minimum number of characters that must remain
- `silent`: (logical) suppress messages
- `debug`: (logical) display additional messages for debugging
- `callFrom`: (character) allow easier tracking of messages produced

Value

This function returns a modified character vector

See Also

- `trimRedundText`: Inverse: Find/keep common text
- `keepCommonText`: you may also look for related functions in package `stringr`

Examples

```r
txt1 <- c("abcd_ccc","bcd_ccc","cde_ccc")
.trimLeft(txt1)
```

Describe

This function allows trimming/removing redundant text-fragments from right side

Usage

```r
.trimRight(x, minNchar = 1, silent = TRUE, debug = FALSE, callFrom = NULL)
```

Arguments

- `x`: character vector to be treated
- `minNchar`: (integer) minimum number of characters that must remain
- `silent`: (logical) suppress messages
- `debug`: (logical) display additional messages for debugging
- `callFrom`: (character) allow easier tracking of messages produced

Value

This function returns a modified character vector
See Also

trimRedundText: Inverse: Find/keep common text keepCommonText: you may also look for related functions in package stringr

Examples

txt1 <- c("abcd_ccc","bcd_ccc","cde_ccc")
.trimRight(txt1)

Description

This function is an enhanced version of unique, names of elements are maintained

Usage

.uniqueWName(
  x,
  splitSameName = TRUE,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

x (numeric or character vector) main input
splitSameName (logical)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

vector like input

See Also

unique

Examples

aa <- c(a=11, b=12, a=11, d=14, c=11)
.uniqueWName(aa)
.uniqueWName(aa[-1]) # value repeated but different name
\textit{addBeforFileExtension} \hspace{1cm} \textit{Add text before file-extension}

\subsection*{Description}

This function helps changing character strings like file-names and allows adding the character vector 'add' (length 1) before the extension (defined by last '.') of the input string 'x'. Used for easily creating variants/additional filenames but keeping current extension.

\subsection*{Usage}

\begin{verbatim}
addBeforFileExtension(x, add, sep = '_', silent = FALSE, callFrom = NULL, debug = FALSE)
\end{verbatim}

adjBy2ptReg

Arguments

x main character vector
add character vector to be added
sep (character) separator between 'x' & 'add' (character, length 1)
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced
debug (logical) additional messages for debugging

Value
modified character vector

Examples

addBeforeFileExtension(c("abd.txt","ghg.ijij.txt","kjh"),"new")

adjBy2ptReg Linear rescaling of numeric vector or matrix

Description
adjBy2ptReg takes data within window defined by 'lims' and determines linear transformation so that these points get the regression characteristics 'regrTo', all other points (ie beyond the limits) will follow the same transformation. In other words, this function performs 'linear rescaling', by adjusting (normalizing) the vector 'dat' by linear regression so that points falling in 'lims' (list with upper & lower boundaries) will end up as 'regrTo'.

Usage

adjBy2ptReg(
  dat,
  lims,
  regrTo = c(0.1, 0.9),
  refLines = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments
dat numeric vector, matrix or data.frame
lims (list, length=2) should be list giving limits (list(lo=c(min,max),hi=c(min,max))) in data allowing identifying which points will be used for determining slope & offset
appendNR

(regrTo (numeric, length=2) to which characteristics data should be regressed
refLines (NULL or integer) optional subselection of lines of dat (will be used internal as refDat)
silent (logical) suppress messages
debug (logical) display additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value
This function returns a matrix (of same dimensions as input matrix) with normalized values

See Also
normalizeThis

Examples
set.seed(2016); dat1 <- round(runif(50,0,100),1)
## extreme values will be further away :
adjBy2ptReg(dat1, lims=list(c(5,9), c(60,90)))
plot(dat1, adjBy2ptReg(dat1, lims=list(c(5,9), c(60,90))))

appendNR
Append vectors or lists, without duplicating common elements

Description
This function allows combining two vectors or lists without duplicating common content (defined by name of list-elements).

Usage
appendNR(x, y, rmDuplicate = TRUE, silent = FALSE, callFrom = NULL)

Arguments
x (vector or list) must have names to allow checking for duplicate names in y
y (vector or list) must have names to allow checking for duplicate names in x
rmDuplicate (logical) avoid duplicating list-elements present in both x and y (based on names of list-elements)
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Details
When setting the argument rmDuplicate=FALSE the function will behave like append.
Value

If both \( x \) and \( y \) are vectors, the output will be a vector, otherwise it will be a list.

See Also

append; lrbind

Examples

```r
l1 <- list(a=1, b=2, c=3)
l2 <- list(A=11, B=12, C=3)
appendNR(l1, l2)
append(l1, l2)
```

arrayCV

\[ \text{CV of array} \]

Description

arrayCV gets CVs for replicates in 2 or 3 dim array and returns CVs as matrix. This function may be used to calculate CVs from replicate microtiter plates (eg 8x12) where replicates are typically done as multiple plates, ie initial matrixes that are the organized into arrays.

Usage

```r
arrayCV(arr, byDim = 3, silent = TRUE, callFrom = NULL)
```

Arguments

- `arr` (3-dim) array of numeric data like where replicates are along one dimension of the array
- `byDim` (integer) over which dimension replicates are found
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of message produced

Value

matrix of CV values

See Also

rowCVs, rowGrpCV, replPlateCV

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
head(arrayCV(dat1,byDim=2))
```
asSepList allows reorganizing most types of input into a list with separate numeric vectors. For example, matrixes or data.frames will be split into separate columns (different to partUnlist which maintains the original structure). This function also works with lists of lists. This function may be helpful for reorganizing data for plots.

**Usage**

```r
asSepList(
  y,
  minLen = 4,
  asNumeric = TRUE,
  exclElem = NULL,
  sep = "_",
  fillNames = TRUE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

**Arguments**

- `y` list to be separated/split in vectors
- `minLen` (integer) min length (or number of rows), as add’l element to eliminate arguments given without names when asSepList is called in vioplot2
- `asNumeric` (logical) to transform all list-elements in simple numeric vectors (won’t work if some entries are character)
- `exclElem` (character) optimal names to exclude if any (lazy matching) matches (to exclude other arguments be misinterpreted as data)
- `sep` (character) separator when combining name of list-element to columns
- `fillNames` (logical) add names for list-elements/series when not given
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of messages produced
- `debug` (logical) display additional messages for debugging

**Value**

This function returns a list, partially unlisted to vectors

**See Also**

partUnlist, unlist
Examples

bb <- list(fa=gl(2,2), c=31:33, L2=matrix(21:28,nc=2),
1i=list(li1=11:14, li2=data.frame(41:44)))
asSepList(bb)
## multi data-frame examples
cia <- data.frame(a=11:15, b=21:25, c=31:35)
cb <- data.frame(a=51:53, b=61:63)
cc <- list(gl(3,2), ca, cb, 91:94, short=81:82, letters[1:5])
asSepList(cc)
cd <- list(e1=gl(3,2), e2=ca, e3=cb, e4=91:94, short=81:82, e6=letters[1:5])
asSepList(cd)

Description

It is assumed that multiple fragments from a common ancestor may be characterized by their start- and end-sites by integer values. For example, if 'abcdefg' is the ancestor, the fragments 'bcd' (from position 2 to 4) to and 'efg' may then be assembled. To do so, all fragments must be presented as matrix specifying all start- and end-sites (and fragment-names). buildTree searches contiguous fragments from columns 'posCo' (start/end) from 'disDat' to build tree & extract path information starting with line 'startFr'. Made for telling if dissociated fragments contribute to long assemblies. This function uses various functions of package data.tree which must be installed, too.

Usage

buildTree(
  disDat,
  startFr = NULL,
  posCo = c("beg", "end"),
  silent = FALSE,
  callFrom = NULL
)

Arguments

disDat (matrix or data.frame) integer values with 1st column, ie start site of fragment, 2nd column as end of fragments, rownames as unique IDs (node-names)
startFr (integer) index for 1st node (typically =1 if 'disDat' sorted by "beg"), should point to a terminal node for consecutive growing of branches
posCo (character) colnames specifying the begin & start sites in 'disDat', if NULL 1st & 2nd col will be used
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced
Value

This function returns a list with $paths (branches as matrix with columns 'sumLen' & 'n'), $usedNodes (character vector of all names used to build tree) and $tree (object from data.tree)

See Also

package data.tree original function used Node; in this package : for exploiting edge/tree related issues simpleFragFig, countSameStartEnd and contribToContigPerFrag.

Examples

frag2 <- cbind(beg=c(2,3,7,13,15,7,9,7,3,7,5,7,3),end=c(6,12,8,18,20,20,19,12,12,4,12,7,12,4))
rownames(frag2) <- c("A","E","B","C","D","F","H","G","I", "J","K","L","M","N")
builtTree(frag2)
countSameStartEnd(frag2)

cbindNR

cbind to non-redundant

Description

cbindNR combines all matrixes given as arguments to non-redundant column names (by ADDING the number of 'duplicated' columns !). Thus, this function works similar to cbind, but allows combining multiple matrix-objects containing redundant column-names. Of course, all input-matrixes must have the same number of rows ! By default, the output gets sorted by column-names. Note, due to the use of '...' arguments must be given by their full argument-names, lazy evaluation might not recognize properly argument names.

Usage

cbindNR(
    ..., convertDFtoMatr = TRUE, sortOutput = TRUE, summarizeAs = "sum", silent = FALSE, callFrom = NULL
)

Arguments

... all matrixes to get combined in cbind way
convertDFtoMatr (logical) decide if output should be converted to matrix
sortOutput (logical) optional sorting by column-names
summarizeAs (character) decide of combined values should get summed (default, 'sum') or averaged ('mean')
checkAvSd

silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced

Value

This function returns a matrix or data.frame (as cbind would return)

See Also

cbind, nonAmbiguousNum, firstOfRepLines

Examples

ma1 <- matrix(1:6, ncol=3, dimnames=list(1:2,LETTERS[3:1]))
ma2 <- matrix(11:16, ncol=3, dimnames=list(1:2,LETTERS[3:5]))
cbindNR(ma1, ma2)
cbindNR(ma1, ma2, summarizeAs="mean")

checkAvSd Check how multiple groups of data separate or overlap based on mean +/- sd

Description

checkAvSd compares if/how neighbour groups separate/overlap via the 'engineering approach' (+/- 2 standard-deviations is similar to a=0.05 t.test). This approach may be used as less elegant alternative to (multi-group) logistic regression. The function uses 'daAv' as matrix of means (rows are tested for up/down character/progression) which get compared with boundaries taken from daSd (for Sd values of each mean in 'daAv').

Usage

checkAvSd(
  daAv,
  daSd,
  nByGr = NULL,
  multSd = 2,
  codeConst = "const",
  extSearch = FALSE,
  outAsLogical = TRUE,
  silent = FALSE,
  callFrom = NULL
)
checkGrpOrder

Arguments

- **daAv**: matrix or data.frame
- **daSd**: matrix or data.frame
- **nByGr**: optional specifying number of Elements per group, allows rather using SEM (adopt to variable n of different groups)
- **multSd**: (numeric) the factor specifying how many sd values should be used as margin
- **codeConst**: (character) which term/word to use when specifying 'constant'
- **extSearch**: (logical) if TRUE, extend search to one group further (will call result 'nearUp' or 'nearDw')
- **outAsLogical**: to switch between 2col-output (separate col for 'up' and 'down') or simple categorical vector ('const','okDw','okUp')
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of message(s) produced

Value

vector describing character as 'const' or 'okUp','okDw' (or if extSearch=TRUE 'nearUp','nearDw')

See Also

- rowGrpMeans

Examples

```r
mat1 <- matrix(rep(11:24,3)[1:40],byrow=TRUE,ncol=8)
checkGrpOrderSEM(mat1,grp=gl(3,3)[-1])
checkAvSd(rowGrpMeans(mat1,gl(3,3)[-1]),rowGrpSds(mat1,gl(3,3)[-1]) )
# consider variable n :
checkAvSd(rowGrpMeans(mat1,gl(3,3)[-1]),rowGrpSds(mat1,gl(3,3)[-1]),nByGr=c(2,3,3))
```

Description

checkGrpOrder tests each line of 'x' if expected order appears. Used for comparing groups of measures with expected profile (simply by matching expected order)

Usage

```r
checkGrpOrder(
  x,
  rankExp = NULL,
  revRank = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

```
checkGrpOrderSEM

Arguments

- **x**: matrix or data.frame
- **rankExp**: (numeric) expected order for values in columns, default 'rankExp' =1:ncol(x)
- **revRank**: (logical) if 'revRank'=TRUE, the initial ranks & reversed ranks will be tested
- **silent**: (logical) suppress messages
- **debug**: (logical) display additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Value

vector of logical values

See Also

- checkGrpOrderSEM

Examples

set.seed(2005); mat1 <- rbind(matrix(round(runif(40),1),nc=4), rep(1,4))
checkGrpOrder(mat1)
checkGrpOrder(mat1,c(1,4,3,2))

---

checkGrpOrderSEM Check order of multiple groups including non-overlapping SEM-margins

Description

checkGrpOrderSEM tests each line of 'x' if expected order of (replicate-) groups (defined in 'grp') appears intact, while including SEM of groups (replicates) via a proportional weight 'sdFact' as (avGr1-gr1SEM) < (avGr1+gr1SEM) < (avGr2-gr2SEM) < (avGr2+gr2SEM). Used for comparing groups of measures with expected profile (by matching expected order) to check if data in 'x' representing groups ('grp') as lines follow. Groups of size=1: The sd (and SEM) can’t be estimated directly without any replicates, however, an estimate can be given by shrinking if 'shrink1sampSd'=TRUE under the hypothesis that the overall mechanisms determining the variances is constant across all samples.

Usage

checkGrpOrderSEM(  
  x,  
  grp,  
  sdFact = 1,  
  revRank = TRUE,  
  shrink1sampSd = TRUE,  
  silent = FALSE,  
  callFrom = NULL  
)
**Arguments**

- `x`: matrix or data.frame
- `grp`: (factor) to organize replicate columns of `x`
- `sdFact`: (numeric) is proportional factor how many units of SEM will be used for defining lower & upper bounds of each group
- `revRank`: (logical) optionally revert ranks
- `shrink1sampSd`: (logical)
- `silent`: (logical) suppress messages
- `callFrom`: (character) allow easier tracking of message(s) produced

**Value**

logical vector if order correct (as expected based on ranks)

**See Also**

takes only 10

**Examples**

```r
mat1 <- matrix(rep(11:24,3)[1:40],byrow=TRUE,ncol=8)
checkGrpOrderSEM(mat1,grp=gl(3,3)[-1])
```

---

**Description**

checkSimValueInSer checks all values of `x` for similar values outside/within (relative) range of `ppm` (i.e., ambiguous within given range). Return logical vector: FALSE for each entry of `x` if value inside of ppm range to neighbour.

**Usage**

```r
checkSimValueInSer(x, ppm = 5, sortX = TRUE)
```

**Arguments**

- `x`: numeric vector
- `ppm`: (numeric) ppm-range for considering as similar
- `sortX`: (logical) allows speeding up function when set to FALSE, for large data that are already sorted

**Value**

logical vector: FALSE for each entry of `x` if value inside of ppm range to neighbour
checkStrictOrder

See Also

similar with more options withinRefRange

Examples

```r
va1 <- c(4:7,7,7,7,7,7,8:10)+(1:11)/28600; checkSimValueInSer(va1)
cbind(va=va1,simil=checkSimValueInSer(va1))
```

---

checkStrictOrder  
*Check for strict (ascending or descending) order*

Description

checkStrictOrder tests lines of 'dat' (matrix of data.frame) for strict order (ascending, descending or constant), each col of data is tested relative to the col on its left.

Usage

```r
checkStrictOrder(
  dat,  
invertCount = FALSE,  
silent = FALSE,  
debug = FALSE,  
callFrom = NULL
)
```

Arguments

- **dat**: matrix or data.frame
- **invertCount**: (logical) inverse counting (ie return 0 for all elements in order)
- **silent**: (logical) suppress messages
- **debug**: (logical) display additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Value

matrix with counts of up pairs, down pairs, equal-pairs, if 'invertCount'=TRUE all non-events are counted, ie a resulting 0 means that all columns are following the described characteristics (with variable col-numbers easier to count)

See Also

order, checkGrpOrder

Examples

```r
set.seed(2005); mat1 <- rbind(matrix(round(runif(40),1),nc=4), rep(1,4))
checkStrictOrder(mat1); mat1[which(checkStrictOrder(mat1)[,2]==0),]
```
checkVectLength checks argument 'x' for expected length 'expeL' and return either message or error when expectation not met. May be used for parameter ('sanity') checking in other user front-end functions.

Usage

checkVectLength(
  x,
  expeL = 1,
  stopOnProblem = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

  x     (numeric or character vector) input to check length
  expeL (numeric) expected length
  stopOnProblem (logical) continue on problems with message or stop (as error message)
  silent (logical) suppress messages
  debug (logical) display additional messages for debugging
  callFrom (character) allow easier tracking of message(s) produced

Value

This function returns NULL; it produces either error-message if length is not OK or optional message if length is OK

Examples

aa <- 1:5; checkVectLength(aa,exp=3)
cleanReplicates  

Replace Most Distant Values by NA

Description

This procedures aims to streighten (clean) the most extreme values of noisy replicates by identifying the most distant points (among a set of replicates). The input 'x' (matrix or data.frame) is supposed to come from multiple different measures taken in replicates (eg weight of different individuals as rows taken as multiple replicate measures in subsequent columns).

Usage

```r
cleanReplicates(
  x,
  centrMeth = "median",
  nOutl = 2,
  retOffPos = FALSE,
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

- **x**: matrix (or data.frame)
- **centrMeth**: (character) method to summarize (mean or median)
- **nOutl**: (integer) determines how many points per line will be set to NA (with n=1 the worst row of replicates will be 'cleaned')
- **retOffPos**: (logical) if TRUE, replace the most extreme outlyer only
- **silent**: (logical) suppres messages
- **callFrom**: (character) allow easier tracking of messages produced

Details

With the argument nOutl the user chooses the total number of most extreme values to replace by NA. how many of the most extreme replicates of the whole dataset will replaced by NA, ie with nOutl=1 only the single most extreme outlyer will be replaced by NA. Outlier points are determined as point(s) with highest distance to (row) center (median and mean choice via argument 'centrMeth'). Thus function returns input data with "removed" points set to NA, or if retOffPos=TRUE the most extreme/outlier positions.

Value

This function returns a matrix of same dimensions as input x, data-points which were tagged/removed are set to NA, or if retOffPos=TRUE the most extreme/outlier positions.
closeMatchMatrix

Reorganizes results from search for close (similar) values in matrix-view

Description

closeMatchMatrix reorganizes/refines results from simple search of similar values of 2 sets of data by findCloseMatch (as list for one-to-many relations) to more human friendly/readable matrix. This function returns results combining two sets of data which were initially compared (eg measured and theoretical values) as matrix-view using output of findCloseMatch and both original datastes Additional information (covariables, annotation, ...) may be included as optional columns for either 'predMatr' or 'measMatr'. Note: It is important to run findCloseMatch with sortMatch=FALSE! Note: Results presented based on view of 'predMatr', so if multiple 'measMatr' are at within tolared distance, lines of 'measMatr' will be repeated; Note: Distances 'disToMeas' and 'ppmToPred' are oriented: neg value if measured is lower than predicted (and pos values if higher than predicted); Note: Returns NULL when nothing within given limits of comparison;

Usage

closeMatchMatrix(
  closeMatch,
  predMatr,
  measMatr,
  prefMatch = c("^x", "^y"),
  colPred = 1,
  colMeas = 1,
  limitToBest = TRUE,
  asDataFrame = FALSE,
  origNa = TRUE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)

Arguments

closeMatch (list) output from findCloseMatch, is list with hits for each 'x' (1st argument) : named vectors of value & x index in name; run with sortMatch=FALSE

predMatr (vector or matrix) predicted values, the column 'colPred' indicates which column is used for matching from findCloseMatch; if column 'id' present this column will be used as identifier for matching

measMatr (vector or matrix) measured values, the column 'colMeas' indicates which column is used for matching from findCloseMatch; if column 'id' present this column will be used as identifier for matching

Examples

mat3 <- matrix(c(19,20,30, 18,19,28, 16,14,35),ncol=3)
cleanReplicates(mat3, nOutl=1)
closeMatchMatrix

- **prefMatch**: (character, length=2) prefixes ('^x' and/or '^y') that may have been added by `findCloseMatch`
- **colPred**: (integer or text, length=1) column of 'predMatr' with main values of comparison
- **colMeas**: (integer or text, length=1) column of 'measMatr' with main measures of comparison
- **limitToBest**: (integer) column of 'measMatr' with main measures of comparison
- **asDataFrame**: (logical) convert results to data.frame if non-numeric matrix produced (may slightly slow down big results)
- **origNa**: (logical) will try to use original names of objects 'predMatr','measMatr', if they are not multi-column and not conflicting other output-names (otherwise 'predMatr','measMatr' will appear)
- **silent**: (logical) suppress messages
- **callFrom**: (character) allows easier tracking of message(s) produced
- **debug**: (logical) for bug-tracking: more/enhanced messages

**Value**

results as matrix-view based on initial results from `findCloseMatch`, including optional columns of suppelemental data for both sets of data for comparison. Returns NULL when nothing within limits

**See Also**

- `findCloseMatch`
- `checkSimValueInSer`

**Examples**

```r
aA <- c(11:17); bB <- c(12.001,13.999); cC <- c(16.2,8,9,12.5,15.9,13.5,15.7,14.1,5)
(cloMa <- findCloseMatch(aA,cC,com="diff",lim=0.5,sor=FALSE)) # all matches (of 2d arg) to/within limit for each of 1st arg ('x'); 'y' ..to 2nd arg = cC
(maAa <- closeMatchMatrix(cloMa,aA,cC,lim=TRUE)) #
(maAa <- closeMatchMatrix(cloMa,aA,cC,lim=FALSE,origN=TRUE)) #
(maAa <- closeMatchMatrix(cloMa,cbind(valA=81:87,aA),cbind(valC=91:99,cC),colM=2,colP=2,lim=FALSE))
(maAa <- closeMatchMatrix(cloMa,cbind(valA=81:87,aA),cbind(valC=91:99,cC),colM=2,colP=2,lim=FALSE,asData=TRUE))
(maAa <- closeMatchMatrix(cloMa,cbind(id=names(aA),valA=81:87,aA),cbind(id=names(cC),valC=91:99,cC),colM=3,colP=3,lim=FALSE,deb=FALSE))
```

```r
a2 <- aA; names(a2) <- letters[1:length(a2)]; c2 <- cC; names(c2) <- letters[10+1:length(c2)]
(cloM2 <- findCloseMatch(x=a2,y=c2,com="diff",lim=0.5,sor=FALSE)) #
a2 <- aA; names(a2) <- letters[1:length(a2)]; c2 <- cC; names(c2) <- letters[10+1:length(c2)]
(cloM2 <- findCloseMatch(x=a2,y=c2,com="diff",lim=0.5,sor=FALSE)) #
(maA2 <- closeMatchMatrix(cloM2,cbind(valA=81:87,a2),measM=cbind(valC=91:99,c2),
    colM=2,colP=2,lim=FALSE,asData=TRUE))
(maA2 <- closeMatchMatrix(cloM2,cbind(id=names(a2),valA=81:87,a2),cbind(id=names(c2),
    valC=91:99,c2),colM=3,colP=3,lim=FALSE,deb=FALSE))
```
coinPermTest

Compare means of two vectors by permutation test

Description

Run coin-flipping like permutation tests (to compare difference of 2 means: 'x1' and 'x2') without any distribution-assumptions. This function uses the package coin, if not installed, the function will return NULL and give a warning.

Usage

coinPermTest(
  x1,
  x2,
  orient = "two.sided",
  nPerm = 5000,
  silent = FALSE,
  callFrom = NULL
)

Arguments

x1 numeric vector (to be compared with vector 'x2')
x2 numeric vector (to be compared with vector 'x1')
orient (character) may be "two.sided", "greater" or "less"
nPerm (integer) number of permutations
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced

Value

This function returns an object of "MCp" class numeric output with p-values

See Also

oneway_test in LocationTests

Examples

coinPermTest(2, 3, nPerm=200)
**colMedSds**

*Standard error of median for each column by bootstrap*

**Description**

Determine standard error (sd) of median by bootstrapping for multiple sets of data (rows in input matrix 'dat'). Note: The package `boot` must be installed from CRAN.

**Usage**

`colMedSds(dat, nBoot = 99, silent = FALSE, callFrom = NULL)`

**Arguments**

- `dat` (numeric) matix
- `nBoot` (integer) number if iterations
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of messages produced

**Value**

This function returns a (numeric) vector with estimated standard errors

**See Also**

`boot`

**Examples**

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10,20)), ncol=10)
colMedSds(dat1)
```

---

**colorAccording2**

*Transform numeric values to color-gradient*

**Description**

This function helps making color-gradients for plotting a numerical variable. It requires the package 'RColorBrewer' being installed from CRAN. Note: RColorBrewer palettes were not integrated here, since they are not continuous.
Usage

colorAccording2(
  x,
  gradTy = "rainbow",
  nStartOmit = NULL,
  nEndOmit = NULL,
  revCol = FALSE,
  alpha = 1,
  callFrom = NULL
)

Arguments

x (character) color input
gradTy (character) type of gradeint may be 'rainbow', 'heat.colors', 'terrain.colors', 'topo.colors', 'cm.colors', 'hcl.colors', 'grey.colors', 'gray.colorsW' or 'logGray'
nStartOmit (integer) omit n steps from begining of gradient range
nEndOmit (integer or "sep") omit n steps from end of gradient range, if nEndOmit="sep" 20 percent of initial grades will be removed to obtain 'separate' ie non-closing color-circles/gradients eg with rainbow
revCol (logical) reverse order
alpha (numeric) optional transparency value (1 for no transparency, 0 for complete opaqueness)
callFrom (character) allow easier tracking of message(s) produced

Value

This function returns a character vector (of same length as x) with color encoding

See Also

cut

Examples

set.seed(2015); dat1 <- round(runif(15),2)
plot(1:15,dat1,pch=16,cex=2,col=colorAccording2(dat1))
plot(1:15,dat1,pch=16,cex=2,col=colorAccording2(dat1,nStartO=0,nEndO=4,revCol=TRUE))
plot(1:9,pch=3)
points(1:9,1:9,col=transpGraySca(st=0,en=0.8,nSt=9,trans=0.3),cex=42,pch=16)
### colSds

**Description**

colSds is a speed optimized sd for matrix or data.frames. It and treats each line as an independent set of data for calculating the sd (equiv to apply(dat, 1, sd)). NAs are ignored from data.

**Usage**

colSds(dat)

**Arguments**

dat  
matrix (or data.frame) with numeric values (may contain NAs)

**Value**

numeric vector of sd values

**See Also**

sd

**Examples**

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10, 20)), nc=10)
colSds(dat1)
```

---

### combinatIntTable

**Description**

Provide all combinations for each of n elements of vector 'nMax' (positive integer, eg number of max multiplicative value). For example, imagine, we have 3 cities and the (maximum) voting participants per city. Results must be read vertically and allow to see all total possible compositons.

**Usage**

```r
combinatIntTable(
  nMax,
  include0 = TRUE,
  asList = FALSE,
  callFrom = NULL,
  silent = TRUE
)
```
Arguments

nMax (positive integer) could be max number of voting participants form different cities, eg Paris max 2 persons, Lyon max 1 person ...
include0 (logical) include 0 occurrences, ie provide all combinations starting from 0 or from 1 up to nMax
asList (logical) return result as list or as array
callFrom (character) allow easier tracking of messages produced
silent (logical) suppress messages

Value

list or array (as 2- or 3 dim) with possible number of occurrences for each of the 3 elements in nMax. Read results vertical: out[[1]] or out[,1] .. (multiplicative) table for 1st element of nMax; out[,2] .. for 2nd

See Also

combn

Examples

combinatIntTable(c(1,1,1,2), include0=TRUE, asList=FALSE, silent=TRUE)
## Imagine we have 3 cities and the (maximum) voting participants per city:
nMa <- c(Paris=2, Lyon=1, Strasbourg=1)
combinatIntTable(nMa, include0=TRUE, asList=TRUE, silent=TRUE)

combineAsN (Combine Vectors From List And Return Basic Count Statistics)

Description

The aim of this function is to choose a fixed number (nCombin) of list-elements from lst and count the number of common values/words. Furthermore, one can define levels to fine-tune the types of combinations to examine. In case multiple combinations for a given level are possible, some basic summary statistics are provided, too.

Usage

combineAsN(
  lst,
  lev = NULL,
  nCombin = 3,
  remDouble = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst</td>
<td>(list of character or integer vectors) main input</td>
</tr>
<tr>
<td>lev</td>
<td>(character) define groups of lst</td>
</tr>
<tr>
<td>nCombin</td>
<td>(integer) number of list-elements to combine from lst</td>
</tr>
<tr>
<td>remDouble</td>
<td>(logical) remove intra-duplicates (defaults to TRUE)</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>debug</td>
<td>(logical) additional messages for debugging</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allow easier tracking of messages produced</td>
</tr>
</tbody>
</table>

Details

Note of caution: With very long lists and/or high numbers of repeats of given levels, however, the computational effort increases very much (like it does when using `table`). Thus, when exploring all different combinations of large data-sets may easily result in queries consuming many resources (RAM and processing time)! It is recommended to start testing with test smaller sub-groups.

The main idea of this function is to count frequency of terms when combining different drawings. For example, you ask students from different cities which are their preferred hobbies, they may have different preference depending on the city (defined by `lev`). Now, if you want to make groups of 3 students, possibly with one from each city (A, B, and C), you want to count/estimate the frequency of different combinations possible. Thus, using this function all combinations of the students from city A with the students from city B and C will be made when counting the number of common hobbies (by `nCombin` students). Then, all counting results will be summarized to the average count for the various categories (which hobbies were seen once, twice, or three times...), sem (standard error of the mean) and CI (95% confidence interval).

Of course, the number of potential combinations may quickly get very large. Using the argument `remDouble=TRUE` you can limit the search to either finding all students giving the same answer plus all student giving different answers. In this case, when a given level appears multiple times, all possible combinations using one of the respective entries will be made with the other levels.

Value

This function returns an array with 3 dimensions:

- i) the combinations of `nCombin` list-elements
- ii) the number of counts (n), sem (standard error of the mean), CI (confidence interval) and sd

See Also

`table, replicateStructure`

Examples

```r
## all list-elements are considered equal
tm1 <- list(a1=LETTERS[1:17], a2=LETTERS[3:19], a3=LETTERS[6:20], a4=LETTERS[8:22])
combineAsN(tm1, lev=gl(1,4))[,1,]

## different levels/groups in list-elements
tm4 <- list(a1=LETTERS[1:15], a2=LETTERS[3:16], a3=LETTERS[6:17], a4=LETTERS[8:19],
            b1=LETTERS[5:19], b2=LETTERS[7:20], b3=LETTERS[11:24], b4=LETTERS[13:25], c1=LETTERS[17:26],
```
combineByEitherFactor

Create factor-like column regrouping data regrouping simultaneously by two factors

Description

This function aims to address the situation when two somehow different groupings (of the same data) exist and need to be joined. It is not necessary that both alternative groupings use the same labels, neither. combineByEitherFactor adds new (last) column named 'grp' to input matrix representing the combined factor relative to 2 specified columns from input matrix mat (via 'refC1','refC2'). Optionally, the output may be sorted and a column giving n per factor-level may be added. The function treats selected columns of mat as pairwise combination of 2 elements (that may occur multiple times over all lines of mat) and sorts/organizes all instances of such combined elements (ie from both selected columns) as repeats of a given group, who’s class number is given in output column 'grp', the (total) number of repeats may be displayed in column 'nGrp' (nByGrp=TRUE). If groups are overlapping (after re-ordering), an iterative process of max 3x2 passes will be launched after initial matching. Works on numeric as well as character input.

Usage

combineByEitherFactor(
    mat,
    refC1,
    refC2,
    nByGrp = FALSE,
    convergeMax = TRUE,
    callFrom = NULL,
    debug = FALSE,
    silent = FALSE
)

Arguments

mat main input matrix
refC1 (integer) column-number of 'mat' to use as 1st set
refC2 (integer) column-number of 'mat' to use as 2nd set
nByGrp (logical) add last col with n by group
convergeMax (logical) if TRUE, run 2 add’l iterative steps to search convergence to stable result
callFrom (character) allow easier tracking of message(s) produced
debug (logical) display additional messages for debugging
silent (logical) suppress messages
Value

This function returns a matrix containing both selected columns plus additional column(s) indicating group-number of the pair-wise combination (and optional the total n by group).

Examples

```r
nn <- rep(c("a", "e", "b", "c", "d", "g", "f"), c(3, 1, 2, 2, 1, 2, 1))
qq <- rep(c("m", "n", "p", "o", "q"), c(2, 1, 1, 4, 4))
nq <- cbind(nn, qq)[c(4, 2, 9, 11, 6, 10, 7, 3, 5, 1, 12, 8),]
combineByEitherFactor(nq, 1, 2, nBy=TRUE); combineByEitherFactor(nq, 1, 2, nBy=FALSE)
combineByEitherFactor(nq, 1, 2, conv=FALSE); combineByEitherFactor(nq, 1, 2, conv=TRUE)
##
mm <- rep(c("a", "b", "c", "d", "e"), c(3, 4, 2, 3, 1)); pp <- rep(c("m", "n", "o", "p", "q"), c(2, 2, 2, 2, 5))
combineByEitherFactor(cbind(mm, pp), 1, 2, con=FALSE, nBy=TRUE)
combineByEitherFactor(cbind(mm, pp), 1, 2, con=TRUE, nBy=TRUE)
```

**Description**

Search points in x,y space that are located very close and thus likely to overlap. In case of points close enough, various options for joining names (and shortening longer descriptions) are available.

**Usage**

```r
combineOverlapInfo(
  dat,
  suplInfo = NULL,
  disThr = 0.01,
  addNsimil = TRUE,
  txtSepChar = ",",
  combSym = "+",
  maxOverl = 50,
  callFrom = NULL,
  debug = FALSE,
  silent = FALSE
)
```

**Arguments**

- `dat` (matrix) matrix or data.frame with 2 cols (used ONLY 1st & 2nd column !), used as x & y coordinates
- `suplInfo` (NULL or character) when points are considered overlapping the text from 'suplInfo' will be reduced to fragment before 'txtSepChar' and combined (with others from overlapping text) using 'combSym', if NULL $combInf will appear with row-numbers
combineRedBasedOnCol

**Description**

This function works similar to `unique`, but it takes a matrix as input and considers one specified column to find unique instances. It identifies 'repeated' lines of the input-matrix (or data.frame) 'mat' based on (repeated) elements in/of column with name 'colNa' (or column-number). Redundant lines (ie repeated lines) will disappear in output. Eg used with extracted annotation where 1 gene has many lines for different GO annotation.

**Usage**

```r
combineRedBasedOnCol(mat, colNa, sep = "", silent = FALSE, callFrom = NULL)
```
**Arguments**

- **mat**: input matrix or data.frame
- **colNa**: character vector (length 1) matching 1 column name (if mutl only 1st will be used), in case of mult matches only 1st used
- **sep**: (character) separator (default=".")
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of message(s) produced

**Value**

matrix containing the input matrix without lines considered repeated (unique-like)

**See Also**

- `findRepeated`, `firstOfRepLines`, `organizeAsListOfRepl`

**Examples**

```r
matr <- matrix(c(letters[1:6],"h","h","f","e",LETTERS[1:5]),ncol=3, 
dimnames=list(letters[11:15],c("xA","xB","xC")))
combineRedBasedOnCol(matr,colN="xB")
combineRedBasedOnCol(rbind(matr[1,],matr),colN="xB")
```

---

**Description**

Suppose multiple measures (like multiple channels) are taken for subjects and these measures are organized as groups in a list, like multiple parameters (= channels) or types of measurements (typically many parameters are recorded when screening compounds in microtiter plates). Within one parameter/channel all replicate-data from separate list-entries ("lst") will get combined according to names of list-elements. The function will trim any redundant text in names of list-elements, try to isolate separator (may vary among replicate-groups, but should be 1 character long). eg names "hct116 1.1.xlsx" & "hct116 1.2.xlsx" will be combined as replicates, "hct116 2.1.xlsx" will be considered as new group.

**Usage**

```r
combineReplFromListToMatr(lst, callFrom = NULL)
```

**Arguments**

- **lst**: (list) list of arrays (typically multi-parameter measures of micortiterplate data)
- **callFrom**: (character) allows easier tracking of message(s) produced
Value

list of arrays now with same dimension of arrays (but shorter, since replicate-arrays were combined)

See Also

extractChannel, organizeAsListOfReplicates

Examples

```r
lst2 <- list(aa_1x=matrix(1:12,nrow=4,byrow=TRUE),ab_2x=matrix(24:13,nrow=4,byrow=TRUE))
combineReplFromListToMatr(lst2)
```

```
combineSingleT(mat)  Get all combinations with TRUE from each column

Description

This function addresses the case when multiple alternative ways exit to combine two elements. combineSingleT makes combinatory choices: if multiple TRUE in given column of ’mat’ make all multiple selections with always one TRUE from each column. The resultant output contains index for first and second input columns elements to be combined.

Usage

combineSingleT(mat)

Arguments

mat 2-column matrix of logical values

Value

matrix with indexes of combinations of TRUE

Examples

```r
## Example: Fist column indicates which boys want to dance and second column
## which girls want to dance. So if several boys want to dance each of the girls
## will have the chance to dance with each of them.
matr <- matrix(c(TRUE,FALSE,TRUE,FALSE,TRUE,FALSE),ncol=2)
combineSingleT(matr)
```
**completeArrLst**  
*Complete list of arrays for same dimensions*

**Description**

This function aims to inspect repeating structures of data given as list of arrays and will try to complete arrays with fewer lines or columns (as this may appear eg with the very last set of high-throughput screening data if fewer measures remain in the last set). Thus, the dimensions of the arrays are compared and cases with fewer (lost) columns (eg fewer experimental replicates) will be adjust/complete by adding column(s) of NA. Used eg when at reading microtiterplate data the last set is not complete.

**Usage**

```
completeArrLst(arrLst, silent = FALSE, callFrom = NULL)
```

**Arguments**

- `arrLst` (list) list of arrays (typically 1st and 2nd dim for specific genes/objects, 3rd for different measures associated with)
- `silent` (logical) suppress messages
- `callFrom` (character) allows easier tracking of message(s) produced

**Value**

list of arrays, now with same dimension of arrays

**See Also**

`organizeAsListOfRepl`, `extr1chan`

**Examples**

```r
arr1 <- array(1:24,dim=c(4,3,2),dimnames=list(c(LETTERS[1:4]),
paste("col",1:3,sep=""),c("ch1","ch2")))
arr3 <- array(81:96,dim=c(4,2,2),dimnames=list(c(LETTERS[1:4]),
paste("col",1:2,sep=""),c("ch1","ch2")))
arrL3 <- list(pl1=arr1,pl3=arr3)
completeArrLst(arrL3)
```
Description

This little function returns the confidence interval associated to a given significance level \( \alpha \) under the hypothesis of the Normal distribution is valid.

Usage

```r
confInt(x, alpha = 0.05, distrib = "Normal", silent = FALSE, callFrom = NULL)
```

Arguments

- **x** (numeric) main input
- **alpha** (numeric) significance level, accepted type I error
- **distrib** (character) distribution, so far only Normal is implemented
- **silent** (logical) suppress messages
- **callFrom** (character) allow easier tracking of message(s) produced

Value

This function returns the confidence interval to a given \( \alpha \) under the hypothesis of the Normal distribution.

See Also

- TDist
- confint

Examples

```r
confInt(c(5,2:6))
```


**contribToContigPerFrag**

*Characterize individual contribution of single edges in tree-structures*

### Description

This function helps investigating tree-like structures with the aim of indicating how much individual tree components contribute to compose long stretches. `contribToContigPerFrag` characterizes individual (isolated) contribution of single edges in tree-structures. Typically used to process/exploit summarized trees (as matrix) made by `buildTree` which makes use of the package `data.tree`. For example if A,B and C can be joined as well and B +D, this function will check if A+B+C is longer and if A contributes to the longest tree.

### Usage

```r
contribToContigPerFrag(joinMat, fullLength = NULL, nDig = 3)
```

### Arguments

- **joinMat** (matrix) matrix with concatenated edges as rownames (separated by slashes), column `sumLen` for total length and column `n` for number of edges
- **fullLength** (integer) custom total length (useful if the concatenated edges do not cover 100 percent of the original precursor whose fragments are studied)
- **nDig** (integer) rounding: number of digits for 3rd column `len.rat` in output

### Value

Matrix of 3 columns: with length of longest tree-branches where given edge participates (column `sumLen`), the (total) number of edges therein (col `n.frag`) and a relative value (`len.rat`)

### See Also

- to build tree `buildTree`

### Examples

```r
path1 <- matrix(c(17,19,18,17, 4,4,2,3),ncol=2, dimnames=list(c("A/B/C/D","A/B/G/D","A/H","A/H/I"),c("sumLen","n")))
contribToContigPerFrag(path1)
```
conv01toColNa  
*Convert matrix of integer to matrix of x-times repeated column-names*

**Description**

conv01toColNa transforms matrix of integers (eg 0 and 1) to repeated & concatenated text from argument colNa, the character string for 0 occurrences of argument zeroTex may be customized. Used eg when specifying (and concatenating) various counted elements (eg properties) along a vector like variable peptide modifications in proteomics.

**Usage**

```r
conv01toColNa(mat, colNa = NULL, zeroTex = "", pasteCol = FALSE)
```

**Arguments**

- `mat`: input matrix (with integer values)
- `colNa`: alternative (column-)names to the ones from 'mat' (default colnames of 'mat')
- `zeroTex`: text to display if 0 (default '')
- `pasteCol`: (logical) allows to collapse all columns to single chain of characters in output

**Value**

character vector

**Examples**

```r
(ma1 <- matrix(sample(0:3,40,repl=TRUE), ncol=4, dimnames=list(NULL, letters[11:14])))
conv01toColNa(ma1)
conv01toColNa(ma1, colNa=LETTERS[1:4], ze=".")
conv01toColNa(ma1, colNa=LETTERS[1:4], pasteCol=TRUE)
```

convColorToTransp  
*Assign new transparency to given colors*

**Description**

This function allows (re-)defining a new transparency. A color encoding vector will be transformed to the same color(s) but with new transparency (alpha).

**Usage**

```r
convColorToTransp(color, alph = 1)
```
**convMatr2df**

**Convert matrix (eg with redundant) row-names to data.frame**

**Description**

This function provides flexible converting of matrix to data.frame. For example repeated/redundant rownames are not allowed in data.frame(), thus the corresponding column-names have to be renamed using a counter-suffix. In case of non-redundant rownames, a new column 'addIniNa' will be introduced at beginning to document the initial (redundant) rownames, non-redundant rownames will be created. Finally, this functions converts the corrected matrix to data.frame and checks/converts columns for transforming character to numeric if possible. If the input is a data.frame containing factors, they will be converted to character before potential conversion. Note: for simpler version (only text to numeric) see from this package .convertMatrToNum.

**Usage**

```r
convMatr2df(
  mat,
  addIniNa = TRUE,
  duplTxtSep = "_",
  silent = FALSE,
  callFrom = NULL
)
```

**Arguments**

- **color** (character) color input
- **alph** (numeric) transparency value (1 for no transparency, 0 for complete opacity), values <1 will be treated as percent-values

**Value**

character vector (of same length as input) with color encoding for new transparency

**See Also**

`rgb`, `par`

**Examples**

```r
col0 <- c("#998FCC","#5AC3BA","#CBD34E","#FF7D73")
col1 <- convColorToTransp(col0,alph=0.7)
layout(1:2)
pie(rep(1,length(col0)),col=col0)
pie(rep(1,length(col1)),col=col1,main="new transparency")
```
Arguments

- `mat` (matrix (or data.frame) to be converted)
- `addIniNa` (logical) if TRUE an additional column ('ID') with rownames will be added at beginning
- `duplTxtSep` (character) separator for enumerating replicated names
- `silent` (logical) suppres messages
- `callFrom` (character) allow easier tracking of message(s) produced

Value

This functions returns a data.frame equivalent to the input matrix, an additional column named 'ID' will be added for initial rownames

See Also

- `numeric`, for simpler version (only text to numeric) see from this package .convertMatrToNum

Examples

```r
dat1 <- matrix(1:10, ncol=2)
rownames(dat1) <- letters[c(1:3,2,5)]
## as.data.frame(dat1) ... would result in an error
convMatr2df(dat1)

df1 <- data.frame(a=as.character((1:3)/2), b=LETTERS[1:3], c=1:3)
str(convMatr2df(df1))

df2 <- df1; df2$b <- as.factor(df2$b)
str(convMatr2df(df2))
```

Description

This function checks if input vector/character string contains numbers (with or without comma) and attempts converting to numeric. This functions was designed for extracting the numeric part of character-vectors (or matrix) containing both numbers and character-elements. Depending on the parameters `convert` and `remove` text-entries can be converted to NA (in resulting numeric objects) or removed (the number of elements/lines gets reduced, in consequece). Note: if `x` is a matrix, its matrix-dimensions & -names will be preserved. Note: so far Inf and -Inf do not get recognized as numeric.
**Usage**

```r
convToNum(
  x,
  autoConv = TRUE,
  spaceRemove = TRUE,
  convert = c(NA, "sparseChar"),
  remove = NULL,
  euroStyle = TRUE,
  sciIncl = TRUE,
  callFrom = NULL,
  silent = TRUE
)
```

**Arguments**

- **x** vector to be converted
- **autoConv** (logical) simple automatic conversion based on `as.numeric`; if TRUE all other arguments except `spaceRemove` will not be considered
- **spaceRemove** (logical) to remove all heading and trailing (white) space (until first non-space character)
- **convert** (character) define which type of non-conform entries to convert to NAs. Note, if `remove` is selected to eliminate character-entries they cannot be converted any more. Use 'allChar' for all character-entries; 'sparseChar' sparse (ie rare) character entries; NA for converting 'Na' or 'na' to NA; if 'none' or NULL no conversions at all.
- **remove** (character) define which type of non-conform entries to remove, removed items cannot be converted to NA any more. Use 'allChar' for removing all character entries; NA for removing all instances of NA (except thise created by converting text); all elements will be kept if 'none' or NULL.
- **euroStyle** (logical) if TRUE will convert all ',' (eg used as European decimal-separator) to '.' (as internally used by R as decimal-separator), thus allowing converting the European decimal format.
- **sciIncl** (logical) include recognizing scientific notation (eg 2e-4)
- **callFrom** (character) allow easier tracking of messages produced
- **silent** (logical) suppress messages

**Details**

This function may be used in two modes, depening if argument `autoConv` is TRUE or FALSE. The first options allows accessing an automatic mode based on `as.numeric`, while the second options investigates all characters if they may belong to numeric expressions and allows removing specific text-elements.

**Value**

This function returns a numeric vector (or matrix (if 'x' is matrix))
coordOfFilt

get coordinates of values/points in matrix according to filtering condition

Description

Get coordinates of values/points in matrix according to filtering condition

Usage

coordOfFilt(mat, cond, sortByRows = FALSE, silent = FALSE, callFrom = NULL)

Arguments

mat (matrix or data.frame) matrix or data.frame
cond (logical or integer) condition/test to see which values of mat fulfull test, or integer of index passing
sortByRows (logical) optional sorting of results by row-index
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Value

matrix columns 'row' and 'col'

See Also

which
correctToUnique checks `x` for unique entries, while maintaining the original length. If necessary a counter will be added to non-unique entries.

Usage

```r
correctToUnique(
  x,
  sep = "_",
  atEnd = TRUE,
  maxIter = 4,
  NAenum = TRUE,
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

- `x` input character vector
- `sep` (character) separator used when adding counter
- `atEnd` (logical) decide location of placing the counter (at end or at beginning of initial text)
- `maxIter` (numeric) max number of iterations
- `NAenum` (logical) if TRUE all NAs will be enumerated (NA_1,NA_2,...)
- `silent` (logical) suppress messages
- `callFrom` (character) for better tracking of use of functions

Value

This function returns a character vector

See Also

- `unique` will simply remove repeated elements, i.e. length of `x` won’t remain constant, `filtSizeUniq` is more complex and slower. `treatTxtDuplicates`
correctWinPath

Examples

```
correctToUnique(c("li0","n",NA,NA,rep(c("li2","li3"),2),rep("n",4)))
```

correctWinPath	Correct mixed slash and backslash in file path

Description

This function corrects paths character strings for mixed slash and backslash in file path. In Windows the function `tempdir()` will use double backslashes as separator while `file.path()` uses regular slashes. So when combining these two one might encounter a mix of slashes and double backslashes which may cause trouble, unless this is straightened out to a single separator used. When pointing to given files inside html-files, paths need to have a prefix, this can be added using the argument `asHtml`.

Usage

```
correctWinPath(
  x,
  asHtml = FALSE,
  anyPlatf = FALSE,
  silent = TRUE,
  callFrom = NULL
)
```

Arguments

- **x** (character) input path to test and correct
- **asHtml** (logical) option for use in html : add prefix "file:/"
- **anyPlatf** (logical) if TRUE, checking will only be performed in Windows environment
- **silent** (logical) suppress messages
- **callFrom** (character) allows easier tracking of message(s) produced

Value

character vector with corrected path

See Also

`tempfile`, `file.path`

Examples

```
path1 <- 'D:\temp\Rtmp6X8\working_dir\RtmpKC/example.txt'
(path1b <- correctWinPath(path1, anyPlatf=TRUE))
(path1h <- correctWinPath(path1, anyPlatf=TRUE, asHtml=TRUE))
```
countCloseToLimits

Count from two vectors number of values close within given limits

Description

This function summarizes the search of similar (or identical) numeric values from 2 initial vectors, it evaluates the result from initial search run by findCloseMatch(), whose output is a less convenient list. countCloseToLimits checks furthermore how many results within additional (more stringent) distance-limits may be found and returns the number of distance values within the limits tested. Designed for checking if threshold used with findCloseMatch() may be set more stringent, eg when searching reasonable FDR limits ...

Usage

countCloseToLimits(closeMatch, limitIdent = 5, prefix = "lim_")

Arguments

closeMatch (list) output from findCloseMatch(), ie list indicating which instances of 2 series of data have close matches

limitIdent (numeric) max limit or panel of threshold values to test (if single value, in addition a panel with values below will be tested)

prefix (character) prefix for names of output

Value

integer vector with counts for number of list-elements with at least one absolute value below threshold, names

See Also

findCloseMatch

Examples

set.seed(2019); aa <- sample(12:15,20,replace=TRUE) + runif(20,2)*0.5
bb <- 11:18
match1 <- findCloseMatch(aa,bb,com="diff",lim=0.65)
head(match1)
(tmp3 <- countCloseToLimits(match1,lim=c(0.5,0.35,0.2)))
(tmp4 <- countCloseToLimits(match1,lim=0.7))
countSameStartEnd  

Count same start- and end- sites of edges (or fragments)

Description

Suppose a parent sequence/string 'ABCDE' gets cut in various fragments (eg 'ABC','AB' ...).
countSameStartEnd counts how many (ie re-occurring) start- and end- sites of edges do occur in the
input-data. The input is presented as matrix of/indicating start- and end-sites of edges. The function
is used to characterize partially redundant edges and accumulation of cutting/breakage sites.

Usage

```r
countSameStartEnd(frag, minFreq = 2, nDig = 4)
```

Arguments

- `frag`  
  (matrix) 1st column beg start-sites, 2nd column end end-sites of edges, row-names to precise fragment identities are recommended
- `minFreq`  
  (integer) min number of accumulated sites for taking into account (allows filtering with large datasets)
- `nDig`  
  (integer) rounding: number of digits for columns beg.rat and end.rat in output

Value

matrix of 6 columns: input (beg and end), beg.n, beg.rat, end.n, end.rat

See Also

- to build initial tree `buildTree`, `contribToContigPerFrag`, `simpleFragFig`

Examples

```r
frag1 <- cbind(beg=c(2,3,7,13,13,15,7,9,7,3,3,5), end=c(6,12,8,18,20,20,19,12,12,4,5,7))
rownames(frag1) <- letters[1:nrow(frag1)]
countSameStartEnd(frag1)
simpleFragFig(frag1)
```
cutArrayInCluLike

**Description**

cutArrayInCluLike cuts 'dat' (matrix, data.frame or 3-dim array) in list (of appended lines) according to 'cluOrg', which serves as instruction which line of 'dat' should be placed in which list-element (like sorting according to cluster-numbers).

**Usage**

cutArrayInCluLike(dat, cluOrg, silent = FALSE, debug = FALSE, callFrom = NULL)

**Arguments**

dat array (3 dim)
cluOrg (factor) organization of lines to clusters
silent (logical) suppress messages
debug (logical) display additional messages for debugging
callFrom (character) allow easier tracking of message(s) produced

**Value**

This function retruns a list of matrixes (or arrays)

**Examples**

mat1 <- matrix(1:30,nc=3,dimnames=list(letters[1:10],1:3))
cutArrayInCluLike(mat1,cluOrg=factor(c(2,rep(1:4,2),5)))

---

cutAtMultSites

**Description**

This function cuts character vector after 'cutAt' (ie keep the search subtsing 'cutAt', different to strsplit). Used for theoretical enzymatic digestion (eg in proteomics)

**Usage**

cutAtMultSites(y, cutAt)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>character vector (better if of length=1, otherwise one won't know which fragment stems from which input)</td>
</tr>
<tr>
<td>cutAt</td>
<td>(character) search substring, ie 'cutting rule'</td>
</tr>
</tbody>
</table>

Value

modified (ie cut) character vector

See Also

`strsplit`, `nFragments0`, `nFragments`

Examples

```r
tmp <- "MSVSRMDSCELSDLVYVTERRLVAVFSTANEENFRSNLREVAQMLSKGNYLLLFNLSSRFDPITKLHAKVLEFGWPDHTPALEKI"
cutAtMultSites(c(tmp,"ojioRij"),c("R","K"))
```

---

**cutToNgrp**

*Cut numeric vector to n groups (ie convert to factor)*

**Description**

cutToNgrp is a more elaborate version of `cut` for cutting the content of a numeric vector 'x' into a given number of groups, taken from the length of 'lev'. Besides, this function provides the group borders/limits for convenient use with legends.

**Usage**

cutToNgrp(x, lev, NAuse = FALSE, callFrom = NULL)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>numeric vector</td>
</tr>
<tr>
<td>lev</td>
<td>(character or numeric), the length of this argument tells the number of groups to be used for cutting</td>
</tr>
<tr>
<td>NAuse</td>
<td>(logical) include NAs as separate group</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) for better tracking of use of functions</td>
</tr>
</tbody>
</table>

**Value**

list with $grouped telling which element of 'x' goes in which group and $legTxt with group-borders for convenient use with legends

**See Also**

`cut`
Examples

```r
cutToNgrp(dat,1:5)
plot(dat,col=(1:5)[as.numeric(cutToNgrp(dat,1:5)$grouped)])
```

---

diffCombin

Compute matrix of differences for all pairwise combinations of numeric vector

Description

diffCombin returns matrix of differences (eg resulting from substitution) for all pairwise combinations of numeric vector `x`.

Usage

diffCombin(x, diagAsNA = FALSE, prefix = TRUE, silent = FALSE, callFrom = NULL)

Arguments

- **x**: numeric vector to compute differences for all combinations
- **diagAsNA**: (logical) return all self-self combinations as NA (otherwise 0)
- **prefix**: (logical) if TRUE, dimnames of output will specify orientation (prefix='from.' and 'to.')</n- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of message(s) produced

Value

numeric matrix of all pairwise differences

See Also

diff for simple differences

Examples

diffCombin(c(10,11.1,13.3,16.6))
diffPPM

Difference in ppm between numeric values

Description

This is a `diff()`-like function to return difference in ppm between subsequent values. Result is oriented, i.e., neg ppm value means decrease (from higher to lower value). Note that if the absolute difference remains the same the difference in ppm will not remain same. Any difference to NA is returned as NA, thus a single NA will result in two NAs in output (unless NA is 1st or last).

Usage

```r
diffPPM(dat, toPrev = FALSE, silent = FALSE, callFrom = NULL)
```

Arguments

- `dat` (numeric) vector for calculating difference to preceeding/following value in ppm
- `toPrev` (logical) determine orientation
- `silent` (logical) suppress messages
- `callFrom` (character) allows easier tracking of messages produced

Value

This function returns a list with close matches of 'x' to given 'y', the numeric value depends on 'sortMatch' (if FALSE then always value of 'y' otherwise of longest of x&y)

See Also

- `checkSimValueInSer` and (from this package) `.compareByDiff`, `diff`

Examples

```r
aa <- c(1000.01, 1000.02, 1000.05, 1000.08, 1000.09, 1000.08)
.compareByPPM(list(aa, aa), 30, TRUE) # tabular 'long' version
diffPPM(aa)
```
elimCloseCoord  

**Description**

`elimCloseCoord` reduces number of rows in 'dat' by eliminating lines where x & y coordinates (columns of matrix 'dat' defined by 'useCol') are identical (overlay points) or very close. The stringency for 'close' values may be fine-tuned using `nDig`), this function uses internally `firstOfRepeated`.

**Usage**

```r
elemCloseCoord(
  dat,
  useCol = 1:2,
  elimIdentOnly = FALSE,
  refine = 2,
  nDig = 3,
  callFrom = NULL,
  silent = FALSE
)
```

**Arguments**

- `dat` matrix (or data.frame) with main numeric input
- `useCol` (numeric) index for numeric columns of 'dat' to use/consider
- `elimIdentOnly` (logical) if TRUE, eliminate real duplicated points only (ie identical values only)
- `refine` (numeric) allows increasing stringency even further (higher 'refine' .. more lines considered equal)
- `nDig` (integer) number of significant digits used for rounding, if two 'similar' values are identical after this rounding the second will be eliminated.
- `callFrom` (character) allows easier tracking of message(s) produced
- `silent` (logical) suppress messages

**Value**

resultant matrix/data.frame

**See Also**

`findCloseMatch`, `firstOfRepeated`

**Examples**

```r
da1 <- matrix(c(rep(0:4,5),0.01,1.1,2.04,3.07,4.5),nc=2); da1[,1] <- da1[,1]*99; head(da1)
elimCloseCoord(da1)
```
equLenNumber | Equal character-length number

Description

equLenNumber convert numeric entry 'x' to text, with all elements getting the same number of characters (ie by adding preceeding or tailing 0s, if needed). So far, the function cannot handle scientific annotations.

Usage

equLenNumber(x, silent = FALSE, callFrom = NULL, debug = FALSE)

Arguments

x (character) input vector
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced
debug (logical) additional messages for debugging

Value

character vector formatted as equal number of characters per value

See Also

sprintf

Examples

equLenNumber(c(12,-3,321))
equLenNumber(c(12,-3,3,321))

exclExtrValues | Exclude extreme values (based on distance to mean)

Description

This function aims to identify extreme values (values most distant to mean, thus potential outliers), mark them as NA or directly exclude them (depending on 'showNAs'). Note that every set of non-identical values will have at least one most extreme value. Extreme values are part of many distributions, they are not necessarily true outliers.
Usage

```r
exclExtrValues(
  dat,
  result = "val",
  CVlim = NULL,
  maxExcl = 1,
  showNA = FALSE,
  goodValues = TRUE,
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

- **dat**
  - numeric vector, main input
- **result**
  - (character) may be 'val' for returning data without extreme values or 'pos' for returning position/index of extreme values
- **CVlim**
  - (NULL or numeric) allows to retain extreme values only if a certain CV (for all 'dat') is exceeded (to avoid calling extreme values from homogenous data-sets)
- **maxExcl**
  - (integer) max number of elements to exclude
- **showNA**
  - (logical) will display extreme values as NA
- **goodValues**
  - (logical) allows to display rather the good values instead of the extreme values
- **silent**
  - (logical) suppress messages
- **callFrom**
  - (character) allow easier tracking of message(s) produced

Value

numeric vector wo extremle values or index-position of extreme values

See Also

`firstOfRepLines`, `get1stOfRepeatedByCol` for treatment of matrix

Examples

```r
x <- c(rnorm(30),-6,20)
exclExtrValues(x)
```
Description

This function normalizes 'dat' by optimizing exponent function (ie dat ^exp) to fit best to 'ref' (default: average of each line of 'dat').

Usage

```r
exponNormalize(
  dat,
  useExpon,
  dynExp = TRUE,
  nStep = 20,
  startExp = 1,
  simMeas = "cor",
  refDat = NULL,
  refGrp = NULL,
  refLines = NULL,
  rSquare = FALSE,
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

- `dat`: matrix or data.frame of numeric data to be normalized
- `useExpon`: (numeric vector or matrix) exponent values to be tested
- `dynExp`: (logical) require 'useExpon' as 2 values (matrix), will gradually increase exponent from 1st to 2nd; may be matrix or data.frame for dynamic, in this case 1st line for exp for lowest data, 2nd line for highest
- `nStep`: (integer) number of exponent variations (steps) when testing range from-to
- `startExp`: (numeric)
- `simMeas`: (character) similarity metric to be used (so far only "cor"), if rSquare=TRUE, the r-squared will be returned
- `refDat`: (matrix or data.frame) if null average of each line from 'dat' will be used as reference in similarity measure
- `refGrp`: (factor) designing which col of 'ref' should be used with which col of 'dat' (length equal to number of cols in 'dat'). Note: 'refGrp' not yet coded optimally to extract numeric part of character vector, potential problems when all lines or cols of dat are NA
- `refLines`: (NULL or integer) optional subset of lines to be considered (only) when determining normalization factors
rSquare (logical) if TRUE, add r-squared
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced

Value
This function returns a matrix of normalized data

See Also
more evolved than normalizeThis with argument set to ’exponent’

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),nc=10)
head(rowGrpCV(dat1,gr=gl(4,3,labels=LETTERS[1:4])[2:11]))
set.seed(2016); dat1 <- c(0,1,0.2,0.3,0.5)*rep(c(1,10),each=4)
dat1 <- matrix(round(c(sqrt(dat1),dat1^1.5,3*dat1+runif(length(dat1))),2),nc=3)
dat2a <- exponNormalize(dat1[,1],useExpon=2,nSte=1,refD=dat1[,3])
layout(matrix(1:2,nc=2))
plot(dat1[,1],dat1[,3],type="b",main="init",ylab="ref")
plot(dat2a$datNor[,1],dat1[,3],type="b",main="norm",ylab="ref")
dat2b <- exponNormalize(dat1[,1],useExpon=c(1.7,2.3),nSte=5,refD=dat1[,3])
plot(dat1[,1],dat1[,3],type="b",main="init",ylab="ref")
plot(dat2b$datNor[,1],dat1[,3],type="b",main="norm",ylab="ref")

dat2c <- exponNormalize(dat1[,-3],useExpon=matrix(c(1.7,2.3,0.6,0.8),nc=2),nSte=5,refD=dat1[,3])
plot(dat1[,1],dat1[,3],type="b",main="init",ylab="ref")
plot(dat2c$datNor[,1],dat1[,3],type="b",main="norm 1",ylab="ref")
plot(dat1[,2],dat1[,3],type="b",main="init",ylab="ref")
plot(dat2c$datNor[,2],dat1[,3],type="b",main="norm 2",ylab="ref");
```

extr1chan

`Extract just one series, ie channel, of list of arrays`

Description
This function was designed for handling measurements stored as list of multiple arrays, like eg compound-screens using microtiter-plates where multiple parameters (‘channels’) were recorded for each well (element). The elements (eg compounds screened) are typically stored in the 1st dimension of the arrays, the replicated in the 2nd dimension and different measure types/parameters in the 3rd channel. In order to keep the structure of of individual microtiter-plates, typically each plate forms a separate array (of same dimensions) in a list. The this function allows extracting a single channel of the list of arrays (3rd dim of each array) and return row-appended matrix.

Usage
extr1chan(arrLst, cha, na.rm = TRUE, rowSep = "__")
extractLast2numericParts

Description

extractLast2numericParts extracts last 2 (integer) numeric parts between punctuations out of character vector 'x'. Runs faster than gregexpr. Note: won't work correctly with decimals or exponential signs !! (such characters will be considered as punctuation, ie as separator)

Usage

extractLast2numericParts(x, silent = FALSE, callFrom = NULL)

Arguments

  x     main character input
  silent (logical) suppress messages
  callFrom (character) allow easier tracking of message(s) produced

Value

  (numeric) matrix with 2 columns (eg from initial concatenated coordinates)
extrColsDeX

See Also
gregexpr from grep

Examples
extractLast2numericParts(c("M01.1-4","M001/2.5","M_0001_03-16","zyx","012","a1.b2.3-7,2"))

dFr <- data.frame(a=11:14, b=24:21, cc=LETTERS[1:4], dd=rep(c(TRUE,FALSE),2))
eextrColsDeX(dFr,c("b","cc","notThere"))
eextrColsDeX(dFr,c("b","cc","notThere"), doExtractCols=TRUE)
eextrColsDeX(dFr, list(c("nn","b","a"), c("cc","a"),"notThere"))

Description
This function provides flexible checking if a set of columns may be extracted from a matrix or
data.frame 'x'. If argument extrCol is list of character vectors, this allows to search among given
options, the first matching name for each vector will be identified.

Usage
eextrColsDeX(x, extrCol, doExtractCols = FALSE, callFrom = NULL, silent = FALSE)

Arguments
  x
    (matrix or data.frame) main input (where data should be extracted from)
  extrCol
    (character, integer or list) columns to be extracted, may be column-names or
column index; if is list each first-level element will be considered as options
for one choice
  doExtractCols
    (logical) if default FALSE only the column indexes will be returned
  callFrom
    (character) allows easier tracking of message(s) produced
  silent
    (logical) suppress messages

Value
integer-vector (if doExtractCols=FALSE return depending on input matrix or data.frame)

See Also
read.table, filterList

Examples
dFr <- data.frame(a=11:14, b=24:21, cc=LETTERS[1:4], dd=rep(c(TRUE,FALSE),2))
eextrColsDeX(dFr,c("b","cc","notThere"))
eextrColsDeX(dFr,c("b","cc","notThere"), doExtractCols=TRUE)
eextrColsDeX(dFr, list(c("nn","b","a"), c("cc","a"),"notThere"))
extrNumericFromMatr  *Extract numeric part of matrix or data.frame*

**Description**

`extrNumericFromMatr` extracts numeric part of matrix or data.frame, removing remaining non-numeric elements if `trimToData` is set to TRUE. Note, that cropping entire lines where a (single) text element appeared may quickly reduce the overall content of the input data.

**Usage**

```r
extrNumericFromMatr(dat, trimToData = TRUE, silent = FALSE, callFrom = NULL)
```

**Arguments**

- `dat` matrix (or data.frame) for extracting numeric parts
- `trimToData` (logical) default to remove (crop) lines and cols contributing to NA, non-numeric data is transformed to NA
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of message(s) produced

**Value**

matrix of numeric data

**Examples**

```r
mat <- matrix(c(letters[1:7],14:16,LETTERS[1:6]),nrow=4,dimnames=list(1:4,letters[1:4]))
mat; extrNumericFromMatr(mat)

mat <- matrix(c(letters[1:4],1,"e",12:19,LETTERS[1:6]),nr=5,dimnames=list(11:15,letters[1:4]))
mat; extrNumericFromMatr(mat)
```

---

extrSpecText  *Extract specific text*

**Description**

This function extracts/cuts text-fragments out of `txt` following specific anchors defined by arguments `cutFrom` and `cutTo`.

---
Usage

extrSpcText(
  txt,
  cutFrom = " GN=",
  cutTo = " PE=",
  missingAs = NA,
  exclFromTag = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

txt character vector to be treated
cutFrom (character) text where to start cutting
cutTo (character) text where to stop cutting
missingAs (character) specific content of output at line/location of 'exclLi'
exclFromTag (logical) to exclude text given in 'cutFrom' from result
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Details

In case cutFrom is not found missingAs will be returned. In case cutTo is not found, text gets extracted with chaMaxEl characters.

Value

This function returns a modified character vector

See Also

substr

Examples

eextrSpcText(c(" ghjg GN=thisText PE=001"," GN=_ PE="), NA, "abcd"))
eextrSpcText(c("ABCDEF.3-6","05g","bc.4-5"), cutFr="\." , cutT="-")
filter3dimArr  
Filter three-dimensional array of numeric data

Description
Filtering of matrix or (3-dim) array x: filter column according to filtCrit (e.g., 'inf') and threshold filtVal.

Usage
```
filt3dimArr(
  x,
  filtVal,
  filtTy = ">",
  filtCrit = NULL,
  displCrit = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments
- **x**: array (3-dim) of numeric data
- **filtVal**: (numeric, length=1) for testing inferior/superior/equal condition
- **filtTy**: (character, length=1) which type of testing to perform (may be 'eq', 'inf', 'infeq', 'sup', 'supeq', '>','<', '>=','<=','==')
- **filtCrit**: (character, length=1) which column-name consider when filtering filter with 'filtVal' and 'filtTy'
- **displCrit**: (character) column-name(s) to display
- **silent**: (logical) suppress messages
- **debug**: (logical) additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Details
and extract/display all col matching 'displCrit'.

Value
This function returns a list of filtered matrixes (by 3rd dim)

See Also
- `filterList`
- `filterLiColDeList`
Examples

arr1 <- array(11:34, dim=c(4,3,2), dimnames=list(c(LETTERS[1:4]), paste("col",1:3,sep=""), c("ch1","ch2"))
filt3dimArr(arr1,displCrit=c("col1","col2"),filtCrit="col2",filtVal=7)

filterLiColDeList Filter lines(rows) and/or columns from all suitable elements of list

Description

Filter all elements of list (or S3-object) according to criteria designed to one selected reference-element of the list. All simple vectors, matrix, data.frames and 3-dimensional arrays will be checked if matching number of rows and/or columns to decide if they should be filtered the same way. If the reference element has same number of rows and columns simple (1-dimensional) vectors won’t be filtered since it not clear if this should be done to lines or columns.

Usage

filterLiColDeList(
  lst, useLines, useCols = NULL, ref = 1, silent = FALSE, callFrom = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst</td>
<td>(list or S3 object) main input</td>
</tr>
<tr>
<td>useLines</td>
<td>(integer, logical or character) vector to assign lines to keep when filtering along lines; set to NULL for no filtering; if 'allNA' all lines composed uniquely of NA values will be removed.</td>
</tr>
<tr>
<td>useCols</td>
<td>(integer, logical or character) vector for filtering columns; set to NULL for no filtering; if 'allNA' all columns uniquely NA values will be removed.</td>
</tr>
<tr>
<td>ref</td>
<td>(integer) index for designing the element of 'lst' to take as reference for checking which other list-elements have suitable number of rows or columns</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allow easier tracking of message(s) produced</td>
</tr>
</tbody>
</table>

Details

This function is used eg in package wrProteo to simultaneously filter raw and transformed data.

Value

this function returns the correct(ed) input (object of same class, same length)
See Also

`moderTest2grp` for single comparisons, `lmFit`

Examples

```r
lst1 <- list(m1=matrix(11:18,ncol=2), m2=matrix(21:30,ncol=2), indR=31:34,
             m3=matrix(c(21:23,NA,25:27,NA),ncol=2))
## here $m2 has more lines than $m1, and thus will be ignored when ref=1
filterLiColDeList(lst1, useLines=2:3)
filterLiColDeList(lst1, useLines="allNA", ref=4)
```

---

**filterList**  
*Filter for unique elements*

**Description**

This function aims to apply a given filter-criterium, a matrix or vector of FALSE/TRUE which is typically combined with a second layer which filters for a min content of filter-passing values per line for the first/main criterium. Then all lines concerned will be removed. This will be done for all list-elements (of appropriate size) of the input-list (while maintaining the list-structure in the output) not matching the filtering criteria.

**Usage**

```r
filterList(lst, filt, minLineRatio = 0.5, silent = FALSE, callFrom = NULL)
```

**Arguments**

- `lst` (list) main input, each vector, matrix or data.frame in this list will be filtered if its length or number of lines fits to `filt`
- `filt` (logical) vector of FALSE/TRUE to use for filtering. If this a matrix is given, the value of `minLineRatio` will be applied as threshold of min content of `TRUE` for each line of `filt`
- `minLineRatio` (numeric) in case `filt` is a matrix of FALSE/TRUE, this value will be used as threshold of min content of `TRUE` for each line of `filt`
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of message(s) produced

**Value**

filtered list

**See Also**

`correctToUnique`, `unique`, `duplicated`, `extrColsDeX`
Examples

```r
set.seed(2020); dat1 <- round(runif(80),2)
list1 <- list(m1=matrix(dat1[1:40],ncol=8), m2=matrix(dat1[41:80],ncol=8), other=letters[1:8])
rownames(list1$m1) <- rownames(list1$m2) <- paste0("line",1:5)
filterList(list1, list1$m1[,1] >0.4)
filterList(list1, list1$m1 >0.4)
```

filterNetw

Filter nodes & edges for extracting networks This function allows extracting and filtering network-data based on fixed threshold (limInt) and add sandwich-nodes (nodes inter-connecting initial nodes) out of node-based queries.

Description

Filter nodes & edges for extracting networks

This function allows extracting and filtering network-data based on fixed threshold (limInt) and add sandwich-nodes (nodes inter-connecting initial nodes) out of node-based queries.

Usage

```r
filterNetw(
  lst,
  filtCol = 3,
  limInt = 5000,
  sandwLim = 5000,
  filterAsInf = TRUE,
  outFormat = "matrix",
  remOrphans = TRUE,
  remRevPairs = TRUE,
  elemNa = "genes",
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

Arguments

- `lst` (list, composed of multiple matrix or data.frames ) main input (each list-element should have same number of columns)
- `filtCol` (integer, length=1) which column of lst should be used to filter using thresholds `limInt` and `sandwLim`
- `limInt` (numeric, length=1) filter main edge-scores according to `filterAsInf`
- `sandwLim` (numeric, length=1) filter sandwich connection edge-scores according to `filterAsInf`
- `filterAsInf` (logical) filter as 'inferior or equal' or 'superior or equal'
outFormat (character) may be 'matrix' for tabular output, 'all' as list with matrix and list of node-names

remOrphans (logical) remove networks consisting only of 2 connected edges

remRevPairs (logical) remove duplicate edges due to reverse massping (eg A - B and B - A); NOTE: use only when edges don’t have orientation!

elemNa (character) used only for messages

silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced
debug (logical) display additional messages for debugging

Value

This function returns a matrix or data.frame

See Also

in cbind

Examples

```r
lst2 <- list('121'=data.frame(ID=as.character(c(141,221,228,229,449)),11:15),
'131'=data.frame(ID=as.character(c(228,331,332,333,339)),11:15),
'141'=data.frame(ID=as.character(c(121,151,229,339,441,442,449)),c(11:17)),
'151'=data.frame(ID=as.character(c(449,141,551,552)),11:14),
'161'=data.frame(ID=as.character(171),11), '171'=data.frame(ID=as.character(161),11),
'181'=data.frame(ID=as.character(881:882),11:12))

lst2 <- list('121'=data.frame(ID=as.character(c(141,221,228,229,449)),11:15, 21:25),
'131'=data.frame(ID=as.character(c(228,331,332,333,339)),11:15, 21:25),
'141'=data.frame(ID=as.character(c(121,151,229,339,441,442,449)), c(11:17), 21:27),
'151'=data.frame(ID=as.character(c(449,141,551,552)), 11:14, 21:24),
'161'=data.frame(ID=as.character(171), 11,21), '171'=data.frame(ID=as.character(161), 11,21),
'181'=data.frame(ID=as.character(881:882), 11:12,21:22))

(te1 <- filterNetw(lst2, limInt=90, remOrphans=FALSE))
(te2 <- filterNetw(lst2, limInt=90, remOrphans=TRUE))

(te3 <- filterNetw(lst2, limInt=13, remOrphans=FALSE))
(te4 <- filterNetw(lst2, limInt=13, remOrphans=TRUE))
```
filtSizeUniq  
*Filter for unique elements*

**Description**

This function aims to identify and remove duplicated elements in a list and maintain the list-structure in the output. `filtSizeUniq` filters ‘lst’ (list of character-vectors or character-vector) for elements being unique (to ‘ref’ or if NULL to all ‘lst’) and of character length. In addition, the min- and max- character length may be filtered, too. Eg, in proteomics this helps removing peptide sequences which would not be measured/detected any way.

**Usage**

```r
filtSizeUniq(
  lst,
  ref = NULL,
  minSize = 6,
  maxSize = 36,
  filtUnique = TRUE,
  byProt = TRUE,
  inclEmpty = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `lst`  
  list of character-vectors or character-vector

- `ref`  
  (character) optional alternative ‘reference’, if not NULL used in addition to ‘lst’ for considering elements of ‘lst’ as unique

- `minSize`  
  (integer) minimum number of characters, if NULL set to 0

- `maxSize`  
  (integer) maximum number of characters

- `filtUnique`  
  (logical) if TRUE return unique-only character-strings

- `byProt`  
  (logical) if TRUE organize output as list (by names of input, eg protein-names) - if ‘lst’ was named list

- `inclEmpty`  
  (logical) optional including empty list-elements when all elements have been filtered away - if ‘lst’ was named list

- `silent`  
  (logical) suppress messages

- `debug`  
  (logical) additional messages for debugging

- `callFrom`  
  (character) allow easier tracking of messages produced

**Value**

list of filtered input
findCloseMatch

Find close numeric values between two vectors

Description

findCloseMatch finds close matches (similar values) between two numeric vectors (`x`, `y`) based on method `compTy` and threshold `limit`. Return list with close matches of `x` to given `y`, the numeric value depends on `sortMatch` (if FALSE then always value of `y` otherwise of longest of x&y). Note: Speed & memory improvement if `sortMatch`=TRUE (but result might be inverted!): adopt search of x->y or y->x to searching matches of each longest to each shorter (ie flip x & y). Otherwise, if length of `x` & `y` are very different, it may be advantageous to use a long(er) `x` and short(er) `y` (with `sortMatch`=FALSE). Note: Names of `x` & `y` or (if no names) prefix letters `x` & `y` are always added as names to results.

Usage

```r
closeMatch(x, y, compTy = "ppm", limit = 5, asIndex = FALSE, maxFitShort = 100, sortMatch = FALSE, silent = FALSE, debug = FALSE, callFrom = NULL)
```

Arguments

- **x**: numeric vector for comparison
- **y**: numeric vector for comparison
- **compTy**: (character) may be `diff` or `ppm`, will be used with threshold from argument `limit`
- **limit**: (numeric) threshold value for retaining values, used with distance-type specified in argument `compTy`

See Also

correctToUnique, unique, duplicated

Examples

```r
filtSizeUniq(list(A="a",B=c("b","bb","c"),D=c("dd","d","ddd","c")),filtUn=TRUE,minSi=NULL) # input: c and dd are repeated
filtSizeUniq(list(A="a",B=c("b","bb","c"),D=c("dd","d","ddd","c")),ref=c(letters[c(1:26,1:3)], "dd","dd","bb","ddd"),filtUn=TRUE,minSi=NULL) # a,b,c,dd repeated
```
Find repeated elements

Description

findRepeated gets index of repeated items/values in vector 'x' (will be treated as character). Return (named) list of indexes for each of the repeated values, or NULL if all values are unique. This approach is similar but more basic compared to get1stOfRepeatedByCol.
findSimilFrom2sets

Usage

findRepeated(x, nonRepeated = FALSE, silent = FALSE, callFrom = NULL)

Arguments

x character vector
nonRepeated (logical) if =TRUE, return list with elements $rep and $nonrep
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Value

(named) list of indexes for each of the repeated values, or NULL if all values unique

See Also

similar approach but more basic than get1stOfRepeatedByCol

Examples

aa <- c(11:16,14:12,14); findRepeated(aa)

findSimilFrom2sets Find similar numeric values from two vectors/matrixes

Description

findSimilFrom2sets compares to vectors or matrixes and returns combined view including only all close (by findCloseMatch). Return matrix (predMatr) with add'l columns for index to and 'grp' (group of similar values (1-to-many)), 'nGrp' (n of grp), 'isBest' or 'nBest', 'disToMeas' (distance/difference between pair) & 'ppmToPred' (distance in ppm). Note: too wide 'limitComp' will result in large window and many 'good' hits will compete (and be mutually excluded) if selec- tion 'bestOnly' is selected

Usage

findSimilFrom2sets(
    predMatr,
    measMatr,
    colMeas = 1,
    colPre = 1,
    compareTy = "diff",
    limitComp = 0.5,
    bestOnly = FALSE,
    silent = FALSE,
    callFrom = NULL,
    debug = FALSE
)
findSimilFrom2sets

Arguments

predMatr  (matrix or numeric vector) dataset number 1, referred to as 'predicted', the column specified in argument colPre points to the data to be used

measMatr  (matrix or numeric vector) dataset number 2, referred to as 'measured', the column specified in argument colMeas points to the data to be used

colMeas  (integer) which column number of 'measMatr' to consider

colPre  (integer) which column number of 'predMatr' to consider

compareTy  (character) 'diff' (difference) 'ppm' (relative difference)

limitComp  (numeric) limit used by 'compareTy'

bestOnly  (logical) allows to filter only hits with min distance (defined by 'compareTy'), 3rd last col will be 'nBest' - otherwise 3rd last col 'isBest'

silent  (logical) suppress messages

callFrom  (character) allow easier tracking of messages produced

debug  (logical) for bug-tracking: more/enhanced messages

Value

This function returns a matrix (predMatr) with add'l columns for index to and 'grp' (group of similar values (1-to-many)), 'nGrp' (n of grp), 'isBest' or 'nBest', 'disToMeas' (distance/difference between pair) & 'ppmToPred' (distance in ppm)

See Also

checkSimValueInSer findCloseMatch closeMatchMatrix

Examples

aA <- c(11:17); bB <- c(12.001,13.999); cC <- c(16.2,8,9,12.5,12.6,15.9,14.1)
aZ <- matrix(c(aA,aA+20),ncol=2,dimnames=list(letters[1:length(aA)],c("aaA","aZ")))
cZ <- matrix(c(cC,cC+20),ncol=2,dimnames=list(letters[1:length(cC)],c("ccC","cZ")))
findCloseMatch(cC,aA,com="diff",lim=0.5,sor=FALSE)
findSimilFrom2sets(aA,cC)
findSimilFrom2sets(cC,aA)
findSimilFrom2sets(aA,cC,best=FALSE)
findSimilFrom2sets(aA,cC,comp="ppm",lim=5e4,deb=TRUE)
findSimilFrom2sets(aA,cC,comp="ppm",lim=9e4,bestO=FALSE)
# below: find fewer 'best matches' since search window larger (ie more good hits compete !)
findSimilFrom2sets(aA,cC,comp="ppm",lim=9e4,bestO=TRUE)
findUsableGroupRange  Select groups within given range

Description
This function aims to help finding stretches/segments of data with a given maximum number of NA-instances. This function is used to inspect/filter each line of 'dat' for a subset with sufficient presence/absence of NA values (ie limit number of NAs per level of 'grp'). Note: optimal performance with n.lines > n.groups

Usage
findUsableGroupRange(dat, grp, maxNA = 1, callFrom = NULL)

Arguments
  dat  (matrix or data.frame) main input
  grp  (factor) information which column of 'dat' is replicate of whom
  maxNA  (integer) max number of tolerated NAs
  callFrom  (character) allow easier tracking of message(s) produced

Value
matrix with boundaries of 1st and last usable column (NA if there were no suitable groups found)

Examples
dat1 <- matrix(1:56,nc=7)
dat1[c(2,3,4,5,6,10,12,18,19,20,22,23,26,27,28,30,31,34,38,39,50,54)] <- NA
rownames(dat1) <- letters[1:nrow(dat1)]
findUsableGroupRange(dat1,gl(3,3)[-(3:4)])

firstLineOfDat  Filter matrix to keep only first of repeated lines

Description
This function aims to reduce the complexity of a matrix (or data.frame) in case column 'refCol' has multiple lines with same value. In this case, it reduces the input-data to 1st line of redundant entries and returns a matrix (or data.frame) without lines identified as redundant entries for 'refCol'). In sum, this function works like using unique on a given column, and propagates the same treatment to all other columns.

Usage
firstLineOfDat(dat, refCol = 2, silent = FALSE, debug = FALSE, callFrom = NULL)
Arguments

- **dat**: (matrix or data.frame) main input
- **refCol**: (integer) column number of reference-column
- **silent**: (logical) suppress messages
- **debug**: (logical) additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Value

matrix (same number of columns as input)

See Also

- `firstOfRepeated`, `unique`, `duplicated`

Examples

```r
(mat1 <- matrix(c(1:6,rep(1:3,1:3)),ncol=2,dimnames=list(letters[1:6],LETTERS[1:2])))
firstLineOfDat(mat1)
```

---

**Description**

This function works similar to `unique`, but provides additional information about which elements of original input 'x' are repeated by providing indexes relative to the input. `firstOfRepeated` makes list with 3 elements: $\text{indRepeated.. index for first of repeated 'x', indUniq.. index of all unique } + \text{ first of repeated, indRedund.. index of all redundant entries, ie non-unique (wo 1st). Used for reducing data to non-redundant status, however, for large numeric input the function nonAmbiguousNum() may perform better/faster. NAs won't be considered (NAs do not appear in reported index of results), see also firstOfRepLines().

Usage

`firstOfRepeated(x, silent = FALSE, debug = FALSE, callFrom = NULL)`

Arguments

- **x**: (charcter or numeric) main input
- **silent**: (logical) suppress messages
- **debug**: (logical) display additional messages for debugging
- **callFrom**: (character) allow easier tracking of message(s) produced
Value

list with indices: $indRepeated, $indUniq, $indRedund

See Also

duplicated, nonAmbiguousNum, firstOfRepLines gives less detail in output (lines/elements/indexes of omitted not directly accessible) and works faster

Examples

```r
x <- c(letters[c(3,2:4,8,NA,3:1,NA,5:4)]); names(x) <- 100+(1:length(x))
firstOfRepeated(x)
x[firstOfRepeated(x)$indUniq]  # only unique with names
```

firstOfRepLines

Reduce to first occurrence of repeated lines

Description

This function concatenates all columns of input-matrix and then searches like unique for unique elements, optionally the indexes of unique elements may get returned. Note: This function reads input as character (thus won’t understand 10==10.0). Returns simplified/non-redundant vector/matrix (ie fewer lines), or respective index. faster than firstOfRepeated

Usage

```r
firstOfRepLines(
  mat,
  outTy = "ind",
  useCol = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat</td>
<td>initial matrix to treat</td>
</tr>
<tr>
<td>outTy</td>
<td>for output type: ‘ind’.. index to 1st occurrence (non-red),’orig’..non-red lines of mat, ‘conc’.. non-red concatenated values, ‘num’.. index to which group/category the lines belong</td>
</tr>
<tr>
<td>useCol</td>
<td>(integer) custom choice of which columns to paste/concatenate</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>debug</td>
<td>(logical) additional messages for debugging</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allow easier tracking of messages produced</td>
</tr>
</tbody>
</table>
Value

simplified/non-redundant vector/matrix (ie fewer lines for matrix), or respective index

See Also

unique, nonAmbiguousNum, faster than firstOfRepeated which gives more detail in output (lines/elements/indexes of omitted)

Examples

```r
mat <- matrix(c("e","n","a","n","z","z","n","z","b",
 "","n","c","n","","n","z","z"),ncol=2)
firstOfRepeated(mat,out="conc")
```

fuseAnnotMatr

Fuse annotation matrix to initial matrix

Description

In a number of instances experimental measurements and additional information (annotation) are provided by separate objects (matrixes) as they may not be generated the same time. The aim of this function is provide help when matching appropriate lines for 2 sets of data (experimental measures in iniTab and annotation from annotTab) for fusing. fuseAnnotMatr adds suppelmental columns/annotation to an initial matrix iniTab: using column 'refIniT' as key (in iniTab) to compare with key 'refAnnotT' (from 'annotTab'). The columns to be added from annotTab must be chosen explicitely. Note: if non-unique IDs in iniTab: runs slow (but save) due to use of loop for each unique ID.

Usage

```r
fuseAnnotMatr(
  iniTab, annotTab, refIniT = "Uniprot", refAnnotT = "combName",
  addCol = c("ensembl_gene_id", "description", "geneName", "combName"),
  debug = TRUE, silent = FALSE, callFrom = NULL)
```

Arguments

iniTab (matrix), that may have lines with multiple (=repeated) key entries
annotTab (matrix) containing reference annotation
refIniT (character) type of reference (eg 'Uniprot')
refAnnotT (character) column name to use for reference-annotation
fuseCommonListElem

Fuse content of list-elements with redundant (duplicated) names

Description

fuseCommonListElem fuses (character or numeric) elements of list re-occurring under same name, so that resultant list has unique names. Note: will not work with list of matrices

Usage

fuseCommonListElem(lst, initOrd = TRUE, removeDuplicates = FALSE, callFrom = NULL)

Arguments

lst (list) main input, list of numeric vectors
initOrd (logical) preserve initial order in output (if TRUE) or otherwise sort alphabetically
removeDuplicates
(logical) allow to remove duplicate entries (if vector contains names, both the name and the value need to be identical to be removed; note: all names must have names with more than 0 characters to be considered as names)
callFrom (character) allows easier tracking of message(s) produced

Value
fused list (same names as elements of input)

See Also
unlist

Examples
val1 <- 10 +1:26
names(val1) <- letters
lst1 <- list(c=val1[3:6],a=val1[1:3],b=val1[2:3],a=val1[12],c=val1[13])
fuseCommonListElem(lst1)

Description
Fuse previously identified pairs to 'clusters', return vector with cluster-numbers.

Usage
fusePairs(
datPair,
refDatNames = NULL,
inclRepLst = FALSE,
maxFuse = NULL,
debug = FALSE,
silent = TRUE,
callFrom = NULL
)

Arguments
datPair 2-column matrix where each line represents 1 pair
refDatNames (NULL or character) allows placing selected pairs in context of larger data-set (names to match those of 'datPair')
inclRepLst (logical) if TRUE, return list with 'clu' (clu-numbers, default output) and 'refLst' (list of clustered elements, only n>1)
get1stOfRepeatedByCol

maxFuse (integer, default NULL) maximal number of groups/clusters
debug (logical) display additional messages for debugging
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Value
This function returns a vector with cluster-numbers

Examples
daPa <- matrix(c(1:5,8,2:6,9), ncol=2)
fusePairs(daPa, maxFuse=4)

get1stOfRepeatedByCol

get1stOfRepeatedByCol Get first of repeated by column

Description
get1stOfRepeatedByCol sorts matrix 'mat' and extracts only 1st occurrence of values in column 'sortBy'. Returns then non-redundant matrix (ie for column 'sortBy', if 'markIfAmbig' specifies existing col, mark ambig there). Note: problem when sortSupl or sortBy not present (or not intended for use)

Usage
get1stOfRepeatedByCol(
  mat,
  sortBy = "seq",
  sortSupl = "ty",
  asFirstLast = c("full", "inter"),
  markIfAmbig = c("ambig", "seqNa"),
  asList = FALSE,
  abmiPref = "_"
)

Arguments
mat (matrix or data.frame) numeric vector to be tested
sortBy column name for which elements should be made unique, numeric or character column; 'sortSupl' .. add'l colname to always select specific 1st
sortSupl default="ty"
asFirstLast (character,length=2) to force specific strings from colunm 'sortSupl' as first and last when selecting 1st of repeated terms, default=c("full","inter")
markIfAmbig (character,length=2) 1st will be set to 'TRUE' if ambiguous/repeated, 2nd will get (heading) prefix, default=c("ambig","seqNa")
getValuesByUnique

- **asList**: (logical) to return list with non-redundant ('unique') and removed lines ('repeats')
- **abmiPref**: (character) prefix to note ambiguous entries/terms, default="_"

**Value**

depending on 'asList' either list with non-redundant ('unique') and removed lines ('repeats')

**See Also**

- `firstOfRepeated` for (more basic) treatment of simple vector, `nonAmbiguousNum` for numeric use (much faster !!!)

**Examples**

```r
aa <- cbind(no=as.character(1:20), seq=sample(LETTERS[1:15],20, repl=TRUE),
            ty=sample(c("full","Nter","inter"),20, repl=TRUE), ambig=rep(NA,20), seqNa=1:20)
get1stOfRepeatedByCol(aa)
```

**Description**

When data have repeated elements (defined by names inside the vector), it may be advantageous to run some operations only on a unique set of the initial data, or sometimes all repeated occurrences need to be replaced by a common (summarizing) value. This function allows to re-introduce new values from on second vector with unique names, to return a final vector of initial input-length and order of names (elements) like initial, too. Normally the user would provide 'datUniq' (without repeated names) containing new values which will be expanded to structure of 'dat', if 'datUniq' is not provided a vector with unique names will be made using the first occurance of repeated value(s). For more complex cases the indexing relative to 'datUniq' can be returned (setting asIndex=TRUE). Note: If not all names of 'dat' are found in 'datUniq' the missing spots will be returned as NA.

**Usage**

```r
getValuesByUnique(
  dat,
  datUniq = NULL,
  asIndex = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat</td>
<td>(numeric or character) main long input, must have names</td>
</tr>
<tr>
<td>datUniq</td>
<td>(numeric or character) will be used to impose values on dat, must have names</td>
</tr>
<tr>
<td>asIndex</td>
<td>(logical) if TRUE index values will be returned instead of replacing values</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>debug</td>
<td>(logical) additional messages for debugging</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allow easier tracking of messages produced</td>
</tr>
</tbody>
</table>

Value

vector of length dat with imposed values, or index values if asIndex=TRUE

See Also

unique, findRepeated, correctToUnique, treatTxtDuplicates

Examples

```r
dat <- 11:19
names(dat) <- letters[c(6:3,2:4,8,3)]
## let's make a 'datUniq' with the mean of repeated values :
datUniq <- round(tapply(dat,names(dat),mean),1)
## now propagate the mean values to the full vector
getValuesByUnique(dat,datUniq)
cbind(ini=dat,firstOfRep=getValuesByUnique(dat,datUniq),
   indexUniq=getValuesByUnique(dat,datUniq,asIn=TRUE))
```

---

**Description**

This functions converts a given urlName so that from data from git-hub can be read correctly that tabular data. Thus, this will remove '/blob/' and change starting characters to 'raw.githubusercontent.com'

**Usage**

```r
gitDataUrl(
    urlName,
    replTxt = NULL,
    silent = FALSE,
    debug = FALSE,
    callFrom = NULL
)
```
htmlSpecCharConv

Arguments

urlName (character) main url-address
replTxt (NULL or matrix) adjust/custom-modify search- and replacement items; should be matrix with 2 columns, the 1st column entries will be used as 'search-for' and the 2nd as 'replace by' for each row.
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value
corrected urlName

See Also
sub;

Examples
url1 <- paste0("https://github.com/bigbio/proteomics-metadata-standard/blob/",
"master/annotated-projects/PXD001819/PXD001819.sdrf.tsv")
gitDataUrl(url1)

htmlSpecCharConv

Html special character conversion

Description
Converts 'txt' so that (the most common) special characters (like 'beta','micro','square' etc) will be displayed correctly when used for display in html (eg at mouse-over). Note : The package stringi is required for the conversions (the input will get returned if stringi is not available). Currently only the 16 most common special characters are implemented.

Usage
htmlSpecCharConv(txt, silent = FALSE, callFrom = NULL, debug = FALSE)

Arguments

txt character vector including special characters
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced
debug (logical) additional messages for debugging
Value

This function returns a corrected character vector adopted for HTML display.

See Also


Examples

```r
(x <- stringi::stri_unescape_unicode("\u00b5\u003d\u0061\u0062"))
htmlSpecCharConv(x)
```

keepCommonText

**Extract Longest Common Text Out Of Character Vector**

Description

This function allows recovering the single longest common text-fragments (from center, head or tail) out of character vector `txt`. Only the first of all of the longest solutions will be returned.

Usage

```r
keepCommonText(
  txt,
  minNchar = 1,
  side = "center",
  hiResol = TRUE,
  silent = TRUE,
  callFrom = NULL,
  debug = FALSE
)
```

Arguments

- `txt` character vector to be treated
- `minNchar` (integer) minimum number of characters that must remain
- `side` (character) may be be either 'center', 'any', 'terminal', 'left' or 'right'; only with side='center' or 'any' internal text-segments may be found
- `hiResol` (logical) find best solution, but at much higher computational cost (eg 3x slower, however hiResol=FALSE rather finds anchor which may need to get extended)
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of messages produced
- `debug` (logical) display additional messages for debugging
Details

Please note, that finding common parts between chains of characters is not a completely trivial task. This topic still has ongoing research for the application of sequence-alignments, where chains of characters to be compared get very long. This function uses a k-mer inspired approach. The initial aim with this function was allowing to treat smaller chains of characters (and finding shorter stretches of common text), like eg with column-names.

Important: This function identifies only the first best hit, ie other shared/common character-chains of the same length will not be found!

Using the argument `hiResol=FALSE` it is possible to accelerate the search aprox 3x (with larger character-vectors), however, frequently the very best solution may not be found. This means, that in this case the result should rather be considered a 'seed', allowing check if further extension may improve the result, ie for identifying a (slightly) longer chain of common characters.

With longer vectors and longer character chains this may get demanding on computational resources, the argument `hiResol=FALSE` allows reducing this at the price of missing the best solution. With this argument single common/matching characters will not be searched if all text-elements are longer than 500 characters, an empty character vector will be returned.

When argument `side` is either `left`, `right` or `terminal` only terminal common text may be found (a potentially even longer internal text will be lost). Of course, choosing this option makes searches much faster.

This function does not return the position of the shared/common characters within the text, you may use `gregexpr` or `regexec` to locate them.

Value

This function returns a character vector of length=1, ie only one (normally the longest) common sequence of characters is identified. If nothing is found common/shared an empty character-vector is returned.

See Also

Use `gregexpr` or `regexec` in `grep` for locating the identified common characters in the initial query.

Inverse: Trim redundant text (from either side) to keep only variable part using `trimRedundText`; you may also look for related functions in package `stringr`.

Examples

txt1 <- c("abcd_abc_kjh", "bcd_abc123", "cd_abc_po")
keepCommonText(txt1, side="center") # trim from right

txt2 <- c("ddd_ab","ddd_bcd","ddd_cde")
trimRedundText(txt2, side="left") #
keepCommonText(txt2, side="center") #
levIndex  

Transform (factor) levels into index

Description

This function helps transforming a numeric or character vector into indexes of levels (of its original values). By default indexes are assigned by order of occurance, ie, the first value of x will be get the index of 1. Using the argument byOccurance=FALSE the resultant indexes will follow the sorted values.

Usage

levIndex(
  dat,
  byOccurance = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dat  
  (numeric or character vector or factor) main input

byOccurance  
  (logical) toogle if lowest index should be based on alphabetical order or on order of input

silent  
  (logical) suppress messages

debug  
  (logical) additional messages for debugging

callFrom  
  (character) allow easier tracking of messages produced

Value

matrix with mean values

See Also

rowSds, colSums

Examples

x1 <- letters[rep(c(5,2:3),1:3)]
levIndex(x1)
levIndex(x1, byOccurance=FALSE)
## with factor
fa1 <- factor(letters[rep(c(5,2:3),1:3)], levels=letters[1:6])
levIndex(fa1)
levIndex(fa1, byOccurance=FALSE)
linModelSelect

Test multiple starting levels for linear regression model, select best and plot

Description

The aim of this function is to select the data suiting set of levels of the main input data to construct a linear regression model. In real world measurements one may be confronted to the case of very low level analytes below the detection limit (LOD) and resulting read-outs fluctuate around around a common baseline (instead of NA). With such data it may be preferable to omit the read-outs for the lowest concentrations/levels of analytes if they are spread around a base-line value. This function allows trying to omit all starting levels designed in startLev, then the resulting p-values for the linear regression slopes will be checked and the best p-value chosen. The input may also be a MArrayLM-type object from package limma or from moderTestXgrp or moderTest2grp. In the graphical representation all points associated to levels omitted are shown in light green. For the graphical display additional information can be used: If the dat is list or MArrayLM-type object, the list-elements $raw (according to argument lisNa will be used to display points initially given as NA ad imputed lateron in grey. Logarithmic (ie log-linear) data can be treated by setting argument logExpect=TRUE. Then the levels will be taken as exponent of 2 for the regression, while the original values will be displayed in the figure.

Usage

linModelSelect(
  rowNa,
  dat,
  expect,
  logExpect = FALSE,
  startLev = NULL,
  lisNa = c(raw = "raw", annot = "annot", datImp = "datImp"),
  plotGraph = TRUE,
  tit = NULL,
  pch = c(1, 3),
  cexLeg = 0.95,
  cexSub = 0.85,
  xLab = NULL,
  yLab = NULL,
  cexXAxis = 0.85,
  cexYAxis = 0.9,
  xLabLAns = 1,
  cexLab = 1.1,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
Arguments

rowNa  (character, length=1) rowname for line to be extracted from dat

dat  (matrix, list or MArrayLM-object from limma) main input of which columns
should get re-ordered, may be output from moderTestXgrp or moderTest2grp.

expect  (numeric of character) the expected levels; if character, constant unit-characters
will be stripped away to extract the numeric content

logExpect  (logical) toggle to TRUE if the main data are logarithmic but expect is linear

startLev  (integer) specify all starting levels to test for omitting here (multiple start sites
for modelling linear regression may be specified to finally pick the best model)

lisNa  (character) in case dat is list or MArrayLM-type object, the list-elements with
these names will be used as $raw (for indicating initial NA-values, $datImp (the
main quantitation data to use) and $annot for displaying the corresponding value
from the "Accession"-column.

plotGraph  (logical) display figure

tit  (character) optional custom title

pch  (integer) symbols to use n optional plot; 1st for regular values, 2nd for values
not used in regression

cexLeg  (numeric) size of text in legend

cexSub  (numeric) text-size for line (as subtitle) giving regression details of best linear
model)

xLab  (character) custom x-axis label

yLab  (character) custom y-axis label

cexXAxis  (character) cex-type for size of text for x-axis labels

cexYAxis  (character) cex-type for size of text for y-axis labels

xLabLas  (integer) las-type orientation of x-axis labels (set to 2 for vertical axix-labels)

cexLab  (numeric) cex-type for size of text in x & y axis labels (will be passed to cex.lab in plot())

silent  (logical) suppress messages

debug  (logical) additional messages for debugging

callFrom  (character) allow easier tracking of messages produced

Value

This function returns a list with $coef (coefficients), $name (as/from input rowNa), $startLev the
best starting level)

See Also

moderTestXgrp for single comparisons, order
Examples

## Construct data
li1 <- rep(c(4,3,3:6),each=3) + round(runif(18)/5,2)
names(li1) <- paste0(rep(letters[1:5], each=3), rep(1:3,6))
li2 <- rep(c(6,3:7), each=3) + round(runif(18)/5, 2)
dat2 <- rbind(P1=li1, P2=li2)
exp2 <- rep(c(11:16), each=3)

## Check & plot for linear model
linModelSelect("P2", dat2, expect=exp2)

## Log-Linear data
## Suppose dat2 is result of measures in log2, but exp4 is not
exp4 <- rep(c(3,10,30,100,300,1000), each=3)
linModelSelect("P2", dat2, expect=exp4, logE=FALSE)  # bad
linModelSelect("P2", dat2, expect=exp4, logE=TRUE)

linRegrParamAndPVal  

Fit linear regression, return parameters and p-values

Description

This function fits a linear regression and returns the parameters, including p-values from Anova. Here the vector `y` (scalar response or dependent variable, i.e., the value that should get estimated) will be estimated according to `dep` (explanatory or independent variable). Alternatively, `dep` may me a matrix where 1st column will be used as `dep` and the 2nd column as `y`.

Usage

linRegrParamAndPVal(
  dep,
  y = NULL,
  asVect = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dep  (numeric vector, matrix or data.frame) explanatory or dependent variable, if matrix or data.frame the 1st column will be used, if `y` = NULL the 2nd column will be used as `y`
y  (numeric vector) independent variable (the value that should get estimated based on `dep`)
asVect  (logical) return numeric vector (Intercept, slope, p.intercept, p.slope) or matrix or results
silent (logical) suppress messages
dbg (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value
numeric vector (Intercept, slope, p.intercept, p.slope), or if asVect==TRUE as matrix (p.values in 2nd column)

See Also
lm

Examples
linRegrParamAndPVal(c(5,5.1,8,8.2),gl(2,2))

listBatchReplace (Replacements in list)

Description
listBatchReplace replaces in list lst all entries with value searchValue by replaceBy

Usage
listBatchReplace(
  lst,
  searchValue,
  replaceBy,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments
lst input-list to be used for replacing
searchValue (character, length=1)
replaceBy (character, length=1)
silent (logical) suppress messages
dbg (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value
This function returns a corrected list
See Also

basic replacement sub in grep

Examples

```r
lst1 <- list(aa=1:4, bb=c("abc","efg","abhh","effge"), cc=c("abdc","efg"))
listBatchReplace(lst1, search="efg", repl="EFG", sil=FALSE)
```

```
listGroupsByNames(x, sep = ".", silent = FALSE, debug = FALSE, callFrom = NULL)
```

Arguments

- `x` (list) main input
- `sep` (character) separator (note that typical separators may need to be protected, only automatically added for ".")
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Value

matrix or data.frame

See Also

rbind in cbind

Examples

```r
listGroupsByNames((1:10)/5)
ser1 <- 1:6; names(ser1) <- c("AA","BB","AA.1","CC","AA.b","BB.e")
listGroupsByNames(ser1)
```
Description

`lmSelClu` runs linear regression on data segmented previously (eg by clustering). This function offers various types of (2-coefficient) linear regression on 2 columns of `dat` (matrix with 3rd col named 'clu' or 'cluID', numeric elements for cluster-number). If argument 'clu' is (default) 'max', the column 'clu' will be inspected to take most frequent value of 'clu', otherwise a numeric entry specifying the cluster to extract is expected. Note: this function was initially made for use with results from `diagCheck()` Note: this function lacks means of judging goodness of fit of the regression preformed & means for plotting

Usage

```r
lmSelClu(
  dat,
  useCol = 1:2,
  clu = "max",
  regTy = "lin",
  filt1 = NULL,
  filt2 = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- **dat** matrix or data.frame
- **useCol** (integer or character) specify which 2 columns of 'dat' to use for linear regression
- **clu** (character) name of cluster to be extracted and treated
- **regTy** (character) change type used for linear regression: 'lin' for 1st col ~ 2nd col, 'res' for residue ~ 2nd col, 'norRes' for residue/2nd col ~2nd col or 'sqNorRes','inv' for 1st col ~ 1/(2nd col), 'invRes' for residue ~ 1/(2nd col)
- **filt1** (logical or numerical) filter criteria for 1st of 'useCol', if numeric then select all lines of dat less than max of filt1
- **filt2** (logical or numerical) filter criteria for 2nd of 'useCol', if numeric then select all lines of dat less than max of filt2
- **silent** (logical) suppress messages
- **debug** (logical) additional messages for debugging
- **callFrom** (character) allow easier tracking of messages produced

Value

lm object (or NULL if no data left)
lrbind

See Also

lm

Examples

set.seed(2016); ran1 <- runif(220)
mat1 <- round(rbind(matrix(c(1:100+ran1[1:100],rep(1,50)),ncol=3),
                   matrix(c(1:60,68:9+ran1[101:160],rep(2,60)),nc=3)),1)
colnames(mat1) <- c("a","BB","clu")
lmSelClu(mat1)
plot(mat1[which(mat1[,3]=="2"),1:2],col=grey(0.6))
abline(lmSelClu(mat1),lty=2,lwd=2)
#
mat2 <- round(rbind(matrix(c(1:100+ran1[1:100],rep(1,50)),ncol=3),
                   matrix(c(1:60,(2:61+ran1[101:160])**2,rep(2,60)),nc=3)),1)
colnames(mat2) <- c("a","BB","clu")
(reg2 <- lmSelClu(mat2,regTy="sqNor"))
plot(function(x) coef(reg2)[2] + (coef(reg2)[2]*x^2),xlim=c(1,70))
points(mat2[which(mat2[,3]=="2"),1:2],col=2)

lrbind

rbind on lists

Description

lrbind-like function to append list-elements containing matrixes (or data.frames) and return one long table. All list-elements must have same number of columns (and same types of classes in case of data.frames. Simple vectors (as list-elements) will be considered as sigle lines for attaching.

Usage

lrbind(lst, silent = FALSE, debug = FALSE, callFrom = NULL)

Arguments

lst

(list, composed of multiple matrix or data.frames or simple vectors) main input
(each list-element should have same number of columns, numeric vectors will be converted to number of columns of other columns/elements)
silent

(logical) suppress messages
debug

(logical) additional messages for debugging
callFrom

(character) allow easier tracking of messages produced

Value

This function returns (depending on input) a matrix or data.frame
makeMAList

Make MA-List object

Description

makeMAList extracts sets of data-pairs (like R & G series) and makes MA objects as MA-List object (eg for ratio oriented analysis). The grouping of columns as sets of replicate-measurements is done according to argument 'MAfac'. The output is fully compatible to functions of package limma (Bioconductor).

Usage

makeMAList(
  mat,
  MAfac,
  useF = c("R", "G"),
  isLog = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

mat main input matrix
MAfac (factor) factor orgnaizing columns of `mat` (if useF contains the default 'R' and 'G', they should also be part of MAfac)
useF (character) two specific factor-leves of MAfac that will be used/extracted
isLog (logical) tell if data is already log2 (will be considered when computing M and A values)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Details

This function requires Bioconductor package limma being installed.
**Value**

limma-type "MAList" containing M and A values

**See Also**

test2factLimma, for creating RG-lists within limma: MA.RG in normalizeWithinArrays

**Examples**

```r
set.seed(2017); t4 <- matrix(round(runif(40,1,9),2), ncol=4,
       dimnames=list(letters[c(1:5,3:4,6:4)], c("AA1","BB1","AA2","BB2")))
makeMAList(t4, gl(2,2,labels=c("R","G")))
```

---

**Description**

makeNRedMatr takes matrix or data.frame 'dat' to summarize redundant lines (column argument iniID) along method specified in summarizeRedAs to treat all lines with redundant iniID by same approach (ie for all columns the line where specified column is at eg max = 'maxOfRef' ). If no name given, the function will take the last numeric (factors may be used - they will be read as levels).

**Usage**

```r
makeNRedMatr(
   dat,
   summarizeRedAs,
   iniID = "iniID",
   retDataFrame = TRUE,
   silent = FALSE,
   debug = FALSE,
   callFrom = NULL
)
```

**Arguments**

- **dat** (matrix or data.frame) main input for making non-redundant
- **summarizeRedAs** (character) summarization method(s), typical choices 'median', 'mean', 'min' or 'maxOfRef', 'maxAbsOfRef' for summarizing according to 1 specified column, may be single method for all or different method for each column (besides col 'iniID') or special method looking at column (if found, first of special methods used, everything else not considered).
- **iniID** (character) column-name used as initial ID (default="iniID")
- **retDataFrame** (logical) if TRUE, check if text-columns may be converted to data.frame with numeric
matchMatrixLinesToRef

Match All Lines of Matrix To Reference Note

Description

This function allows adjusting the order of lines of a matrix mat to a reference character-vector ref, even when initial direct matching of character-strings using match is not possible/successful. In this case, various variants of using grep will be used to see if unambiguous matching is possible of characteristic parts of the text. All columns of mat will be tested an the column giving the bes

Usage

matchMatrixLinesToRef(
  mat,
  ref,
  addRef = TRUE,
  inclInfo = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
Arguments

- **mat** (matrix or data.frame) main input, all columns of of mat will be tested for (partial) matching of ref
- **ref** (character, length must match ) reference for trying to match each of the columns of mat
- **addRef** (logical), if TRUE the content of ref will be added to mat as additional column
- **inclInfo** (logical) allows returning list with new matrix and additional information
- **silent** (logical) suppress messages
- **debug** (logical) additional messages for debugging
- **callFrom** (character) allow easier tracking of messages produced

Details

This function tests all columns of mat to find perfect matching results to the reference ref. In case of multiple results the In case no direct matching is possible, grep will be used to find the best partial matching. The order of the rows of input mat will be adjusted according to the matching results.

If addRef=TRUE, the reference will be included as additional column to the results, too.

Value

This function returns the input matrix in an adjusted order (plus an optional additional column showing the reference) or if inclInfo=TRUE a list with $mat (adjusted matrix), $byColumn, $newOrder and $method; the reference can bee added as additional last column if addRef=TRUE

See Also

- match, grep, trimRedundText, replicateStructure

Examples

```r
## Note : columns b and e allow non-ambigous match, not all elements of e are present in a
mat0 <- cbind(a=c("mvvk","axxd","bxkd","vv"),b=c("iwwy","iyyu","kvvh","gxx"),
c=rep(9,4),
d=c("hgf","hgf","vxc","nvmn"), e=c("vv_","ww_","xx_","yy_"))
matchMatrixLinesToRef(mat0[,1:4], ref=mat0[,5])
matchMatrixLinesToRef(mat0[,1:4], ref=mat0[1:3,5], inclInfo=TRUE)
matchMatrixLinesToRef(mat0[,-2], ref=mat0[,2], inclInfo=TRUE)  # needs 'reverse grep'
```
Value Matching with optional reversing of sub-parts of non-matching elements

Description

This function provides a variant to `match`, where initially non-matching elements of `x` will be tested by decomposing non-matching elements, reversing the parts in front and after the separator `sep` and re-matching. If separator `sep` does not occur, a warning will be issued, if it occurs more than once, the parts before and after the first separator will be used and a warning issued.

Usage

```r
matchNamesWithReverseParts(
  x,
  y,
  sep = "-",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- `x` (character) first vector for match
- `y` (character) second vector for match
- `sep` (character) separator between elements
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Value

index for matching (integer) `x` to `y`

See Also

- `match`

Examples

```r
tx1 <- c("a-b","a-c","d-a","d-b","b-c","d-c")
tmp <- triCoord(4)
tx2 <- paste(letters[tmp[,1]],letters[tmp[,2]],sep="-")
## Some matches won't be found, since 'a-d' got reversed to 'd-a', etc...

match(tx1,tx1)
matchNamesWithReverseParts(tx1,tx2)
```
**matchSampToPairw**  
*Match names to concatenated pairs of names*

**Description**

The column-names of multiple pairwise testing contain the names of the initial groups/conditions tested, plus there is a separator (eg `'-'` in `moderTestXgrp`). Thus function allows to map back which groups/conditions were used by returning the index of the respective groups used in pair-wise sets.

**Usage**

```r
matchSampToPairw(
  grpNa,
  pairwNa,
  sep = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `grpNa` (character) the names of the groups of replicates (i.e., conditions) used to test
- `pairwNa` (character) the names of pairwise-testing (i.e., 'concatenated' sampNa)
- `sep` (character) if not NULL the characters given will be used via `stringsplit`
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

**Details**

There are two modes of operation: 1) Argument `sep` is set to `NULL`: The names of initial groups/conditions (`grpNa`) will be tested for exact pattern matching either at beginning or at end of pair-wise names (`pairwNa`). This approach has the advantage that it does not need to be known what character(s) were used as separator (or they may change), but the disadvantage that in case the perfect `grpNa` was not given, the longest best match of `grpNa` will be returned.

2) The separator `sep` is given and exact matches at both sides will be searched. However, if the character(s) from `sep` do appear inside `grpNa` no matches will be found.

If some `grpNa` are not found in `pairwNa` this will be marked as NA.

**Value**

matrix of 2 columns with indices of sampNa with pairwNa as rows
See Also

(for running multiple pair-wise test) `moderTestXgrp, grep, strsplit`

Examples

```r
pairwNa1 <- c("abc-efg","abc-hij","efg-hij")
grpNa1 <- c("hij","abc","abcc","efg","klm")
matchSampToPairw(grpNa1, pairwNa1)

pairwNa2 <- c("abc-efg","abcc-hij","abc-hij","abc-hijj","zz-zz","efg-hij")
macthSampToPairw(grpNa1, pairwNa2)
```

---

**matr2list**

Transform columns of matrix to list of vectors

Description

convert matrix to list of vectors: each column of ‘mat’ as vector of list

Usage

```r
matr2list(mat, concSym = ".", silent = FALSE, debug = TRUE, callFrom = NULL)
```

Arguments

- `mat` (matrix) main input
- `concSym` (character) symbol for concatenating: concatenation of named vectors in list names as colname(s)+'concSym'+rowname
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Value

matrix or array (1st dim is intraplate-position, 2nd .. plate-group/type, 3rd .. channels)

See Also

`convToNum`

Examples

```r
mat1 <- matrix(1:12,ncol=3,dimnames=list(letters[1:4],LETTERS[1:3]))
mat2 <- matrix(LETTERS[11:22],ncol=3,dimnames=list(letters[1:4],LETTERS[1:3]))
matr2list(mat1); matr2list(mat2)
```
**mergeMatrices**  
*Merge Multiple Matrices*

**Description**

This function allows merging of multiple matrix-like objects. The matrix-rownames will be used to align common elements, either be returning all common elements `mode='intersect'` or containing all elements `mode='union'` (the result may contains additional NAs).

**Usage**

```r
mergeMatrices(
  ..., 
  mode = "intersect", 
  useColumn = 1, 
  na.rm = TRUE, 
  extrRowNames = FALSE, 
  silent = FALSE, 
  debug = FALSE, 
  callFrom = NULL
)
```

**Arguments**

- `...` (matrix or data.frame) multiple matrix or data.frame objects may be entered
- `mode` (character) allows choosing restricting to all common elements (mode='intersect') or union (mode='union')
- `useColumn` (integer, character or list) the column(s) to consider, may be 'all' to use all, integer to select specific indexes or list of indexes or colnames for cutom-selection per matrix
- `na.rm` (logical) suppress NAs
- `extrRowNames` (logical) decide whether columns with all values different (ie no replicates or max divergency) should be excluded
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

**Details**

Custom column-names can be given by entering matrices like named arguments (see examples below). The choice of columns tu use may be adopted to each matrix entered, in this case the argument `useColumn` may be a list with matrix-names to use or a list of indexes (see examples below).

Note, that matrices may contain repeated rownames (see examples, `mat3`). In this case only the first of repeated rownames will be considered (and lines of repeated names ignored).
mergeMatrixList

Value

This function returns a matrix containing all selected columns of the input matrices to fuse.

See Also

merge, mergeMatrixList

Examples

```r
mat1 <- matrix(11:18, ncol=2, dimnames=list(letters[3:6], LETTERS[1:2]))
mat2 <- matrix(21:28, ncol=2, dimnames=list(letters[2:5], LETTERS[3:4]))
mat3 <- matrix(31:38, ncol=2, dimnames=list(letters[c(1,3:4,3)], LETTERS[4:5]))

mergeMatrices(mat1, mat2)
mergeMatrices(mat1, mat2, mat3, mode="union", useCol=2)
## custom names for matrix-origin
mergeMatrices(m1=mat1, m2=mat2, mat3, mode="union", useCol=2)
## flexible/custom selection of columns
mergeMatrices(m1=mat1, m2=mat2, mat3, mode="union", useCol=list(1,1:2,2))
```

mergeMatrixList

Merge Multiple Matrices from List

Description

This function allows merging of multiple matrix-like objects from an initial list. The matrix-rownames will be used to align common elements, either be returning all common elements mode='intersect' or containing all elements mode='union' (the result may contains additional NAs).

Usage

```r
mergeMatrixList(
  matLst,
  mode = "intersect",
  useColumn = 1,
  na.rm = TRUE,
  extrRowNames = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- **matLst** (list containing matrices or data.frames) main input (multiple matrix or data.frame objects)
- **mode** (character) allows choosing restricting to all common elements (mode='intersect') or union (mode='union')
mergeSelCol

useColumn (integer, character or list) the column(s) to consider, may be 'all' to use all, integer to select specific indexes or list of indexes or colnames for custom-selection per matrix

na.rm (logical) suppress NAs

extrRowNames (logical) decide whether columns with all values different (ie no replicates or max divergency) should be excluded

silent (logical) suppress messages

debug (logical) additional messages for debugging

callFrom (character) allow easier tracking of messages produced

Details

Custom column-names can be given by entering matrices like named arguments (see examples below). The choice of columns to use may be adopted to each matrix entered, in this case the argument useColumn may be a list with matrix-names to use or a list of indexes (see examples below).

Note, that matrices may contain repeated rownames (see examples, mat3). In this case only the first of repeated rownames will be considered (and lines of repeated names ignored).

Value

This function returns a matrix containing all selected columns of the input matrices to fuse

See Also

merge, mergeMatrices for separate entries

Examples

mat1 <- matrix(11:18, ncol=2, dimnames=list(letters[3:6],LETTERS[1:2]))
mat2 <- matrix(21:28, ncol=2, dimnames=list(letters[2:5],LETTERS[3:4]))
mat3 <- matrix(31:38, ncol=2, dimnames=list(letters[c(1,3:4,3)],LETTERS[4:5]))

mergeMatrixList(list(mat1, mat2))

mergeMatrixList(list(m1=mat1, m2=mat2, mat3), mode="union", useCol=2)
mergeSelCol

Usage

mergeSelCol(
    dat1,
    dat2,
    selCols,
    supCols2 = NULL,
    byC = NULL,
    useAll = FALSE,
    setRownames = TRUE,
    newSuff = c(".x", ".y"),
    silent = FALSE,
    debug = FALSE,
    callFrom = NULL
)

Arguments

dat1 matrix or data.frame for fusing
dat2 matrix or data.frame for fusing
selCols will be used to define columns to be used; optionally may be different for 'dat2'
: define in 'supCols2'
supCols2 if additional column-names should be extracted from dat2
byC (character) 'by' value used in merge
useAll (logical) use all lines (will produce NAs when given identifier not found in 2nd group of data)
setRownames (logical) if TRUE, will use values of col used as 'by' as rownames instead of showing as add'l col in output
newSuff (character) prefix (argument 'suffixes' in merge)
silent (logical) suppress messages
debug (logical) display additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a data.frame containing the merged columns

See Also

merge, merge 3 data.frames using mergeSelCol3

Examples

mat1 <- matrix(c(1:7,letters[1:7],11:17), ncol=3, dimnames=list(LETTERS[1:7],c("x1","x2","x3")))
mat2 <- matrix(c(1:6,c("b","e","f","g","k"), 31:36),
              ncol=3, dimnames=list(LETTERS[11:16],c("y1","x2","x3")))
mergeSelCol(mat1, mat2, selC=c("x2","x3"))
mergeSelCol3

Description

successive merge of selected columns out of 3 matrix or data.frames. `selCols` will be used to define columns to be used; optionally may be different for `dat2`: define in `supCols2`. Output-cols will get additions specified in newSuff (default `.x` and `.y`)

Usage

```r
mergeSelCol3(
  dat1,
  dat2,
  dat3,
  selCols,
  supCols2 = NULL,
  supCols3 = NULL,
  byC = NULL,
  useAll = FALSE,
  setRownames = TRUE,
  newSuff = c(".x", ".y", "z"),
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

dat1: matrix or data.frame for fusing
dat2: matrix or data.frame for fusing
dat3: matrix or data.frame for fusing

selCols: will be used to define columns to be used; optionally may be different for `dat2`: define in `supCols2`
supCols2: if additional column-names should be extracted form `dat2`
supCols3: if additional column-names should be extracted form `dat3`

byC: (character) `by` value used in `merge`

useAll: (logical) use all lines (will produce NAs when given identifier not found un 2nd group of data)

setRownames: if TRUE, will use values of col used as `by` as rownames instead of showing as add`l col in output

newSuff: (character) prefix (argument `suffixes` in `merge`)

silent: (logical) suppress messages
debug: (logical) display additional messages for debugging
callFrom: (character) allow easier tracking of messages produced
mergeVectors

Value
This function returns a data.frame containing the merged columns

See Also
merge, mergeSelCol

Examples
mat1 <- matrix(c(1:7,letters[1:7],11:17),ncol=3,dimnames=list(LETTERS[1:7],c("x1","x2","x3")))
mat2 <- matrix(c(1:6,c("b","a","e","f","g","k"),31:36), ncol=3,
               dimnames=list(LETTERS[11:16],c("y1","x2","x3")))
mat3 <- matrix(c(1:6,c("c","a","e","b","g","k"),51:56), ncol=3,
               dimnames=list(LETTERS[11:16],c("z1","x2","x3")))
mergeSelCol3(mat1, mat2, mat3, selC=c("x2","x3"))

mergeVectors

Merge Named Vectors

Description
This function allows merging for multiple simple named vectors (each element needs to be named).
Basically, all elements carrying the same name across different input-vectors will be aligned in the
same column of the output (input-vectors appear as lines). If vectors are not given using a name (see
first example below), they will be names `x.1` etc (see argument namePrefix).

Usage
mergeVectors(
  ..., 
  namePrefix = "x.",
  NAto0 = FALSE,
  callFrom = NULL,
  silent = FALSE
)

Arguments

... all vectors that need to be merged
namePrefix (character) prefix to numer used when vectors are not given with explicit names
  (second example)
NAto0 (logical) optional replacement of NAs by 0
callFrom (character) allow easier tracking of message produced
silent (logical) suppres messages
Details

Note: The arguments 'namePrefix', 'NAto0', 'callFrom' and 'silent' must be given with full
name to be recognized as such (and not get considered as vector for merging).

Value

matrix of merged values

See Also

merge (for two data.frames)

Examples

x1 <- c(a=1, b=11, c=21)
x2 <- c(b=12, c=22, a=2)
x3 <- c(a=3, d=43)
mergeVectors(vect1=x1, vect2=x2, vect3=x3)
x4 <- 41:44 # no names - not conform for merging
mergeVectors(x1, x2, x3, x4)

mergeW2

Extended version of merge for multiple objects (even without row-
names)

Description

mergeW2 provides flexible merging out of 'MArrayLM'-object (if found, won’t consider any other
input-data) or of separate vectors or matrixes. The main idea was to have somthing not adding add’l
lines as merge might do, but to stay within the frame of the 1st argument given, even when IDs
are repeated, so the output follows the order of the 1st argument, non-redundant IDs are created
(orig IDs as new column). If no 'MArrayLM'-object found: try to combine all elements of input
'...', input-names must match predefined variants 'chInp'. IDs given in 1st argument and not found
in later arguments will be displayed as NA in the output matrix of data.frame. Note: (non-data)
arguments must be given with full name (so far no lazy evaluation, may conflict with names in
'inputNamesLst'). Note: special characters in colnames bound to give trouble. Note: when no
names given, mergeW2 will presume order of elements (names) from 'inputNamesLst'. PROBLEM :
error after xxMerg3 when several entries have matching (row)names but some entries match only
partially (what to do : replace with NAs ??)

Usage

mergeW2(
  ...,  
  nonRedundID = TRUE,
  convertDF = TRUE,
  selMerg = TRUE,
  inputNamesLst = NULL,
mergeW2

noMatchPursue = TRUE,
standColNa = FALSE,
lastOfMultCols = c("p.value", "Lfdr"),
duplTxtSep = ".",
silent = FALSE,
depbug = FALSE,
callFrom = NULL)

Arguments

... all data (vectors, matrixes or data.frames) intendes for merge
nonRedundID (logical) if TRUE, allways add 1st column with non-redundant IDs (add anyway if non-redundant IDs found)
convertDF (logical) allows converting output in data.frame, add new heading col with non-red rownames & check which cols should be numeric
selMerg (logical) if FALSE toggle to classic merge() (will give more rows in output in case of redundant names
inputNamesLst (list) named list with character vectors (should be unique), search these names in input for extracting/merging elements use for "lazy matching" when checking names of input, default: 7 groups ('Mvalue', 'Avalue', 'p.value', 'mouseInfo', 'Lfdr', 'link', 'filt') with common short versions
noMatchPursue (logical) allows using entries where 0 names match (just as if no names given)
standColNa (logical) if TRUE return standard colnames as defined in 'inputNamesLst' (ie 'chInp'), otherwise colnames as initially provided
lastOfMultCols may specify input groups where only last col will be used/extracted
duplTxtSep (character) separator for counting/denomiating multiple occurances of same name
silent (logical) suppress messages
depbug (logical) for bug-tracking: more/enhanced messages and intermediate objects written in global name-space
callFrom (character) allows easier tracking of message(s) produced

Value

matrix or data.frame of fused data

See Also

merge

Examples

t1 <- 1:10; names(t1) <- letters[c(1:7,3:8)]
t2 <- 20:11; names(t2) <- letters[c(1:7,3:4,8)]
t3 <- 101:110; names(t3) <- letters[c(11:20)]
t4 <- matrix(100:81,ncol=2,dimnames=list(letters[1:10],c("co1","co2"))
minDiff

Minimum distance/difference between values

Description

This function aims to find the min distance (ie closest point) to any other \( x \) (numeric value), ie intra 'x' and returns matrix with 'index','value','dif','ppm','ncur','nbest','best'. At equal distance to lower & upper neighbour point, the upper (following) point is chosen (as single best). In case of multiple ex-aequo distance returns 1st of multiple, may be different at various repeats.

Usage

```r
minDiff(
  x,
  digSig = 3,
  ppm = TRUE,
  initOrder = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- **x**: (numeric) vector to search minimum difference
- **digSig**: number of significant digits, used for ratio or ppm column
- **ppm**: (logical) display distance as ppm \((1e6 \times \text{diff/refValue})\), ie normalized difference eg as used in mass spectrometry), otherwise the ratio is given as \(\text{value(from 'x')/closestValue (from 'x')}\)
- **initOrder**: (logical) return matrix so that 'x' matches exactly 2nd col of output
- **silent**: (logical) suppress messages
- **debug**: (logical) additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Value

This function returns a matrix

See Also

diff
Examples

```r
set.seed(2017); aa <- 100*c(0.1 + round(runif(20),2),0.53,0.53)
minDiff(aa);
minDiff(aa,initO=TRUE,ppm=FALSE); .minDif(unique(aa))
```

---

**moderTest2grp**  
*Moderated pair-wise t-test from limma*

**Description**

Runs moderated t-test from package 'limma' on each line of data. Note: This function requires the package limma from bioconductor. The limma contrast-matrix has to be read by column, the lines in the contrast-matrix containing ’+1’ will be compared to the ’-1’ lines, eg grpA-grpB. Local false discovery rates (lfdr) estimations will be made using the CRAN-package fdrtool (if available).

**Usage**

```r
moderTest2grp(
  dat,
  grp,
  limmaOutput = TRUE,
  addResults = c("lfdr", "FDR", "Mval", "means"),
  testOrientation = "+",
  silent = FALSE,
  callFrom = NULL
)
```

**Arguments**

- **dat**  
  matrix or data.frame with rows for multiple (independent) tests, use ONLY with 2 groups; assumed as log2-data

- **grp**  
  (factor) describes column-relationship of 'dat' (1st factor is considered as reference -> orientation of M-values !)

- **limmaOutput**  
  (logical) return full (or extended) MArrayLM-object from limma or 'FALSE' for only the (uncorrected) p.values

- **addResults**  
  (character) types of results to add besides basic limma-output, data are assumed to be log2! (eg "lfdr" using fdrtool-package, "FDR" or "BH" for BH-FDR, "BY" for BY-FDR, "bonferroni" for Bonferroni-correction, "qValue" for lfdr by qvalue, "Mval", "means" or "nonMod" for non-moderated test and he equivaent all (other) multiple testing corrections chosen here)

- **testOrientation**  
  (character) for one-sided test (">","greater" or "<","less"), NOTE : 2nd grp is considered control/reference, ’<’ will identify grp1 < grp2

- **silent**  
  (logical) suppress messages

- **callFrom**  
  (character) allow easier tracking of message(s) produced
moderTestXgrp

Value

This function returns a limma-type object of class MArrayLM

See Also

lmFit and the eBayes-family of functions in package limma, p.adjust

Examples

```
set.seed(2017); t8 <- matrix(round(rnorm(1600,10,0.4),2), ncol=8,
   dimnames=list(paste("l",1:200),c("AA1","BB1","CC1","DD1","AA2","BB2","CC2","DD2")))
t8[3:6,1:2] <- t8[3:6,1:2]+3  # augment lines 3:6 for AA1&BB1
  t8[5:8,5:6] <- t8[5:8,5:6]+3  # augment lines 5:8 for AA2&BB2 (c,d,g.h should be found)
t4 <- log2(t8[,1:4]/t8[,5:8])
## Two-sided testing
fit4 <- moderTest2grp(t4,gl(2,2))
# If you have limma installed we can now see further
if("list" %in% mode(fit4)) limma::topTable(fit4, coef=1, n=5)  # effect for 3,4,7,8

## One-sided testing
fit4in <- moderTest2grp(t4,gl(2,2),testO="<")
# If you have limma installed we can now see further
if("list" %in% mode(fit4in)) limma::topTable(fit4in, coef=1, n=5)
```

moderTestXgrp

Multiple moderated pair-wise t-tests from limma

Description

Runs all pair-wise combinations of moderated t-tests from package 'limma' on each line of data against 1st group from 'grp'. Note: This function requires the package limma from bioconductor. The limma contrast-matrix has to be read by column, the lines in the contrast-matrix containing '+1' will be compared to the '-1' lines, eg grpA-grpB.

Usage

```
moderTestXgrp(
  dat,
  grp,
  limmaOutput = TRUE,
  addResults = c("1fdr", "FDR", "Mval", "means"),
  testOrientation = "=",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```
Arguments

dat  matrix or data.frame with rows for multiple (independent) tests, use ONLY with 2 groups; assumed as log2-data !!!
grp  (factor) describes column-relationship of 'dat' (1st factor is considered as reference -> orientation of M-values !!)
limmaOutput  (logical) return full (or extended) MArrayLM-object from limma or 'FALSE' for only the (uncorrected) p.values
addResults  (character) types of results to add besides basic limma-output, data are assumed to be log2 ! (eg "lfdr" using fdrtool-package, "FDR" or "BH" for BH-FDR, "BY" for BY-FDR, "bonferroni" for Bonferroni-correction, "qValue" for lfdr by qvalue, "Mval", "means" or "nonMod" for non-moderated test and he equivaent all (other) multiple testing corrections chosen here)
testOrientation  (character) for one-sided test (">","greater" or "<","less"), NOTE : 2nd grp is considered control/reference, '<' will identify grp1 < grp2
silent  (logical) suppress messages
debug  (logical) additional messages for debugging
callFrom  (character) allow easier tracking of message(s) produced

Value

This function returns a limma-type MA-object (list)

See Also

moderTest2grp for single comparisons, lmFit and the eBayes-family of functions in package limma

Examples

grp <- factor(rep(LETTERS[3:1,4], c(2,3,3)))
set.seed(2017); t8 <- matrix(round(rnorm(208*8,10,0.4),2), ncol=8,
dimnames=list(paste(letters[], rep(1:8, each=26), sep=""), paste(grp,c(1:2,3:3), sep="")))
t8[3:6,1:2] <- t8[3:6,1:2] +3  # augment lines 3:6 (c-f)
t8[5:8, c(1:2,6:8)] <- t8[5:8, c(1:2,6:8)] -1.5 # lower lines
t8[6:7,3:5] <- t8[6:7,3:5] +2.2 # augment lines
## expect to find C/A in c,d,g, (h)
## expect to find C/D in c,d,e,f
## expect to find A/D in f,g,(h)
test8 <- moderTestXgrp(t8, grp)
# If you have limma installed we can now see further
if("list" %in% mode(test8)) head(test8$p.value, n=8)
multiCharReplace  Multiple replacement of entire character elements in simple vector, matrix or data.frame

Description
This function allows multiple types of replacements of entire character elements in simple vector, matrix or data.frame. In addition, the result may be optionally directly transformed to logical or numeric.

Usage
multiCharReplace(
  mat,
  repl,
  convTo = NULL,
  silent = FALSE,
  debug = TRUE,
  callFrom = NULL
)

Arguments
mat  (character vector, matrix or data.frame) main data
repl (matrix or list) tells what to replace by what: If matrix the 1st oolumn will be considered as 'old' and the 2nd as 'replaceBy'; if named list, the names of the list-elements will be considered as 'replaceBy'
convTo (character) optional conversion of content to 'numeric' or 'logical'
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value
This function returns an object of same dimension as input (with replaced content)

See Also
grep

Examples
x1 <- c("ab","bc","cd","efg","ghj")
multiCharReplace(x1, cbind(old=c("bc","efg"), new=c("BBCC","EF")))
x2 <- c("High","n/a","High","High","Low")
multiCharReplace(x2, cbind(old=c("n/a", "Low", "High"), new=c(NA, FALSE, TRUE)), convTo="logical")

# works also to replace numeric content:
x3 <- matrix(11:16, ncol=2)
multiCharReplace(x3, cbind(12:13, 112:113))

---

**multiMatch**  
*Simple Multi-to-Multi Matching of (Concatenated) Terms*

**Description**

This function allows convenient matching of multi-to-multi relationships between two objects/vectors. It was designed for finding common elements in multiple to multiple matching situations (eg when comparing c("aa; bb", "cc") to c("bb; ab", "dd"), ie to find 'bb' as matching between both objects).

**Usage**

```r
multiMatch(
  x,
  y,
  sep = "; ",
  sep2 = NULL,
  method = "byX",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `x` (vector or list) first object to compare; if vector, the (partially) concatenated identifiers (will be split using separator `sep`), or list of items to be matched (ie already split)
- `y` (vector or list) second object to compare; if vector, the (partially) concatenated identifiers (will be split using separator `sep`), or list of items to be matched (ie already split)
- `sep` (character, length=1) separator used to split concatenated identifiers (if `x` or `y` is vector)
- `sep2` (character, length=1) optional separator used when `method="matched"` to concatenate all indexes of `y` for column `y.allInd`
- `method` (character) mode of operation: 'asIndex' to return index of `y` (those who have matches) with names of `x` (which `x` are the corresponding match)
- `silent` (logical) suppress messages
- `debug` (logical) display additional messages for debugging
- `callFrom` (character) allow easier tracking of message(s) produced
**Details**

method = 'byX' .. returns data.frame with view oriented towards entries of x: character column x for entire content of x; integer column x.Ind for index of x; character column TagBest for most frequent matching isolated tag/ID; integer column y.IndBest index of most frequent matching y; character column y.IndAll index for all y matching any of the tags; character column y.Match for entire content of best matching y; character column y.Adj for y adjusted to best matching y for easier subsequent perfect matching.

method=c("byX","filter") .. combinded argument to keep only lines with any matches

method='byTag' .. returns matrix (of integers) from view of isolated tags from x (a separate line for each tag from x matching to y);

method=c("byTag","filter") ..if combined as arguments, this will return a data.frame for all unique tags with any matches between x and y, with additional columns x.AllInd for all matching x-indexes, y.IndBest best matching y index; x.n for number of different x containing this tag; y.AllInd for all matching y-indexes

method='adjustXtoY' .. returns vector with x adjusted to y, ie those elements of x matching are replace by the exact corresponding term of y.

method=NULL .. If no term matching the options shown above is given, another version of 'asIndex' is returned, but indexes to y _after_ spliting by sep. Again, this method can be filtered by using method="filter" to focus on the best matches to x.

**Value**

matrix, data.frame or list with matching results depending on method chosen

**See Also**

match; strsplit

**Examples**

```r
aa <- c("m","k","j; aa","m; aa; bb; o; ee","n; dd; cc","aa","cc")
bb <- c("dd; r","aa","ee; bb; q; cc","p; cc")
(match1 <- multiMatch(aa, bb, method=NULL))  # match bb to aa
(match2 <- multiMatch(aa, bb, method="byX"))  # match bb to aa
(match3 <- multiMatch(aa, bb, method="byTag"))  # match bb to aa
(match4 <- multiMatch(aa, bb, method=c("byTag","filter")))  # match bb to aa
```

**Description**

naOmit removes NAs from input vector. This function has no slot for removed elements while na.omit does so. Resulting objects from naOmit are smaller in size and subsequent execution (on large vectors) is faster (in particular if many NAs get encountered). Note : Behaves differently to na.omit with input other than plain vectors. Will not work with data.frames!
naOmit(x)

Arguments
x (vector or matrix) input

Value
vector without NAs (matrix input will be transformed to vector). Returns NULL if input consists only of NAs.

See Also
na.fail, na.omit

Examples
aA <- c(11:13,NA,10,NA);
naOmit(aA)

nFragments(protSeq, cutAt, sizeRa)

Arguments
protSeq (character) text to be cut
cutAt (character) position to cut
sizeRa (numeric,length=2) min and max size to consider

Value
numeric vector with number of fragments for each entry 'protSeq' (names are 'protSeq')

See Also
cutAtMultSites, simple version {nFragments0} (no size-range)
**nFragments0**

**Examples**

```r
tmp <- "MSVSREDSCELVYTERIIAVSPSTANEENFRSNLREVAQMLSKKHGNYLLFNLSERRPDITKLHAKVLEFGWPDHLP"
nFragments(c(tmp,"ojioRij"),c("R","K"),c(4,31))
```

---

**Description**

*nFragments0* tells the number of fragments/entry when cutting after 'cutAt'

**Usage**

```r
nFragments0(protSeq, cutAt)
```

**Arguments**

- **protSeq** (character) text to be cut
- **cutAt** (integer) position to cut

**Value**

numeric vector with number of fragments for each entry 'protSeq' (names are 'protSeq')

**See Also**

more elaborate {nFragments}; cutAtMultSites

**Examples**

```r
tmp <- "MSVSRTMEDCLELVYTERIIAVSPSTANEENFRSNLREVAQMLSKKHGNYLLFNLSERRPDITKLHAKVLEFGWPDHLP"
nFragments0(c(tmp,"ojioRij"),c("R","K"))
```

---

**nNonNumChar**

**Description**

*nNonNumChar* counts number of non-numeric characters. Made for positive non-scientific values (eg won’t count neg-sign, neither Euro comma ‘,‘)

**Usage**

```r
nNonNumChar(txt)
```
nonAmbiguousMat

Arguments

- `txt` character vector to be treated

Value

This function returns a numeric vector with number of non-numeric characters (i.e., not '.' or 0-9).

See Also

nchar

Examples

nNonNumChar("a1b "); sapply(c("aa","12ab","a1b2","12","0.5"), nNonNumChar)

nonAmbiguousMat

Transform matrix to non-ambiguous matrix (in respect to given column)

Description

`nonAmbiguousMat` makes values of matrix `mat` in col `byCol` unique.

Usage

```r
nonAmbiguousMat(
  mat,
  byCol,
  uniqOnly = FALSE,
  asList = FALSE,
  nameMod = "amb_",
  callFrom = NULL
)
```

Arguments

- `mat` numeric or character matrix (or data.frame), column specified by `byCol` must be used as numeric, 1st column of `mat` will be considered like index & used for adding prefix `nameMod` (unless `byCol`=1, then 2nd col will be used).
- `byCol` (character or integer-index) column by which ambiguity will be tested.
- `uniqOnly` (logical) if `TRUE` return unique only, if `FALSE` return unique and single representative of non-unique values (with ” added to name), selection of representative of repeated: first (of sorted) or middle if >2 instances.
- `asList` (logical) return result as list.
- `nameMod` (character) prefix added to 1st column of `mat` (expect ‘by’) for indicating non-unique/ambiguous values.
- `callFrom` (character) allow easier tracking of message(s) produced.
nonAmbiguousNum

Value

sorted non-ambiguous numeric vector (or list if 'asList'=TRUE and 'uniqOnly'=FALSE)

See Also

for non-numeric use firstOfRepeated - but 1000x much slower !; get1stOfRepeatedByCol

Examples

set.seed(2017); mat2 <- matrix(c(1:100,round(rnorm(200),2)),ncol=3, dimnames=list(1:100,LETTERS[1:3]));
head(mat2U <- nonAmbiguousMat(mat2,by="B",na="_",uniqO=FALSE),n=15)
head(get1stOfRepeatedByCol(mat2,sortB="B",sortS="B"))

nonAmbiguousNum     make numeric vector non-ambiguous (ie unique)

Description

nonAmbiguousNum makes (named) values of numeric vector 'x' unique. Note: for non-numeric use firstOfRepeated - but 1000x slower ! Return sorted non-ambiguous numeric vector (or list if 'asList'=TRUE and 'uniqOnly'=FALSE)

Usage

nonAmbiguousNum(
  x,
  uniqOnly = FALSE,
  asList = FALSE,
  nameMod = "amb_",
  callFrom = NULL
)

Arguments

x     (numeric) main input
uniqOnly     (logical) if=TRUE return unique only, if =FALSE return unique and single representative of non-unique values (with " added to name), selection of representative of repeated: first (of sorted) or middle if >2 instances
asList     (logical) return list
nameMod     (character) text to add in case on ambiguous values, default="amb_"
callFrom     (character) allow easier tracking of message(s) produced

Value

sorted non-ambiguous numeric vector (or list if 'asList'=TRUE and 'uniqOnly'=FALSE)
See Also

*firstOfRepeated* for non-numeric use (much slower !!!), *duplicated*

Examples

```r
set.seed(2017); aa <- round(rnorm(100),2); names(aa) <- 1:length(aa)
str(nonAmbiguousNum(aa))
str(nonAmbiguousNum(aa,uniq=FALSE,asLi=TRUE))
```

---

**nonredDataFrame**  
*Filter for unique elements*

Description

nonredDataFrame filters 'x' (list of char-vectors or char-vector) for elements unique (to 'ref' or if NULL to all 'x') and of character length. May be used for different 'accession' for same pep sequence (same 'peptide_id'). Note : made for treating data.frames, may be slightly slower than matrix equivalent

Usage

```r
nonredDataFrame(
  dataFr,
  useCol = c(pepID = "peptide_id", protID = "accession", seq = "sequence", mod = "modifications"),
  sepCollapse = "//",
  callFrom = NULL
)
```

Arguments

- **dataFr** (data.frame) main input
- **useCol** (character,length=2) column names of 'dataFr' to use : 1st value designates where redundant values should be gathered; 2nd value designates column of which information should be concatenated
- **sepCollapse** (character) concatenation symbol
- **callFrom** (character) allow easier tracking of messages produced

Value

This function returns a data.frame of filtered (fewer lines) with additional 2 columns 'nSamePep' (number of redundant entries) and 'concID' (concatenated content)

See Also

*combineRedBasedOnCol, correctToUnique, unique*
Examples

def1 <- data.frame(cbind(xA=letters[1:5], xB=c("h","h","f","e","f"), xC=LETTERS[1:5]))
nonredDataFrame(df1, useCol=c("xB","xC"))

nonRedundLines  Non-redundant lines of matrix

Description

nonRedundLines reduces complexity of matrix (or data.frame) if multiple consecutive (!) lines
with same values. Return matrix (or data.frame) without repeated lines (keep 1st occurrence)

Usage

nonRedundLines(dat, callFrom = NULL)

Arguments

dat     (matrix or data.frame) main input

callFrom  (character) allow easier tracking of message(s) produced

Value

matrix (or data.frame) without repeated lines (keep 1st occurrence).

See Also

firstLineOfDat, firstOfRepLines, findRepeated, firstOfRepeated, get1stOfRepeatedByCol,
combineRedBasedOnCol, correctToUnique

Examples

mat2 <- matrix(rep(c(1,1:3,3,1),2),ncol=2,dimnames=list(letters[1:6],LETTERS[1:2]))
nonRedundLines(mat2)
normalizeThis Normalize data in various modes

Description

Generic normalization of 'dat' (by columns), multiple methods may be applied. The choice of normalization procedures must be done with care, plotting the data before and after normalization may be critical to understanding the initial data structure and the effect of the procedure applied. Inappropriate methods chosen may render interpretation of (further) results incorrect.

Usage

```r
normalizeThis(
  dat,
  method = "mean",
  refLines = NULL,
  refGrp = NULL,
  mode = "proportional",
  trimFa = NULL,
  minQuant = NULL,
  sparseLim = 0.4,
  nCombin = 3,
  omitNonAlignable = FALSE,
  maxFact = 10,
  quantFa = NULL,
  expFa = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

dat matrix or data.frame of data to get normalized

method (character) may be "mean","median","NULL","none", "trimMean", "rowNormalize", "slope", "exponent", "slope2Sections", "vsn"; When NULL or 'none' is chosen the input will be returned

refLines (NULL or numeric) allows to consider only specific lines of 'dat' when determining normalization factors (all data will be normalized)

refGrp Only the columns indicated will be used as reference, default all columns (integer or colnames)

mode (character) may be "proportional", "additive"; decide if normalization factors will be applied as multiplicative (proportional) or additive; for log2-omics data mode="additive" is suggested

trimFa (numeric, length=1) additional parameters for trimmed mean
normalizeThis

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minQuant</td>
<td>(numeric) only used with method='rowNormalize': optional filter to set all values below given value as NA; see also rowNormalize</td>
</tr>
<tr>
<td>sparseLim</td>
<td>(integer) only used with method='rowNormalize': decide at which min content of NA values the function should go in sparse-mode; see also rowNormalize</td>
</tr>
<tr>
<td>nCombin</td>
<td>(NULL or integer) only used with method='rowNormalize': used only in sparse-mode (ie if content of NAs higher than content of sparseLim): Number of groups of smaller matrixes with this number of columns to be inspected initially; low values (small groups have higher chances of more common elements); see also rowNormalize</td>
</tr>
<tr>
<td>omitNonAlignable</td>
<td>(logical) only used with method='rowNormalize': allow omitting all columns which can’t get aligned due to sparseness; see also rowNormalize</td>
</tr>
<tr>
<td>maxFact</td>
<td>(numeric, length=2) only used with method='rowNormalize': max normalization factor; see also rowNormalize</td>
</tr>
<tr>
<td>quantFa</td>
<td>(numeric, length=2) additional parameters for quantiles to use with method='slope'</td>
</tr>
<tr>
<td>expFa</td>
<td>(numeric, length=1) additional parameters for method='exponent'</td>
</tr>
<tr>
<td>silent</td>
<td>(logical) suppress messages</td>
</tr>
<tr>
<td>debug</td>
<td>(logical) additional messages for debugging</td>
</tr>
<tr>
<td>callFrom</td>
<td>(character) allows easier tracking of messages produced</td>
</tr>
</tbody>
</table>

Details

In most cases of treating 'Omics'-data one works with the hypothesis that there are no global changes in the structure of all data/columns. Under this hypothesis it is very common to assume the the median (via the argument method) of all samples (ie columns) should remain constant. For examples samples/columns with less signal will be considered as having received 'accidentally' less material (eg due to the imprecision when transferring very small amounts of liquid samples). In consequence, a sample having received only 95 Thus, all measures will be multiplied by 1/0.95 (apr 1.053) to compensate for supposed lack of staring material.

With the analysis of 'Omics'-data it is very common to work with data on log-scale. In this case the argument mode should be set to additive, since adding a constant factor to log-data corresponds to a multiplicative factor on regular scale. Please note that (at this point) the methods 'slope', 'exponent', 'slope2Sections' and 'vsn' don’t distinguish between additive and proportional modes, but take the data 'as is' (you may look at the original documentation for more details, see exponNormalize, adjBy2ptReg, justvsn).

Normalization using method="rowNormalize" runs rowNormalize from this package. In this case, the working hypothesis is, that all values in each row are expected to be the same. This method could be applied when all series of values (ie columns) are replicate measurements of the same sample. There is also an option for treating sparse data (see argument sparseLim), which may, however, consume much more computational resources, in particular, when the value nCombin is low (compared to the number of samples/columns).

Normalization using method="vsn" runs justvsn from vsn (this requires a minimum of 42 rows of input-data and having the Bioconductor package vsn installed). Note: Depending on the procedure chosen, the normalized data may appear on a different scale.
numPairDeColNames

Extract pair of numeric values from vector or column-names

Description

This function extracts a pair of numeric values out of a vector or colnames (from a matrix). This is useful when pairwise comparisons are concatenated like '10c-100c', return matrix with 'index'=selComp, log2rat and both numeric. Additional white space or character text can be removed via the argument stripTxt. Of course, the separator sep needs to be specified and should not be included to 'stripTxt'.

Usage

numPairDeColNames(
  dat,
  selComp = NULL,
  stripTxt = NULL,
  sep = "-",
  colunLabel = "conc",
  sortByAbsRatio = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
orderMatrToRef

Order Lines of Matrix According to Reference (Character) Vector

Description

This function orders lines of matrix mat according to a (character) reference vector ref. To do so, all columns of mat will be considered to use the first column from left with the best (partial) matching results. This function first looks for unambiguous perfect matches, and if not found successive rounds of more elaborate partial matching will be engaged: In case of no perfect matches found, grep of ref on all columns of mat and/or grep of all columns of mat on ref (i.e. 'reverse grep') will be applied (finally a 'two way grep' approach). Until a perfect match is found each element of ref will be tested on mat and inversely (for each column) each element of mat will be tested on ref. The approach with the best number of (unique) matches will be chosen. In case of one-to-many matches, it will be tried to use most complete lines (see also last example).

Arguments

dat (matrix or data.frame) main input
dataSel (character) the column index selected
stripTxt (character, max length=2) text to ignore, if NULL heading letter and punctuation characters will be removed; default will remove all letters (and following spaces)
sep (character, length=1) separator between pair of numeric values to extract
columnLabel (character) column labels in output
sortByAbsRatio (logical) optional sorting of output by (absolute) log-ratios (most extreme ratios on top)
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a matrix

See Also

strsplit and help on regex

Examples

```r
## composed column names
mat1 <- matrix(1:8, nrow=2, dimnames=list(NULL, paste0(1:4,-"",6:9)))
numPairDeColNames(mat1)
numPairDeColNames(colnames(mat1))
## works also with simple numeric column names
mat2 <- matrix(1:8, nrow=2, dimnames=list(NULL, paste0("a",6:9)))
numPairDeColNames(mat2)
```
Usage

```r
orderMatrToRef(
  mat,
  ref,
  addRef = TRUE,
  listReturn = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

- **mat** (matrix, data.frame) main input of which rows should get re-ordered according to a (character) reference vector `ref`
- **ref** (character) reference imposing new order
- **addRef** (logical) add `ref` to output as new column
- **listReturn** (logical) allows retrieving more information in form of list
- **silent** (logical) suppress messages
- **debug** (logical) display additional messages for debugging
- **callFrom** (character) allow easier tracking of messages produced

Value

This function returns, depending on `listReturn`, either the input-matrix in new order or a list with `$mat` (the input matrix in new order), `$grep` (matched matrix) and `$col` indicating the column of `mat` finally used

See Also

- for basic ordering see `match`, `checkGrpOrder` for testing each line for expected order, `checkStrictOrder` to check for strict (ascending or descending) order

Examples

```r
mat1 <- matrix(paste0("__",letters[rep(c(1,1,2,2,3),3) +rep(0:2,each=5)], rep(1:5)), ncol=3)
orderMatrToRef(mat1, paste0(letters[c(3,4,5,3,4)],c(1,3,5,2,4)))

mat2 <- matrix(paste0("__",letters[rep(c(1,1,2,2,3),3) +rep(0:2,each=5)],
c(rep(1:5,2),1,1,3:5 ))), ncol=3)
orderMatrToRef(mat2, paste0(letters[c(3,4,5,3,4)],c(1,3,5,1,4)))

mat3 <- matrix(paste0(letters[rep(c(1,1,2,2,3),3) +rep(0:2,each=5)],
c(rep(1:5,2),1,1,3,3,5 ))), ncol=3)
orderMatrToRef(mat3, paste0("__",letters[c(3,4,5,3,4)],c(1,3,5,1,3)))
```
organizeAsListOfRepl  (re)organize data of (3-dim) array as list of replicates

Description
Organize array of all data ('arrIn', long table) into list of (replicate-)arrays (of similar type/layout) based on dimension number 'byDim' of 'arrIn' (eg 2nd or 3rd dim). Argument inspNChar defines the number of characters to consider, so if the beginning of names is the same they will be separated as list of multiple arrays. Default will search for '_' separator or trim from end if not found in the relevant dimnames

Usage
organizeAsListOfRepl(
  arrIn,
  inspNChar = 0,
  byDim = 3,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

arrIn         (array) main input
inspNChar     (integer) if inspNChar=0 the array-names (2nd dim of 'arrIn') will be cut before last '_'
byDim         (integer, length=1) dimension number along which data will be split in separate elements (considering the first inspNChar characters)
silent        (logical) suppress messages
debug         (logical) additional messages for debugging
callFrom      (character) allow easier tracking of messages produced

Value
This function returns a list of arrays (typically 1st and 2nd dim for specific genes/objects, 3rd for different measures associated with)

See Also
array

Examples
arr1 <- array(1:24,dim=c(4,3,2),dimnames=list(c(LETTERS[1:4]), paste("col",1:3,sep=""), c("ch1","ch2"))
organizeAsListofRepl(arr1)
packageDownloadStat  
*Simple Package Download Statistics from CRAN*

**Description**

This function allows accessing the most recent counts of package downloads available on http://www.datasciencemeta.com/rpackages, obtaining rank quantiles and to compare (multiple) given packages to the bulk data, optionally a plot can be drawn.

**Usage**

```r
packageDownloadStat(
    queryPackages = c("wrMisc", "wrProteo", "cif", "bcv", "FinCovRegularization"),
    countUrl = "http://www.datasciencemeta.com/rpackages",
    refQuant = (1:10)/10,
    figure = TRUE,
    log = "",
    silent = FALSE,
    callFrom = NULL,
    debug = FALSE
)
```

**Arguments**

- `queryPackages` (character or integer) package names of interest, if integer, n random packages will be picked by random
- `countUrl` (character) the url where the daily counts are available
- `refQuant` (numeric) add reference quantile values to output matrix
- `figure` (logical) decide if figure should be printed
- `log` (character) set count-axis of figure to linear or log-scale (by setting log="y")
- `silent` (logical) suppress messages
- `callFrom` (character) allow easier tracking of messages produced
- `debug` (logical) additional messages for debugging

**Details**

The task of checking the number of downloads for a given package has been addressed by several packages (eg dlstats, cranlogs, adjustedcranlogs). Detailed articles on this subject have been published on R-Hub (https://blog.r-hub.io/2020/05/11/packagerank-intro/) and on R-bloggers (https://www.r-bloggers.com/2020/10/a-cran-downloads-experiment/).

This function only allows accessing the most recent counts as listed on the website of www.datasciencemeta.com. Please note, that reading all lines from the website may take a few seconds. To get a better understanding of the counts read, reference quantiles for download-counts get added by default (see argument refQuant). The (optional) figure can be drawn in linear scale (default, with minor zoom.
to lower number of counts) or in log (necessary for proper display of the entire range of counts), by setting the argument log="y".

The number of downloads counted by RStudio may not be a perfect measure for the actual usage/popularity of a given package, the articles cited above discuss this in more detail. For example, multiple downloads from the same IP or subsequent downloads of multiple (older) versions of the same package are counted, too.

Value

This function returns a matrix with download counts (or NULL if the web-site can’t be accessed or the query-packages are not found there)

See Also

packages cranlogs and packageRank

Examples

## Let's try a microscopic test-file (NOT representative for true up to date counts !!)
pack1 <- c("cif", "bcv", "FinCovRegularization", "wrMisc", "wrProteo")
testFi <- file.path(system.file("extdata", package="wrMisc"), "rpackagesMicro.html")
packageDownloadStat(pack1, countUrl=testFi, log="y", figure=FALSE)
## For real online counting simply drop the argument countUrl
Details

Note, this has been primarily developed for undirected interaction networks, the resulting propensity-matrix does not show any orientation any more. In a number of applications (eg in protein-protein interaction networks, PPI) the resulting matrix may be rather sparse.

Value

This function returns matrix or data.frame

See Also

uses typically input from filterNetw

Examples

```r
pairs3L <- matrix(LETTERS[c(1,3,3, 2,2,1)], ncol=2) # loop of 3
(netw13pr <- pairsAsPropensMatr(pairs3L)) # as prop matr
```

```
partialDist

Partial distance matrix (focus on closest)

Description

partialDist calculates distance matrix like dist for 1- or 2-dim data, but only partially, ie only cases of small distances. This function was made for treating very large data-sets where only very close distances to a given point need to be found, it allows to overcome memory-problems with larger data (and faster execution with > 50 rows of `dat`).

Usage

```
partialDist(
  dat,
  groups,
  overLap = TRUE,
  method = "euclidean",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

dat  (matrix of numeric values) main input

groups (factor) to split using cut or specific custom grouping (length of dat)

overLap (logical) if TRUE make groups overlapping by 1 value (ie maintain some context-information)
\textbf{partUnlist} \hspace{1cm} \textit{Partial unlist of lists of lists}

\textbf{Description}

\texttt{partUnlist} does partial unlist for treating list of lists: New (returned) list has one level less of hierarchy (Highest level list will be appended). In case of conflicting (non-null) listnames a prefix will be added. Behaviour different to \texttt{unlist} when unlisting list of matrixes.

\textbf{Usage}

\texttt{partUnlist(lst, sep = ",", silent = FALSE, debug = FALSE, callFrom = NULL)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{lst} \hspace{1cm} (list) main input, list to be partially unlisted
  \item \texttt{sep} \hspace{1cm} (character, length=1) separator for names
  \item \texttt{silent} \hspace{1cm} (logical) suppress messages
  \item \texttt{debug} \hspace{1cm} (logical) display additional messages for debugging
  \item \texttt{callFrom} \hspace{1cm} (character) allow easier tracking of message(s) produced
\end{itemize}

\textbf{Value}

This function returns a list with partially reduced nested structure

\textbf{See Also}

\texttt{unlist, asSepList}
Examples

```r
partUnlist(list(list(a=11:12,b=21:24), list(c=101:101,d=201:204)))
li4 <- list(c=1:3, M2=matrix(1:4,ncol=2), L3=list(L1=11:12, M3=matrix(21:26,ncol=2)))
partUnlist(li4)
unlist(li4, rec=FALSE)
```

Advanced paste-collapse

Description

This function is a variant of `paste` for convenient use of paste-collapse and separation of last element to paste (via 'lastCol'). This function was made for more human like enumerating in output and messages. If multiple arguments are given without names they will all be concatenated, if they contain names lazy evaluation for names will be tried (with preference to longest match to argument names). Note that some special characters (like backslash) may need to be protected when used with 'collapse' or 'quoteC'. Returns character vector of length 1 (everything pasted together)

Usage

```r
pasteC(..., collapse = "", lastCol = " and ", quoteC = "")
```

Arguments

- `...`: (character) main input to be collapsed
- `collapse`: (character,length=1) element to use for collapsing
- `lastCol`: (character) text to use before last item enumerated element
- `quoteC`: (character) text to use for citing with quotations (default ")

Value

This function returns a character vector of truncated versions of input `txt`

See Also

- `paste` for basic paste

Examples

```r
pasteC(1:4)
```
Description

This function produces a logical matrix to be used as filter for lines of ‘dat’ for sufficient presence of non-NA values (i.e., limit number of NAs per line). Filter abundance/expression data for min number and/or ratio of non-NA values in at least 1 of multiple groups. This type of procedure is common in proteomics and transcriptomics, where a NA can many times be associated with quantitation below detection limit.

Usage

presenceFilt(
  dat,
  grp,
  maxGrpMiss = 1,
  ratMaxNA = 0.8,
  minVal = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

dat matrix or data.frame (abundance or expression-values which may contain some NAs).
grp factor of min 2 levels describing which column of ‘dat’ belongs to which group (levels 1 & 2 will be used)
maxGrpMiss (numeric) at least 1 group has not more than this number of NAs (otherwise mark line as bad)
ratMaxNA (numeric) at least 1 group reaches this content of non-NA values
minVal (default NULL or numeric), any value below will be treated like NA
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

logical matrix (with separate col for each pairwise combination of ‘grp’ levels) indicating if line of ‘dat’ acceptable based on NAs (and values minVal)
### Examples

```r
mat <- matrix(rep(8, 150), ncol=15, dimnames=list(NULL,
    paste0(rep(LETTERS[4:2], each=6), 1:6)[c(1:5, 7:16)]))
mat[lower.tri(mat)] <- NA
mat[, 15] <- NA
mat[c(2:3, 9), 14:15] <- NA
mat[c(1, 10), 13:15] <- NA
mat
presenceFilt(mat, rep(LETTERS[4:2], c(5, 6, 4)))
presenceFilt(mat, rep(1:2, c(9, 6)))

# one more example
dat1 <- matrix(1:56, ncol=7)
dat1[c(2, 3, 4, 5, 6, 10, 12, 18, 19, 20, 22, 23, 26, 27, 28, 30, 31, 34, 38, 39, 50, 54)] <- NA
dat1; presenceFilt(dat1, gr=gl(3, 3)[-3:4], maxGr=0)
presenceFilt(dat1, gr=gl(2, 4)[-1], maxGr=1, ratM=0.1)
presenceFilt(dat1, gr=gl(2, 4)[-1], maxGr=2, rat=0.5)
```

### Description

The aim of this function is to filter for each group of columns for sufficient data as non-NA.

### Usage

```r
presenceGrpFilt(dat, grp, presThr = 0.75, silent = FALSE, callFrom = NULL)
```

### Arguments

- **dat**: matrix or data.frame (abundance or expression-values which may contain some NAs).
- **grp**: factor of min 2 levels describing which column of ‘dat’ belongs to which group (levels 1 & 2 will be used)
- **presThr**: (numeric) min ratio of non-NA values (per group) for returning a given line & group as TRUE
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of messages produced
Details

This function allows to identify lines with an NA-content above the threshold presThr per group as defined by the levels of factor grp. With different types of projects/questions different threshold presThrs per group may be useful. For example, if one would like to keep the degree of threshold presThrs per group rather low, one could use a value of 0.75 (ie >= 75

Value

logical matrix (with on column for each level of grp)

See Also

presenceFilt, there are also other packages totally dedicated to filtering on CRAN and Bioconductor

Examples

mat <- matrix(NA, nrow=11, ncol=6)
mat[lower.tri(mat)] <- 1
mat <- cbind(mat, mat[,1:4])
rownames(mat) <- c(paste0("re",1:6), paste0("x",1:4))

## accept 1 NA out of 4, 2 NA out of 6 (ie certainly present)
(filt0a <- presenceGrpFilt(mat, rep(1:2, c(6,4)), pres=0.66))

## accept 2 NA out of 4, 2 NA out of 6 (ie min 50% present)
(filt0b <- presenceGrpFilt(mat, rep(1:2, c(6,4)), pres=0.5))

## accept 3 NA out of 4, 4 NA out of 6 (ie possibly present)
(filt0c <- presenceGrpFilt(mat, rep(1:2, c(6,4)), pres=0.19))

Description

Some characters do have a special meaning when used with regular expressions. This concerns characters like a point, parenthesis, backslash etc. Thus, when using grep or any related command, such special characters must get protected in order to get considered as they are.

Usage

protectSpecChar(x, prot = c(".", "\", "|", ("", ")", ",", ",^", ",$", ",*", ",+", ",?"), silent = TRUE, callFrom = NULL)
Arguments

- **x**: character vector to be prepared for use in regular expressions
- **prot**: (character) collection of characters that need to be protected
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of messages produced

Value

This function returns a modified character vector

Examples

```r
aa <- c("abc","abcde","ab.c","ab.c.e","ab*c","ab\d")
grepl("b.", aa)  # all TRUE
grepl("b\.", aa)  # manual protection
grepl(protectSpecChar("b."), aa)
```

---

**pVal2lfdr**

*Convert p-values to lfdr*

Description

This function takes a numeric vector of p-values and returns a vector of lfdr-values (local false discovery) using the package `fdrtool`. Multiple testing correction should be performed with caution, short series of p-values typically pose problems for transforming to lfdr. The transformation to lfdr values may give warning messages, in this case the resultant lfdr values may be invalid!

Usage

```r
pVal2lfdr(x, silent = TRUE, callFrom = NULL)
```

Arguments

- **x**: (numeric) vector of p.values
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of messages produced

Value

This function returns a (numeric) vector of lfdr values (or NULL if data insufficient to run the function 'fdrtool')

See Also

lfdr from `fdrtool`, other p-adjustments (multiple test correction, eg FDR) in `p.adjust`
Examples

```r
## Note that this example is too small for estimating really meaningful fdr values
## In consequence, a warning will be issued.
set.seed(2017); t8 <- matrix(round(rnorm(160,10,0.4),2), ncol=8, 
  dimnames=list(letters[1:20], c("AA1","BB1","CC1","DD1","AA2","BB2","CC2","DD2")))
t8[3:6,1:2] <- t8[3:6,1:2]+3  # augment lines 3:6 (c-f) for AA1&BB1
t8[5:8,5:6] <- t8[5:8,5:6]+3  # augment lines 5:8 (e-h) for AA2&BB2 (c,d,g,h should be found)
head(pVal2lfdr(apply(t8, 1, function(x) t.test(x[1:4], x[5:8])$p.value)))
```

### Distance of categorical data (Jaccard, Rand and adjusted Rand index)

**Description**

`randIndFx` calculates distance of categorical data (as Rand Index, Adjusted Rand Index or Jaccard Index). Note: uses/requires package `flexclust` Methods so far available (via `flexclust`): "ARI" .. adjusted Rand Index, "RI" .. Rand index, "J" .. Jaccard, "FM" .. Fowlkes-Mallows.

**Usage**

```r
randIndFx(ma, method = "ARI", adjSense = TRUE, silent = FALSE, callFrom = NULL)
```

**Arguments**

- `ma` (matrix) main input for distance calculation
- `method` (character) name of distance method (eg "ARI","RI","J","FM")
- `adjSense` (logical) allows introducing correlation/anticorrelation (interprete neg distance results as anti)
- `silent` (logical) suppres messages
- `callFrom` (character) allow easier tracking of messages produced

**Value**

This function returns a distance matrix

**See Also**

`comPart` in `randIndex`

**Examples**

```r
set.seed(2016); tab2 <- matrix(sample(1:2, size=42, replace=TRUE), ncol=7)
if(requireNamespace("flexclust")) { flexclust::comPart(tab2[1,,] , tab2[2,])
    flexclust::comPart(tab2[1,,] , tab2[3,])
    flexclust::comPart(tab2[1,,] , tab2[4,]) }
## via randIndFx():
randIndFx(tab2, adjSense=FALSE)
cor(t(tab2))
randIndFx(tab2, adjSense=TRUE)
```
Description

Count the number of instances where the corresponding columns of 'dat' have a value matching the group number as specified by 'grp'. Counting will be performed/repeated independently for each line of 'dat'. Returns array (1st dim is rows of dat, 2nd is unique(grp), 3rd dim is ok/bad), these results may be tested using eg `fisher.test`. This function was made for preparing to test the ranking of multiple features (lines in 'mat') including replicates (levels of 'grp').

Usage

`rankToContigTab(dat, grp)`

Arguments

- `dat` (matrix or data.frame of integer values) ranking of multiple features (lines), equal ranks may occur
- `grp` (integer) expected ranking

Value

array (1st dim is rows of dat, 2nd is unique(grp), 3rd dim is ok/bad)

See Also

`lm`

Examples

```r
# Let's create a matrix with ranks (equal ranks do occur)
ma0 <- matrix(rep(1:3,each=6), ncol=6, dimnames=list(c("li1","li2","ref"), letters[1:6]))
ma0[1,6] <- 1 # create item not matching correctly
ma0[2,] <- c(3:1,2,1,3) # create items not matching correctly
gr0 <- gl(3,2) # the expected ranking (as duplicates)
(count0 <- rankToContigTab(ma0,gr0))
cTab <- t(apply(count0, c(1,3) ,sum)) # Now we can compare the ranking of line1 to ref ...
fisher.test(cTab[,c(3,1)]) # test li1 against ref
fisher.test(cTab[,c(3,2)]) # test li2 against ref
```
**ratioAllComb**  
*Calculate all ratios between x and y*

**Description**

This function calculates all possible pairwise ratios between all individual values of x and y, or samples up to a maximum number of combinations.

**Usage**

```r
ratioAllComb(
  x,
  y,
  maxLim = 10000,
  isLog = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `x` (numeric) vector, numerator for constructing ratios
- `y` (numeric) vector, denominator for constructing ratios
- `maxLim` (integer) allows reducing complexity by drawing for very long x or y
- `isLog` (logical) adjust ratio calculation to log-data
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

**Examples**

```r
set.seed(2014); ra1 <- c(rnorm(9,2,1),runif(8,1,2))
ratioAllComb(ra1[1:9],ra1[10:17])
boxplot(list(norm=ra1[1:9], unif=ra1[10:17], rat=ratioAllComb(ra1[1:9],ra1[10:17])))
```
ratioToPpm

Convert ratio to ppm

Description

This function transforms ratio 'x' to ppm (parts per million). If 'y' not given (or different length as 'x'), then 'x' is assumed as ratio otherise rations are constructed as x/y is used lateron. Does additional checking: negative values not expected - will be made absolute!

Usage

ratioToPpm(
  x,
  y = NULL,
  nSign = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

  x  (numeric) main input
  y  (numeric) optional value to construct ratios (x/y). If NULL (or different length as 'x'), then 'x' will be considered as ratio.
  nSign  (numeric) number of significan digits
  silent  (logical) suppress messages
  debug  (logical) additional messages for debugging
  callFrom  (character) allow easier tracking of messages produced

Value

This function returns a numeric vector of ppm values

See Also

XYToDiffPpm for ppm of difference as used in mass spectrometrie

Examples

set.seed(2017); aa <- c(1.000001,0.999999,1+rnorm(10,0,0.001))
cbind(x=aa,ppm=ratioToPpm(aa,nSign=4))
**readCsvBatch**

**Read batch of csv-files**

**Description**

This function was designed to read screening data split in parts (with common structure) and saved to multiple files, to extract the numeric columns and to compile all (numeric) data to a single array (or list). Some screening platforms save results while progressing through a pile of microtiter-plates separately. The organization of the resultant files is structured through file-names and all files have exactly the same organization of lines and columns/ European or US-formatted csv files can be read, if argument `fileFormat` is `NULL` both types will be tested, otherwise it allows to specify a given format. The presence of headers (to be used as column-names) may be tested using `checkFormat`.

**Usage**

```r
code
readCsvBatch(
  fileNames = NULL,
  path = ".",
  fileFormat = "Eur",
  checkFormat = TRUE,
  returnArray = TRUE,
  columns = c("Plate", "Well", "StainA"),
  excludeFiles = "All infected plates",
  simpleNames = TRUE,
  minNamesLe = 4,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `fileNames` (character) names of files to be read, if NULL all files fitting 'fileFormat'
- `path` (character) where files should be read (folders should be written in R-style)
- `fileFormat` (character) may be NULL (both US and European formats will be tried), 'Eur' or 'US'
- `checkFormat` (logical) if TRUE: check header, remove empty columns, 1st line if all empty, set output format for each file to matrix, if rownames are increasing integers try to use 2nd of 'columns' as rownames
- `returnArray` (logical) allows switching from array to list-output
- `columns` (NULL or character) column-headers to be extracted (if specified), 2nd value may be column with rownames (if rownames are encountered as increasing rownames)
- `excludeFiles` (character) names of files to exclude (only used when reading all files of given directory)
simpleNames  (logical) allows truncating names (from beginning) to get to variable part (using .trimFromStart()), but keeping `minNamesLe`
minNamesLe  (interger) min length of column-names if simpleNames=TRUE
silent  (logical) suppress messages
debug  (logical) additional messages for debugging
callFrom  (character) allows easier tracking of messages produced

Value

This function returns an array (or list if returnArray=FALSE) of all numeric data read (numerical columns only) from individual files

See Also

read.table, writeCsv, readXlsxBatch

Examples

```r
path1 <- system.file("extdata", package="wrMisc")
fiNa <- c("pl01_1.csv","pl01_2.csv","pl02_1.csv","pl02_2.csv")
datAll <- readCsvBatch(fiNa, path1)
str(datAll)
## batch reading of all csv files in specified path :
datAll2 <- readCsvBatch(fileNames=NULL, path=path1, silent=TRUE)
```

silent = FALSE,
callFrom = NULL,
debug = FALSE
)

Arguments

query (character) vector of file-names to be read, if "." all files will be read (no matter what their extension might be)

path (character) path for reading files, if NULL or NA the current directory will be used

dec (character, length=1) decimals to use, will be passed to fread or read.delim

header (character, length=1) path for reading files, if NULL or NA the current directory will be used, will be passed to fread or read.delim

strip.white (logical, length=1) Strips leading and trailing whitespaces of unquoted fields, will be passed to fread or read.delim

blank.lines.skip (logical, length=1) If TRUE blank lines in the input are ignored. will be passed to fread or read.delim

fill (logical, length=1) If TRUE then in case the rows have unequal length, blank fields are implicitly filled, will be passed to fread or read.delim

filtCol (integer, length=1) which columns should be used for filtering, if NULL or NA all data will be returned

filterAsInf (logical, length=1) filter as inferior or equal (TRUE) or superior or equal threshold

filtVal (numeric, length=1) which numeric threshold should be used for filtering, if NULL or NA all data will be returned

silent (logical) suppress messages

callFrom (character) allow easier tracking of messages produced

debug (logical) display additional messages for debugging

Details

If you want to provide a flexible pattern of file-names, this has to be done before calling this function, eg using grep to provide an explicit collection of files. However, it is possible to read different files from different locations-directories, the length of path must match the length of query

Value

This function returns a list of data.frames

See Also

fread, read.delim, for reading batch of csv files : readCsvBatch
Examples

```r
path1 <- system.file("extdata", package="wrMisc")
fiNa <- c("a1.txt","a2.txt")
allTxt <- readTabulatedBatch(fiNa, path1)
str(allTxt)
```

---

**readVarColumns**

**Read tabular content of files with variable number of columns**

**Description**

Reading the content of files where the number of separators (eg tabulation) is variable poses problems with traditional methods for reading files, like `read.table`. This function reads each line independently and then parses all separators therein. The first line is assumed to be column-headers. Finally, all data will be returned in a matrix adopted to the line with most separators and if the number of column-headers is insufficient, new (unique) column-headers will be generated. Thus, the lines may contain different number of elements, empty elements (ie tabular fields) will always get added to right of data read and their content will be as defined by argument `emptyFields` (default NA).

**Usage**

```r
readVarColumns(
  fiName,
  path = NULL,
  sep = "\t",
  header = TRUE,
  emptyFields = NA,
  refCo = NULL,
  supNa = NULL,
  silent = FALSE,
  callFrom = NULL
)
```

**Arguments**

- `fiName` (character) file-name
- `path` (character) optional path
- `sep` (character) separator (between columns)
- `header` (logical) indicating whether the file contains the names of the variables as its first line.
- `emptyFields` (NA or character) missing headers will be replaced by the content of 'emptyFields', if NA the last column-name will be re-used and a counter added
- `refCo` (integer) for custom choice of column to be used as row-names (default will use 1st text-column)
**supNa** (character) base for constructing name for columns wo names (+counter starting at 2), default column-name to left of 1st col wo colname

**silent** (logical) suppress messages

**callFrom** (character) allow easier tracking of messages produced

**Details**

Note, this functions assumes one line of header and at least one line of data! Note, for numeric data the comma is assumed to be US-Style (as '.'). Note, that it is assumed, that any missing fields for the complete tabular view are missing on the right (ie at the end of line)!

**Value**

This function returns a matrix (character or numeric)

**See Also**

for regular 'complete' data **read.table**

**Examples**

```r
path1 <- system.file("extdata",package="wrMisc")
fiNa <- "Names1.tsv"
datAll <- readVarColumns(fiName=file.path(path1,fiNa))
str(datAll)
```

---

**Description**

**readXlsxBatch** reads data out of multiple xlsx files, the sheet indicated by 'sheetInd' will be considered. All files must have the same organization of data, as this is typically the case when high-throughput measurements are automatically saved while experiments progress. In particular, the first file read is used to structure the output.

**Usage**

```r
readXlsxBatch(
  fileNames = NULL,
  path = ".",
  fileExtension = "xlsx",
  excludeFiles = NULL,
  sheetInd = 1,
  checkFormat = TRUE,
  returnArray = TRUE,
  columns = c("Plate", "Well", "StainA"),
)```
Arguments

fileNames (character) provide either explicit list of file-names to be read or leave NULL for reading all files ending with 'xlsx' in path specified with argument path

path (character) there may be a different path for each file

fileExtension (character) extension of files (default='xlsx')

excludeFiles (character) names of files to exclude (only used when reading all files of given directory)

sheetInd (character or integer) specify which sheet to extract (as exact name of sheet or sheet-number, eg sheet1Ind=2 will extract always the 2nd sheet (no matter the name); if given as sheet-name but nor present in file an empty list-elements wil be returned

checkFormat (logical) if TRUE: check header, remove empty columns, if rownames are increasing integeres it will search for fisrt column with different entries to use as rownames

returnArray (logical) allows switching from array to list-output

columns (NULL or character) column-headers to be extracted (if specified, otherwise all columns will be extracted)

simpleNames (integer), if NULL all characters of fileNames will be maintained, otherwise allows truncating names (from beginning) to get to variable part (using .trimFromStart()), but keeping at least the number of characters indicated by this argument

silent (logical) suppress messages

callFrom (character) allows easier tracking of messages produced

Details

By default all columns with text-content may be eliminated to keep the numeric part only, which may then get organized to a 3-dim numeric array (where the additional files will be used as 2nd dimension and multiple columns per file shown as 3rd dimension).

NOTE: (starting from version wrMisc-1.5.5) requires packages readxl and Rcpp being installed! (This allows much faster and memory efficient processing than previous use of package 'xlsx')

Value

This function returns a list of data.frames

See Also

read_excel; for simple reading of (older) xls-files under 32-bit R one may also see the package RODBC
Examples

```r
path1 <- system.file("extdata", package="wrMisc")
fiNa <- c("pl01_1.xlsx","pl01_2.xlsx","pl02_1.xlsx","pl02_2.xlsx")
datAll <- readXlsxBatch(fiNa, path1)
str(datAll)
## Now let's read all xlsx files of directory
datAll2 <- readXlsxBatch(path=path1, silent=TRUE)
identical(datAll, datAll2)
```

---

reduceTable

Reduce table by aggregating smaller groups

Description

reduceTable treats/reduces results from `table` to `nGrp` groups, optional indiv resolution of 'sep-
arFirst' (numeric or NULL). Mainly made for reducing the number of classes for betters plots with `pie`

Usage

```r
reduceTable(tab, separFirst = 4, nGrp = 15)
```

Arguments

- **tab**: output of `table`
- **separFirst**: (integer or NULL) optinal separation of n 'separFirst' groups (value < 2 or NULL will priviledge more uniform size of groups, higher values will cause small inital and larger tailing groups)
- **nGrp**: (integer) number of groups expected

Value

This function returns a numeric vector with number of counts and class-borders as names (like `table`).

See Also

- `table`

Examples

```r
set.seed(2018); dat <- sample(11:60,200,repl=TRUE)
pie(table(dat))
pie(reduceTable(table(dat), sep=NULL))
pie(reduceTable(table(dat), sep=NULL), init.angle=90,
clockwise=TRUE, col=rainbow(20)[1:15], cex=0.8)
```
regrBy1or2point

Rescaling according to reference data using linear regression.

Description

regrBy1or2point does rescaling: linear transform simple vector ‘inDat’ that (mean of) elements of names cited in ‘refLst’ will end up as values ‘regrTo’. Regress single vector according to ‘refLst’ (describing names of inDat). If ‘refLst’ contains 2 groups, the 1st group will be set to the 1st value of ‘regrTo’ (and the 2nd group of ‘refLst’ to the 2nd ‘regrTo’)

Usage

regrBy1or2point(
inDat,  
refLst,  
regrTo = c(1, 0.5),  
silent = FALSE,  
callFrom = NULL
)

Arguments

inDat matrix or data.frame

refLst list of names existing in inDat (one group of names for each value in ‘regrTo’), to be transformed in values precised in ‘regrTo’; if no matches to names of ‘inDat’ found, the 2 lowest and/or highest highest values will be chosen

regrTo (numeric,length=2) range (at scale 0-1) of target-values for mean of elements cited in ‘refLst’

silent (logical) suppress messages

callFrom (character) allows easier tracking of message(s) produced

Value normalized matrix

See Also

adjBy2ptReg, regrMultBy1or2point

Examples

set.seed(2016); dat1 <- 1:50 +(1:50)*round(runif(50),1)
names(dat1) <- 1:length(dat1)
reg1 <- regrBy1or2point(dat1,refLst=c("2","49"))
plot(reg1,dat1)
regrMultBy1or2point

Rescaling of multiple data-sets according to reference data using regression

Description

regrMultBy1or2point regresses each col of matrix according to 'refLst' (describing rownames of inDat). If 'refLst' contains 2 groups, the 1st group will be set to the 1st value of 'regrTo' (and the 2nd group of 'refLst' to the 2nd 'regrTo')

Usage

regrMultBy1or2point(
inDat,
refLst,
regrTo = c(1, 0.5),
silent = FALSE,
callFrom = NULL
)

Arguments

inDat matrix or data.frame
refLst list of names existing in inDat (one group of names for each value in 'regrTo'), to be transformed in values precised in 'regrTo'; if no matches to names of 'inDat' found, the 2 lowest and/or highest highest values will be chosen
regrTo (numeric,length=2) range (at scale 0-1) of target-values for mean of elements cited in 'refLst'
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Value

normalized matrix

See Also

adjBy2ptReg, regrBy1or2point

Examples

set.seed(2016); dat2 <- round(cbind(1:50 +(1:50)*runif(50),2.2*(1:50) +rnorm(50,0,3)),1)
rownames(dat2) <- 1:nrow(dat2)
reg1 <- regrBy1or2point(dat2[,1],refLst=list(as.character(5:7),as.character(44:45)))
reg2 <- regrMultBy1or2point(dat2,refLst=list(as.character(5:7),as.character(44:45)))
plot(dat2[,1],reg2[,1])
identical(reg1,reg2[,1])
identical(dat2[,1],reg2[,1])
renameColumns  

**Description**

This function renames columns of `refMatr` using 2-column matrix (or data.frame) indicating old and new names (for replacement).

**Usage**

```
renameColumns(refMatr, newName, silent = FALSE, debug = FALSE, callFrom = NULL)
```

**Arguments**

- `refMatr`  
  matrix (or data.frame) where column-names should be changed
- `newName`  
  (matrix of character) giving correspondence of old to new names (number of lines must match number of columns of `refMatr`)
- `silent`  
  (logical) suppres messages
- `debug`  
  (logical) additional messages for debugging
- `callFrom`  
  (character) allows easier tracking of messages produced

**Value**

This function returns a matrix (or data.frame) with renamed columns

**Examples**

```r
ma <- matrix(1:8,ncol=4,dimnames=list(1:2,LETTERS[1:4]))
replBy1 <- cbind(new=c("dd","bb","z_"),old=c("D","B","zz"))
replBy2 <- matrix(c("D","B","zz","dd","bb","z_"),ncol=2)
replBy3 <- matrix(c("X","Y","zz","xx","yy","z_"),ncol=2)
renameColumns(ma,replBy1)
renameColumns(ma,replBy2)
renameColumns(ma,replBy3)
```

reorgByCluNo  

**Reorganize matrix according to clustering-output**

**Description**

Reorganize input matrix as sorted by cluster numbers (and geometric mean) according to vector with cluster names, and index for sorting per cluster and per geometric mean. In case mat is an array, the 3rd dimension will be considered as 'column' with arguments useColumn (and cluNo, if it designs a 'column' of mat).
reorgByCluNo

Usage

reorgByCluNo(
  mat,
  cluNo,
  useColumn = NULL,
  meanCol = NULL,
  addInfo = TRUE,
  retList = FALSE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)

Arguments

mat (matrix or data.frame) main input
cluNo (positive integer, length to match nrow(mat) initial cluster numbers for each line of 'mat' (obtained by separate clustering or other segmentation) or may desinn column of mat to use as cluster-numbers
useColumn (character or integer) the columns to use from mat as main data (default will use all, exep cluCol and/or meanCol if they design columns))
meanCol (character or integer) alternative summarizing data for intra-cluster sorting (instead of geometric mean)
addInfo (logical) allows adding of columns 'index', 'geoMean' and 'cluNo' (or array if FALSE)
retList (logical) return as list of matrixes (or array if FALSE)
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced
debug (logical) additional messages for debugging

Value

This function returns a list or array (as 2- or 3 dim) with possible number of occurances for each of the 3 elements in nMax. Read results vertical : out[1][] or out[][1] .. (multiplicative) table for 1st element of nMax; out[2][] .. for 2nd

See Also

pairwise combinations combn, clustering kmeans

Examples

dat1 <- matrix(round(runif(24),2), ncol=3, dimnames=list(NULL,letters[1:3]))
clu <- stats::kmeans(dat1, 5)$cluster
reorgByCluNo(dat1, clu)

dat2 <- cbind(dat1, clu=clu)
reorgByCluNo(dat2, "clu")
replicateStructure  Search and Select Groups of Replicates

Description

This function was designed for mining annotation information organized in multiple columns to identify the (potential) grouping of multiple samples, ie to determine factor levels. The argument method allows further finetuning if high or low number of groups should be preferred, if multiple columns may be combined, or to choose a particular custom column for designating factor levels.

Usage

replicateStructure(
  x,
  method = "median",
  sep = "__",
  exclNoRepl = TRUE,
  trimNames = FALSE,
  includeOther = FALSE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)

Arguments

x (matrix or data.frame) the annotation to inspect; each column is supposed to describe another set of annotation/metadata for the rows of x (min 1 row and 1 column),

method (character, length=1) the procedure to choose column(s) with properties of information, may be highest or max (max number of levels) lowest or min (min number of levels), median (median of all options for number of levels), combAll (combine all columns of x) or combNonOrth (combine only non-orthogonal columns of x, to avoid avoid n lines with n levels); lazy evaluation of the argument is possible

sep (character) separator used when a method combining multiple columns (eg combAll, combNonOrth) is chosen (should not appear anywhere in x)

exclNoRepl (logical) decide whether columns with all values different (ie no replicates or max divergency) should be excluded

trimNames (logical) optional trimming of names in x by removing redundant heading and tailing text

includeOther (logical) include $allCols with pattern of (all) other columns

silent (logical) suppress messages

callFrom (character) allow easier tracking of messages produced

debug (logical) additional messages for debugging
**replNAbyLow**

Details

Statistical tests require specifying which samples should be considered as replicates of whom. In some cases, like the Sdrf-format, automatic mining of such annotation to identify an experiment’s underlying structure of replicates may be challenging, since the key information may not always be found in the same column. For this reason this function allows inspecting all columns of a matrix of data.frame to identify which columns may serve describing groups of replicates.

The argument exclNoRepl=TRUE allows excluding all columns with different content for each line (like line-numbers), i.e., information without any replicates. It is set by default to TRUE to exclude such columns, since statistical tests usually do require some replicates.

When using as method="combAll", there is risk all lines (samples) will be be considered different and no replicates remain. To avoid this situation the argument can be set to method="combNonOrth". Using this mode it will be checked if adding more columns will lead to complete loss of replicates, and -if so- concerned columns omitted.

Value

This function returns a list with $col (column index relativ to x), $lev (abstract labels of level), $meth (note of method finally used) and $allCols with general replicate structure of all columns of x

See Also
duplicated, uses trimRedundText

Examples

```r
## a is all different, b is groups of 2, 
## c & d are groups of 2 but NOT 'same general' pattern as b
strX <- data.frame(a=letters[18:11], b=letters[rep(c(3:1,4), each=2)],
  c=letters[rep(c(5,8:6), each=2)], d=letters[c(1:2,1:3,3:4,4)],
  e=letters[rep(c(4,8,4,7),each=2)], f=rep("z",8) )
strX
replicateStructure(strX[,1:2])
replicateStructure(strX[,1:4], method="combAll")
replicateStructure(strX[,1:4], method="combAll", exclNoRepl=FALSE)
replicateStructure(strX[,1:4], method="combNonOrth", exclNoRepl=TRUE)
replicateStructure(strX, method="lowest")
replicateStructure(strX, method=3, includeOther=TRUE)  # custom choice of 3rd column
```
Description

With several screening techniques used in high-throughput biology values at/below detection limit are returned as NA. However, the resultant NA-values may be difficult to analyse properly, simply ignoring NA-values may not be a good choice. When (technical) replicate measurements are available, one can look for cases where one gave an NA while the other did not with the aim of investigating such 'NA-neighbours'. replNAbyLow locates and replaces NA values by (random) values from same line & same group ‘grp’. The origin of NAs should be predominantly absence of measure (quantitation) due to signal below limit of detection and not saturation at upper detection limit or other technical problems. Note, this approach may be not optimal if the number of NA-neighbours is very low. Replacemnt is done -depending on argument 'unif' - by Gaussian random model based on neighbour values (within same group), using their means and sd, or a uniform random model (min and max of neighbour values). Then numeric matrix (same dim as 'x') with NA replaced is returned.

Usage

replNAbyLow(
  x,
  grp,
  quant = 0.8,
  signific = 3,
  unif = TRUE,
  absOnly = FALSE,
  seed = NULL,
  silent = FALSE,
  callFrom = NULL
)

Arguments

x (numeric matrix or data.frame) main input
grp (factor) to organize replicate columns of (x)
quant (numeric) quantile form 'neighbour' values to use as upper limit for random values
signific number of signif digits for random values
unif (logical) toggle between uniform and Gaussian random values
absOnly (logical) if TRUE, make negative NA-replacement values positive as absolute values
seed (integer) for use with set.seed for reproducible output
silent (logical) suppress messages
callFrom (character) allow easier tracking of message(s) produced

Value

numeric matrix (same dim as 'x') with NA replaced
replPlateCV

See Also
   naOmit, na.fail

Examples
   dat <- matrix(round(rnorm(30),2),ncol=6); grD <- gl(2,3)
   dat[sort(sample(1:30,9,repl=FALSE))] <- NA
   dat; replNAbyLow(dat,gr=grD)

replPlateCV  CV of replicate plates (list of matrixes)

Description
   replPlateCV gets CVs of replicates from list of 2 or 3-dim arrays (where 2nd dim is replicates,
   3rd dim may be channel). Note : all list-elements of must MUST have SAME dimensions ! When
   treating data from microtiter plates (eg 8x12) data are typically spread over multiple plates, ie initial
   matrixes that are the organized into arrays. Returns matrix or array (1st dim is intraplate-position,
   2nd .. plate-group/type, 3rd .. channels)

Usage
   replPlateCV(lst, callFrom = NULL)

Arguments
   lst
      list of matrixes : suppose lines are independent elements, columns are replicates
      of the 1st column. All matrixes must have same dimensions
   callFrom
      (character) allows easier tracking of messages produced

Value
   matrix or array (1st dim is intraplate-position, 2nd .. plate-group/type, 3rd .. channels)

See Also
   rowCVs, seealso arrayCV

Examples
   set.seed(2016); ra1 <- matrix(rnorm(3*96),nrow=8)
   pla1 <- list(ra1[,1:12],ra1[,13:24],ra1[,25:36])
   replPlateCV(pla1)
   arrL1 <- list(a=array(as.numeric(ra1)[1:192],dim=c(8,12,2)),
                  b=array(as.numeric(ra1)[97:288],dim=c(8,12,2)))
   replPlateCV(arrL1)
**Description**

`rmDupl2colMatr` removes lines of matrix that are redundant /duplicated for 1st and 2nd column (irrespective of content of their columns). The first occurrence of redundant /duplicated elements is kept.

**Usage**

```r
rmdupl2colmatr(mat, useCol = c(1, 2))
```

**Arguments**

- `mat` (matrix or data.frame) main input
- `useCol` (integer, length=2) columns to consider/use when looking for duplicated entries

**Value**

matrix with duplicated lines removed

**See Also**

`unlist`

**Examples**

```r
mat <- matrix(1:12, ncol=3)
mat[3,1:2] <- mat[1,1:2]
rmDupl2colMatr(mat)
```

---

**Description**

This function allows indentifying, removing or renaming enumerator tag/name (or remove entire enumerator) from tailing enumerators (eg 'abc_No1' to 'abc_1'). A panel of potential candidates as combination of separator-symbols and separtor text/words will be tested to find if one matches all data. In case the main input is a matrix, all columns will be tested independently to find the first column where one specific combination of separator-symbols and separator text/words is found. Several options exist for the output, the combination of separator-symbols and separtor text/words may be included, too.
Usage

```r
rmEnumeratorName(
  dat,
  nameEnum = c("Number", "No", ",", "Replicate", "Sample"),
  sepEnum = c(" ", ",", ","),
  newSep = ",",
  incl = c("anyCase", "trim2"),
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

dat (character vector or matrix) main input

nameEnum (character) potential enumerator-names

sepEnum (character) potential separators for enumerator-names

newSep (character) potential enumerator-names

incl (character) options to include further variants of the enumerator-names, use "rmEnum" for completely removing enumerator tag/name and digits for different options of trimming names/tags from nameEnum one may use anyCase, trim3 (trimming down to max 3 letters), trim2 (trimming to max 2 letters) or trim1 (trimming down to single letter); trim0 works like trim1 but also includes "", ie no enumerator tag/name in front of the digit(s)

silent (logical) suppress messages

debug (logical) display additional messages for debugging

callFrom (character) allow easier tracking of messages produced

Details

Please note, that checking a variety of different separator text-word and separator-symbols may give an important number of combinations to check. In particular, when automatic trimming of separator text-words is added (eg incl="trim2"), the complexity of associated searches increases quickly. Thus, with large data-sets restricting the content of the arguments nameEnum, sepEnum and (in particular) newSep to the most probable terms/options is suggested to help reducing demands on memory and CPU.

In case the input dat is a matrix and multiple different numerator-types are found, only the first column (from the left) will be treated. If you which to remove/substitute multiple types of enumerators the function rmEnumeratorName must be run independently, see last example below.

Value

This function returns a corrected vector (or matrix), or a list if incl="rmEnumL" containing $dat (corrected data), $pattern (the combination of separator-symbols and separator text/words found), and if input is matrix $column (which column of the input was identified and treated)
See Also
when the exact pattern is known `grep` and `sub` may allow direct manipulations much faster

Examples

```r
xx <- c("hg_Re1","hjRe2_Re2","hk-Re3_Re33")
rmEnumeratorName(xx)
rmEnumeratorName(xx, newSep="--")
rmEnumeratorName(xx, incl="anyCase")

xy <- cbind(a=11:13, b=c("11##11","2_No2","333_samp333"), c=xx)
rmEnumeratorName(xy)
rmEnumeratorName(xy, incl=c("anyCase","trim2","rmEnumL"))

xz <- cbind(a=11:13, b=c("23#11","4#2","567#333"), c=xx)
apply(xz, 2, rmEnumeratorName, sepEnum=c("","-_"), newSep="_", silent=TRUE)
```

---

### rnormW

`rnormW`  
*Normal random number generation with close fit to expected mean and sd*

Description

This function allows creating a vector of random values similar to `rnorm`, but resulting value get recorrected to fit to expected mean and sd. When the number of random values to generate is low, the mean and sd of the resultant values may deviate from the expected mean and sd when using the standard `rnorm` function. In such cases the function `rnormW` helps getting much closer to the expected mean and sd.

Usage

```r
rnormW(  
n,  
mean = 0,  
sd = 1,  
seed = NULL,  
digits = 8,  
silent = FALSE,  
callFrom = NULL  
)
```

Arguments

- **n**  
  (integer, length=1) number of observations. If `length(n) > 1`, the length is taken to be the number required.

- **mean**  
  (numeric, length=1) expected mean
sd  (numeric, length=1) expected sd
seed (integer, length=1) seed for generating random numbers
digits (integer, length=1 or NULL) number of significant digits for output, set to NULL to get all digits
silent (logical) suppress messages
callFrom (character) allow easier tracking of messages produced

Details

For making result reproducible, a seed for generating random numbers can be set via the argument seed. However, with \( n=2 \) the resulting values are ‘fixed’ since no random component is possible at \( n < 3 \).

Value

This function returns a numeric vector of random values

See Also

Normal

Examples

```r
x1 <- (11:16)[-5]
mean(x1); sd(x1)
## the standard way
ra1 <- rnorm(n=length(x1), mean=mean(x1), sd=sd(x1))
## typically the random values deviate (slightly) from expected mean and sd
mean(ra1) -mean(x1)
sd(ra1) -sd(x1)
## random numbers with close fit to expected mean and sd :
ra2 <- rnormW(length(x1), mean(x1), sd(x1))
mean(ra2) -mean(x1)
sd(ra2) -sd(x1)  # much closer to expected value
```

Description

This function returns CV for values in each row (using speed optimized standard deviation). Note : NaN values get replaced by NA.

Usage

```r
rowCVs(dat, autoconvert = NULL)
```
Arguments

- **dat**: (numeric) matrix
- **autoconvert**: (NULL or character) allows converting simple vectors in matrix of 1 row (auto-convert="row")

Value

This function returns a (numeric) vector with CVs for each row of 'dat'

See Also

- colSums, rowGrpCV, rowSds

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10,20)),ncol=10)
head(rowCVs(dat1))
```

---

Description

This function calculates CVs for matrix with multiple groups of data, ie one CV for each group of data. Groups are specified as columns of 'x' in 'grp' (so length of grp should match number of columns of 'x', NAs are allowed)

Usage

```r
rowGrpCV(x, grp, means = NULL, listOutp = FALSE)
```

Arguments

- **x**: numeric matrix where replicates are organized into separate columns
- **grp**: (factor) defining which columns should be grouped (considered as replicates)
- **means**: (numeric) alternative values instead of means by .rowGrpMeans()
- **listOutp**: (logical) if TRUE, provide output as list with $CV, $mean and $n

Value

This function returns a matrix of CV values

See Also

- rowCVs, arrayCV, replPlateCV
Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
head(rowGrpCV(dat1, gr=gl(4,3,labels=LETTERS[1:4])[2:11]))
```

---

**rowGrpMeans**

*rowMeans with distinction of groups (of columns, eg groups of replicates)*

Description

`rowGrpMeans` calculates column-means for matrix with multiple groups of data, ie similar to `rowMeans` but one mean for each group of data. Groups are specified as columns of 'x' in 'grp' (so length of `grp` should match number of columns of 'x', NAs are allowed).

Usage

```
rowGrpMeans(x, grp, na.rm = TRUE)
```

Arguments

- `x` matrix or data.frame
- `grp` (character or factor) defining which columns should be grouped (considered as replicates)
- `na.rm` (logical) a logical value indicating whether NA-values should be stripped before the computation proceeds.

Value

matrix with mean values

See Also

`rowSds, colSums`

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10, 20)), ncol=10)
head(rowGrpMeans(dat1, gr=gl(4, 3, labels=LETTERS[1:4])[2:11]))
```
Description

This function allows easy counting the number of NAs per row in data organized in multiple sub-groups as columns.

Usage

```r
rowGrpNA(mat, grp)
```

Arguments

- `mat` (matrix of data.frame) data to count the number of NAs
- `grp` (character or factor) defining which columns should be grouped (considered as replicates)

Value

matrix with number of NAs per group

See Also

`rowGrpMeans, rowSds, colSums`

Examples

```r
mat2 <- c(22.2, 22.5, 22.2, 22.2, 22.5, 21.5, 22.5, 21.7, 21.5, 22, 22.2, 22.7,
          NA, NA, NA, NA, NA, NA, NA, 21.2, NA, NA, NA, NA,
          NA, 22.6, 23.2, 23.2, 22.4, 22.8, 22.8, NA, 23.3, 23.2, NA, 23.7,
          NA, 23.0, 23.1, 23.0, 23.2, 23.2, NA, 23.3, NA, NA, 23.3, 23.8)
mat2 <- matrix(mat2, ncol=12, byrow=TRUE)
gr4 <- gl(3, 4, labels=LETTERS[1:3])
# overall number of NAs per row
rowSums(is.na(mat2))
# number of NAs per row and group
rowGrpNA(mat2, gr4)
```
rowGrpSds

**Description**

`rowGrpSds` calculate Sd (standard-deviation) for matrix with multiple groups of data, ie one sd for each group of data. Groups are specified as columns of 'x' in 'grp' (so length of grp should match number of columns of 'x', NAs are allowed).

**Usage**

```r
rowGrpSds(x, grp)
```

**Arguments**

- `x`: matrix where replicates are organized into separate columns
- `grp`: (character or factor) defining which columns should be grouped (considered as replicates)

**Value**

This function returns a matrix of sd values

**See Also**

`rowGrpMeans`, `rowCVs`, `rowSEMs`, `sd`

**Examples**

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10, 20)), ncol=10)
head(rowGrpSds(dat1, gr=gl(4,3,labels=LETTERS[1:4])[2:11]))
```

drg

---

rowGrpSums

**Description**

This function calculates row-sums for matrix with multiple groups of data, ie similar to `rowSums` but one summed value for each line and group of data. Groups are specified as columns of 'x' in 'grp' (so length of grp should match number of columns of 'x', NAs are allowed).

**Usage**

```r
rowGrpSums(x, grp, na.rm = TRUE)
```
Arguments

- **x**: matrix or data.frame
- **grp**: (character or factor) defining which columns should be grouped (considered as replicates)
- **na.rm**: (logical) a logical value indicating whether NA-values should be stripped before the computation proceeds.

Value

This function a matrix with row/group sum values

See Also

- rowGrpMeans, rowGrpSds, rowSds, colSums

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10, 20)), ncol=10)
head(rowGrpMeans(dat1, gr=gl(4, 3, labels=LETTERS[1:4])[2:11]))
```

---

**rowMedSds**

sd of median for each row by bootstrap

Description

rowMedSds determines the stand error (sd) of the median for each row by bootstraping each row of 'dat'. Note: requires package `boot`

Usage

```r
rowMedSds(dat, nBoot = 99)
```

Arguments

- **dat**: (numeric) matrix, main input
- **nBoot**: (integer) number if iterations for bootstrap

Value

This functions returns a (numeric) vector with estimated standard errors

See Also

- `boot`

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200) + rep(1:10, 20)), ncol=10)
rowMedSds(dat1); plot(rowSds(dat1), rowMedSds(dat1))
```
rowNormalize

rowNormalize  Row Normalize

Description
This function was designed for normalizing data that is supposed to be particularly similar, like a collection of technical replicates. Thus, initially for each row an independent normalization factor is calculated and the median or mean across all factors will be finally applied to the data. This function has a special mode of operation with higher content of NA values (which may pose problems with other normalization approaches). If the NA-content is higher than the threshold set in sparseLim, a special procedure for sparse data will be applied (iteratively treating subsets of nCombin columns that will be combined in a later step).

Usage
rowNormalize(
  dat,
  method = "median",
  refLines = NULL,
  refGrp = NULL,
  proportMode = TRUE,
  minQuant = NULL,
  sparseLim = 0.4,
  nCombin = 3,
  omitNonAlignable = FALSE,
  maxFact = 10,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat</td>
<td>matrix or data.frame of data to get normalized</td>
</tr>
<tr>
<td>method</td>
<td>(character) may be &quot;mean&quot;,&quot;median&quot; (plus &quot;NULL&quot;,&quot;none&quot;); When NULL or 'none' is chosen the input will be returned as is</td>
</tr>
<tr>
<td>refLines</td>
<td>(NULL or numeric) allows to consider only specific lines of 'dat' when determining normalization factors (all data will be normalized)</td>
</tr>
<tr>
<td>refGrp</td>
<td>(integer) Only the columns indicated will be used as reference, default all columns (integer or colnames)</td>
</tr>
<tr>
<td>proportMode</td>
<td>(logical) decide if normalization should be done by multiplicative or additive factor</td>
</tr>
<tr>
<td>minQuant</td>
<td>(numeric) optional filter to set all values below given value as NA</td>
</tr>
<tr>
<td>sparseLim</td>
<td>(integer) decide at which min content of NA values the function should go in sparse-mode</td>
</tr>
</tbody>
</table>
**rowNormalize**

**Arguments**

- `nCombin` (NULL or integer) used only in sparse-mode (ie if content of NAs higher than content of `sparseLim`): Number of groups of smaller matrixes with this number of columns to be inspected initially; low values (small groups have higher chances of more common elements)
- `omitNonAlignable` (logical) allow omitting all columns which can’t get aligned due to sparseness
- `maxFact` (numeric, length=2) max normalization factor
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) This function allows easier tracking of messages produced

**Details**

Arguments were kept similar with function `normalizeThis` as much as possible. In most cases data get normalized by proportional factors. In case of log2-data (very common in omics-data) normalizing by an additive factor is equivalent to a proportional factor.

This function has a special mode of operation for sparse data (ie containing a high content of NA values). 0-values by themselves will be primarily considered as true measurement outcomes and not as missing. However, by using the argument `minQuant` all values below a given threshold will be set as NA and this may possibly trigger the sparse mode of normalizing.

Note: Using a small value of `nCombin` will give the highest chances of finding sufficient complete combination of columns with sparse data. However, this will also increase (very much) the computational efforts and time required to produce an output.

When using default proportional mode a potential division by 0 could occur, when the initial normalization factor turns out as 0. In this case a small value (default the maximum value of `dat` / 10 will be added to all data before normalizing. If this also creates 0-values in the data this factor will be multiplied by 0.03.

**Value**

This function returns a matrix of normalized data

**See Also**

- `exponNormalize`, `adjBy2ptReg`, `justvsn`

**Examples**

```r
## sparse matrix normalization
set.seed(2); AA <- matrix(rbinom(110,10,0.05), nrow=10)
AA[2,c(2,6,7)] <- 1; AA[3,8] <- 1;
(AA1 <- rowNormalize(AA))
(AA2 <- rowNormalize(AA, minQuant=1)) # set all 0 as NAs
(AA3 <- rowNormalize(AA, refLines=1:6, omitNonAlignable=FALSE, minQuant=1))
```
rowSds

Description

rowSds is speed optimized sd (takes matrix or data.frame and treats each line as set of data for sd equiv to apply(dat,1,sd). NAs are ignored from data unless entire line NA). Speed improvements may be seen at more than 100 lines. Note: NaN instances will be transformed to NA.

Usage

rowSds(dat)

Arguments

dat  matrix (or data.frame) with numeric values (may contain NAs)

Value

numeric vector of sd values

See Also

sd

Examples

set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
rowSds(dat1)

rowSEMs

SEM for each row

Description

This function speed optimized SEM (standard error of the mean) for each row. The function takes a matrix or data.frame and treats each row as set of data for SEM; NAs are ignored from data. Note: NaN instances will be transformed to NA.

Usage

rowSEMs(dat)

Arguments

dat  matrix or data.frame
Value

This function returns a numeric vector with SEM values

See Also

rowSds, colSds, colSums

Examples

```r
set.seed(2016); dat1 <- matrix(c(runif(200)+rep(1:10,20)),ncol=10)
head(rowSEMs(dat1))
```

Usage

```r
sampNoDeMArrayLM(MArrayObj, useComp, groupSep = "-", lstMeans = "means", lstP = c("BH", "FDR", "p.value"), silent = FALSE, debug = FALSE, callFrom = NULL)
```

Arguments

- **MArrayObj** (list or MArray-object) main input
- **useComp** (character or integer) index or name of pairwise-comparison to be addressed
- **groupSep** (character, length=1) separator for pair of names
- **lstMeans** (character, length=1) the list element containing the individual sample names, typically the matrix containing the replicate-mean values for each type of sample, the column-names get used
- **lstP** (character, length=1) the list element containing all pairwise comparisons performed, the column-names get used
- **silent** (logical) suppress messages
- **debug** (logical) additional messages for debugging
- **callFrom** (character) allow easier tracking of messages produced
scaleXY

Scale data to given minimum and maximum

Description

This is a convenient way to scale data to given minimum and maximum without full standardization, i.e., without dividing by the sd.

Usage

scaleXY(x, min = 0, max = 1)

Arguments

x (numeric) vector to rescale
min (numeric) minimum value in output
max (numeric) maximum value in output

Value

vector of rescaled data (in dimensions as input)
searchDataPairs

Search duplicated data over multiple columns, ie pairs of data

Description

searchDataPairs searches matrix for columns of similar data, ie 'duplicate' values in separate columns or very similar columns if realDupsOnly=FALSE. Initial distance measures will be normalized either to diagonale (normRange=TRUE) of 'window' or to the real max distance observed (equal or less than diagonale). Return data.frame with names for sample-pair, percent of identical values (100 for complete identical pair) and relative (Euclidean) distance (ie max dist observed =1.0). Note, that low distance values do not necessarily imply correlating data.

Usage

searchDataPairs(
  dat,
  disThr = 0.01,
  byColumn = TRUE,
  normRange = TRUE,
  altNa = NULL,
  realDupsOnly = TRUE,
  silent = FALSE,
  callFrom = NULL
)

Arguments

dat matrix or data.frame (main input)
disThr (numeric) threshold to decide when to report similar data (applied on normalized distances, low val fewer reported), applied on normalized distances (norm to diagonale of all data for best relative 'unbiased' view)
searchLinesAtGivenSlope

(byColumn) rotates main input by 90 degrees (using t), thus allows to read by rows instead of by columns

(normRange) (logical) normize each columns separately if TRUE

(altNa) (character, default NULL) vector with alternative names (for display)

(realDupsOnly) (logical) if TRUE will consider equal values only, otherwise will also consider very close values (based on argument disThr)

(silent) (logical) suppress messages

(callFrom) (character) allows easier tracking of messages produced

Value

This function returns a data.frame with names of sample-pairs, percent of identical values (100 for complete identical pair) and rel (Euclidean) distance (ie max dist observed =1.0)

See Also

duplicated

Examples

mat <- round(matrix(c(11:40,runif(20)+12,11:19,17,runif(20)+18,11:20), nrow=10), 1)
colnames(mat) <- 1:9
searchDataPairs(mat,disThr=0.05)

searchLinesAtGivenSlope

Search points forming lines at given slope

Description

searchLinesAtGivenSlope searches among set of points (2-dim) those forming line(s) with user-defined slope ('coeff'), ie search optimal (slope-) offset parameter(s) for (regression) line(s) with given slope ('coeff'). Note: larger data-sets : segment residuals to ‘coeff’ & select most homogenous

Usage

searchLinesAtGivenSlope(
  dat,
  coeff = 1.5,
 filtExtr = c(0, 1),
  minMaxDistThr = NULL,
  lmCompare = TRUE,
  indexPoints = TRUE,
  displHist = FALSE,
  displScat = FALSE,
  bestCluByDistRat = TRUE,
Arguments

- **dat**: matrix or data.frame, main input
- **coeff**: (numeric) slope to consider
- **filtExtr**: (integer) lower & upper quantile values, remove points with extreme deviation to offset=0, (if single value: everything up to or after will be used)
- **minMaxDistThr**: (logical) optional minimum and maximum distance threshold
- **lmCompare**: (logical) add’l fitting of linear regression to best results, return offset AND slope based on lm fit
- **indexPoints**: (logical) return results as list with element 'index' specifying retained points
- **displHist**: (logical) display histogram of residues
- **displScat**: (logical) display (simple) scatter plot
- **bestCluByDistRat**: (logical) initial selection of decent clusters based on ratio overallDist/averNeighbDist (or by CV & cor)
- **neighbDiLim**: (numeric) additional threshold for (trimmed mean) neighbour-distance
- **silent**: (logical) suppress messages
- **debug**: (logical) for bug-tracking: more/enhanced messages
- **callFrom**: (character) allow easier tracking of messages produced

Value

This function returns a matrix of line-characteristics (or if indexPoints is TRUE then list (line-characteristics & index & lm-results)

See Also

- **lm**

Examples

```r
set.seed(2016); ra1 <- runif(300)
dat1 <- cbind(x=round(c(1:100+ra1[1:100]/5,4*ra1[1:50]),1),
y=round(c(1:100+ra1[101:200]/5,4*ra1[101:150]),1))
(ll1 <- searchLinesAtGivenSlope(dat1, coeff=1))
```
simpleFragFig

Simple figure showing line from start- to end-sites of edges (or fragments) defined by their start- and end-sites. simpleFragFig draws figure showing start- and end-sites of edges (or fragments).

Description

Simple figure showing line from start- to end-sites of edges (or fragments) defined by their start- and end-sites.

simpleFragFig draws figure showing start- and end-sites of edges (or fragments).

Usage

```
simpleFragFig(
  frag,
  fullSize = NULL,
  sortByHead = TRUE,
  useTit = NULL,
  useCol = NULL,
  dispNa = TRUE,
  useCex = 0.7
)
```

Arguments

- `frag` (matrix) 2 columns defining begin- and end-sites (as integer values)
- `fullSize` (integer) optional max size used for figure (x-axis)
- `sortByHead` (logical) sort by begin-sites (if TRUE) or sort by end-sites
- `useTit` (character) custom title
- `useCol` (character) specify colors, if numeric vector will be considered as score values
- `dispNa` (character) display names of edges (figure may get crowded)
- `useCex` (numeric) expansion factor, see also `par`

Value

matrix with mean values

See Also

`buildTree`, `countSameStartEnd`, `contribToContigPerFrag`.

Examples

```
frag2 <- cbind(beg=c(2,3,7,13,15,7,9,7,3,7,5,7,3),end=c(6,12,8,18,20,20,19,12,12,4,12,7,12,4))
simpleFragFig(frag2,fullSize=21,sortByHead=TRUE)
buildTree(frag2)
```
singleLineAnova 2-factorial Anova on single line of data

Description
This function runs 2-factorial Anova on a single line of data (using `aov` from package `stats`) using a model with two factors (without factor-interaction) and extracts the corresponding p-value.

Usage
```r
singleLineAnova(
  dat,
  fac1,
  fac2,
  inclInteraction = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments
- `dat` numeric vector
- `fac1` (character or factor) vector describing grouping elements of `dat` for first factor, must be of same length as `fac2`
- `fac2` (character or factor) vector describing grouping elements of `dat` for second factor, must be of same length as `fac1`
- `inclInteraction` (logical) decide if factor-interactions (eg synergy) should be included to model
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Value
This function returns the (uncorrected) p for factor 'Pr(>F)' (see `aov`)

See Also
- `aov`, `anova`; for repeated tests using the package `limma` including `lmFit` and eBayes see `test2factLimma`

Examples
```r
set.seed(2012); dat <- round(runif(8),1)
singleLineAnova(dat, gl(2,4), rep(1:2,4))
```
sortBy2CategorAnd1IntCol

Sort matrix by two categorical and one integer columns

Description

This function sorts matrix 'mat' subsequently by categorical and numerical columns of 'mat', ie lines with identical values for category are sorted by numeric value.

Usage

sortBy2CategorAnd1IntCol(mat, categCol, numCol, findNeighb = TRUE, decreasing = FALSE, silent = FALSE, debug = FALSE, callFrom = NULL)

Arguments

mat matrix (or data.frame) from which by 2 columns will be selected for sorting
categCol (integer or character) which columns of 'mat' to be used as categorical columns
numCol (integer or character) which column of 'mat' to be used as integer columns
findNeighb (logical) if 'findNeighb' neighbour cols according to 'numCol' will be identified as groups & marked in new col 'neiGr', orphans marked as NA
decreasing (logical) order of sort
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a sorted matrix (same dimensions as 'mat')

Examples

mat <- cbind(aa=letters[c(3,rep(7:8,3:4),4,4:6,7)],bb=LETTERS[rep(1:5,c(1,3,4,4,1))],
nu=c(23:21,23,21,22,18:12))
mat[3:5,]<-cbind(1)
sortBy2CategorAnd1IntCol(mat,cate=c("bb","aa"),num="nu",findN=FALSE,decr=TRUE)
sortBy2CategorAnd1IntCol(mat,cate=c("bb","aa"),num="nu",findN=TRUE,decr=FALSE)
sortByNRepeated

Make a list of common occurrences sorted by number of repeats

Description

The aim of this function is to count the number of occurrences of words when comparing separate vectors (x, y and z) or from a list (given as x) and to give an output sorted by their frequency. The output lists the various values/words by their frequency, the names of the resulting list-elements indicate number of times the values/words were found repeated.

Usage

sortByNRepeated(
  x,
  y = NULL,
  z = NULL,
  filterIntraRep = TRUE,
  silent = TRUE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

- x (list, character or integer) main input, if list, arguments y and z will not be used
- y (character or integer) supplemental vector to compare with x
- z (character or integer) supplemental vector to compare with x
- filterIntraRep (logical) allow making vectors x, y and z unique before comparing (defaults to TRUE)
- silent (logical) suppress messages
- debug (logical) additional messages for debugging
- callFrom (character) allow easier tracking of messages produced

Details

In order to compare the frequency of values/words between separate vectors or vectors within a list, it is necessary that these have been made unique before calling this function or using filterIntraRep=TRUE.

In case the input is given as list (in x), there is no restriction to the number of vectors to be compared. With very long lists, however, the computational effort increases (like it does when using table).

Value

This function returns a list sorted by number of occurrences. The names of the list indicate the number of repeats.
stableMode

See Also
table, replicateStructure

Examples
sortByNRepeated(x=LETTERS[1:11], y=LETTERS[3:13], z=LETTERS[6:12])
sortByNRepeated(x=LETTERS[1:11], y=LETTERS[c(3:13,5:4)], z=LETTERS[6:12])

stableMode Estimate mode (most frequent value)

Description
Estimate mode, ie most frequent value. In case of continuous numeric data, the most frequent values may not be the most frequently repeated exact term. This function offers various approaches to estimate the mode of a numeric vector. Besides, it can also be used to identify the most frequent exact term (in this case also from character vectors).

Usage
stableMode(
  x,
  method = "density",
  finiteOnly = TRUE,
  bandw = NULL,
  rangeSign = 1:6,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)

Arguments
x (numeric, or character if 'method=’mode’) data to find/estimate most frequent value
method (character) There are 3 options: BBmisc, binning and density (default). If "binning" the function will search context dependent, ie like most frequent class of histogram. Using "binning" mode the search will be refined if either 80 percent of values in single class or >50 percent in single class.
finiteOnly (logical) suppress non-finite values; allows avoiding NULL as result in presence of some Inf values; NA will be ignored in any case
bandw (integer) only used when method="binning" or method="density": defines the number of points to look for density or number of classes used; very "critical” parameter, may change results in strong way. Note: with method="binning": At higher values for “bandw” you will finally loose advantage of histLike-type search of mode!
rangeSign (integer) only used when method="binning": range of numbers used as number of significant values

silent (logical) suppress messages
callFrom (character) allows easier tracking of messages produced
debug (logical) additional messages for debugging

Details

The argument method allows to choose among (so far) 4 different methods available. If "density" is chosen, the most dense region of sqrt(n) values will be chosen; if "binning", the data will be binned (like in histograms) via rounding to a user-defined number of significant values ("rangeSign"). If method is set to "BBmisc", the function computeMode() from package BBmisc will be used. If "mode" is chosen, the first most frequently occuring (exact) value will be returned, if "allModes", all ties will be returned. This last mode also works with character input.

Value

This function returns a numeric vector with value of mode, the name of the value indicates it's position

See Also

computeMode() in package BBmisc

Examples

set.seed(2012); dat <- round(c(rnorm(50), runif(100)), 3)
stableMode(dat)

---

standardW

**Standardize (scale) data**

Description

This functions work similar to scale, however, it evaluates the entire input and not column-wise (and independetly as scale does). With Standardizing we speak of transforming the data to end up with mean=0 and sd=1. Furthermore, in case of 3-dim arrays, this function returns also an object with the same dimensions as the input.

Usage

```r
standardW(
  mat,  
  byColumn = FALSE,  
  na.rm = TRUE,  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```
Arguments

mat (matrix, data.frame or array) data that need to get standardized.
byColumn (logical) if TRUE the function will be run independently over all columns such as apply(mat,2,standardW)
na.rm (logical) if NAs in the data don’t get ignored via this argument, the output will be all NA
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced

Value

This function returns a vector of rescaled data (in dimensions as input)

See Also

scale

Examples

dat <- matrix(2*round(runif(100),2), ncol=4)
mean(dat); sd(dat)

dat2 <- standardW(dat)
apply(dat2, 2, sd)
summary(dat2)

dat3 <- standardW(dat, byColumn=TRUE)
apply(dat3, 2, sd)
summary(dat3)
mean(dat2); sd(dat2)

stdErrMedBoot Standard error of median by boot-strap

Description

stdErrMedBoot estimate standard error of median by boot-strap approach. Note: requires package boot

Usage

stdErrMedBoot(x, nBoot = 99)
Arguments

x (numeric) vector to estimate median and its standard error
nBoot (integer) number for iterations

Value

This function returns a (numeric) vector with estimated standard error

See Also

boot

Examples

set.seed(2014); ra1 <- c(rnorm(9,2,1),runif(8,1,2))
rat1 <- ratioAllComb(ra1[1:9],ra1[10:17])
median(rat1); stdErrMedBoot(rat1)

summarizeCols

Summarize columns (as median, mean, min, last or other methods)

Description

summarizeCols summarizes all columns of matrix (or data.frame). In case of text-columns the
sorted middle (~median) will be given, unless 'maxLast', 'minLast', 'maxLast', 'maxAbsLast' or
'minLast' .. consider only last column of 'matr': choose from all columns the line where (max of)
last col is at min; 'medianComplete' or 'meanComplete' considers only lines/rows where no NA
occur (NA have influence other columns!)

Usage

summarizeCols(

  matr,
  meth = "median",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

matr data.frame matrix of data to be summarized by column (may do different
method for text and numeric columns)

meth (character) summarization method (eg 'maxLast', 'minLast', 'maxLast', 'maxAbsLast',
'minLast', 'medianComplete' or 'meanComplete')

silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced
sumNAperGroup

Value

vector with summary for each column

See Also

rowMeans in colSums

Examples

t1 <- matrix(round(runif(30,1,9)), nc=3); rownames(t1) <- letters[c(1:5,3:4,6:4)]
summarizeCols(t1, me="median")
t(sapply(by(t1,rownames(t1), function(x) x), summarizeCols, me="maxLast"))
t3 <- data.frame(ref=rep(11:15,3), tx=letters[1:15],
  matrix(round(runif(30,-3,2),1), ncol=2), stringsAsFactors=FALSE)
by(t3,t3[,1], function(x) x)
t(sapply(by(t3,t3[,1], function(x) x), summarizeCols, me="maxAbsLast"))

Description

This function will count the number of NAs per group (defined by argument grp) while summing over all lines of a matrix or data.frame. The row-position has no influence on the counting. Using the argument asRelative=TRUE the result will be given as (average) number of NAs per row and group.

Usage

sumNAperGroup(
  x,
  grp,
  asRelative = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

x matrix or data.frame which may contain NAs
grp factor describing which column of `dat` belongs to which group
asRelative (logical) return as count of NAs per row and group
silent (logical) suppress messages
debug (logical) additional messages for debugging
callFrom (character) allow easier tracking of messages produced
Value

This function returns an integer vector with count of NAs per group

See Also

NA, filter NAs by line presenceFilt

Examples

```r
mat <- matrix(1:25, ncol=5)
matrix[lower.tri(mat)] <- NA
sumNAperGroup(mat, rep(1:2,c(3,2)))
sumNAperGroup(mat, rep(1:2,c(3,2)), asRelative=TRUE)
```

sysDate

System-date (compressed format)

Description

This function returns current date (based on Sys.Date) in different format options.

Usage

```r
sysDate(style = "univ1")
```

Arguments

style (character) choose style (default 'univ1' for very compact style)

Details

Multiple options for formatting exist : 'univ1' or 'wr' ... (default) compact style using day, first 3 letters of English name of month (lowercaps) and last 2 letters of year as ddmmmyy, eg 14jun21
'univ2' ... as ddMmmyy, eg 14Jun21
'univ3' ... as ddMonthyyyy, eg 14June2021
'univ4' ... as ddmmonyyyy, eg 14june2021
'univ5' ... as yyyy-mm-dd (output of Sys.Date()), eg 2021-06-14
'univ6' ... as yyyy-number of day (in year), eg 2021-165
'local1' ... compact style using day, first 3 letters of current locale name of month (not necessarily unique!) and last 2 letters of year as ddmmyy, eg 14jui21
'local2' ... as ddmmyy, month based on current locale (not necessarily unique !), eg 14Jui21
'local3' ... as ddmmonyyyy, month based on current locale , eg 14Juin2021
'local4' ... as ddmmontyyyy, month based on current locale , eg 14juin2021
'local5' ... as dd-month-yyyy, month based on current locale , eg 14-juin-2021
'local6' ... as yyyymonthddd, month based on current locale , eg 2021juin14
Value
character vector with formatted date

See Also
date, Sys.Date and Sys.time,

Examples
sysDate()

tableToPlot

Print matrix-content as plot

Description
This function prints all columns of matrix in plotting region for easier inclusion to reports (default values are set to work for output as A4-sized pdf). It was made for integrating listings of text to graphical output to devices like png, jpeg or pdf.

Usage
tableToPlot(
  matr,
  colPos = c(0.05, 0.35, 0.41, 0.56),
  useCex = 0.7,
  useAdj = c(0, 1, 1, 0),
  titOffS = 0,
  useCol = 1,
  silent = FALSE,
  callFrom = NULL
)

Arguments

matr          (matrix) main (character) matrix to display
colPos        (numeric) postion of columns on x-scale (from 0 to 1)
useCex        (numeric) cex expansion factor for size of text (may be different for each column)
useAdj        (numeric) left/center/right alignment for text (may be different for each column)
titOffS       (numeric) offset for title line (relative to 'colPos')
useCol        color specification for text (may be different for each column)
silent        (logical) suppress messages
callFrom      (character) allow easier tracking of message(s) produced
Details

This function was initially designed for listings with small/medium 1st col (eg couner or index), 2nd & 3rd col small and long 3rd col (like file paths). Obviously, the final number of lines one can pack and still read correctly into the graphical output depends on the size of the device (on a pdf of size A4 one can pack up to apr. 11O lines). Of course, Sweave, combined with LaTeX, provides a powerful alternative for wrapping text to pdf-output (and further combining text and graphics). Note: The final result on pdf devices may vary depending on screen-size (ie with of current device), the parameters ’colPos’ and ’titOffS’ may need some refinements. Note: In view of typical page/figure layouts like A4, the plotting region will be split to avoid too wide spacing between rows with less than 30 rows.

Value

This function returns NULL (no R-object returned), print ’plot’ in current device only

See Also

Sweave for more flexible framework

Examples

## as example let's make a listing of file-names and associated parameters in current directory
mat <- dir()
mat <- cbind(no=1:length(mat),fileName=mat,mode=file.mode(mat),
    si=round(file.size(mat)/1024),path=getwd())

## Now, we wrap all text into a figure (which could be saved as jpg, pdf etc)
tableToPlot(mat[,1],colPos=c(0.01,0.4,0.46,0.6),titOffS=c(0.05,-0.03,-0.01,0.06))
tableToPlot(mat,colPos=c(0.016,0.36,0.42,0.75),useAdj=0.5,titOffS=c(-0.01,0,-0.01,0,-0.1))

Description

The aim of this function is to provide convenient acces to two-factorial (linear) testing withing the framework of makeMAList including the empirical Bayes shrinkage. The input data 'datMatr' which should already be organized as limma-type MAList, eg using using makeMAList. Note: This function uses the Bioconductor package limma.

Usage

test2factLimma(
    datMatr,
    fac1,
    fac2,
    testSynerg = TRUE,
    testOrientation = "=",
    addResults = c("lFdr", "FDR", "Mval", "means"),
)
addGenes = NULL,
silent = FALSE,
callFrom = NULL,
debug = FALSE
)

Arguments

datMatr: matrix or data.frame with lines as independent series of measures (eg different genes)

fac1: (character or factor) vector describing grouping elements of each line of 'datMatr' for first factor, must be of same length as fac2

fac2: (character or factor) vector describing grouping elements of each line of 'datMatr' for second factor, must be of same length as fac1

testSynerg: (logical) decide if factor-interactions (eg synergy) should be included to model

testOrientation: (character) default (or any non-recognized input) '='; otherwise either '>'; 'greater'; 'sup'; 'upper' or '<'; 'inf'; 'lower'

addResults: (character) vector defining which types of information should be included to output, may be 'lfdr', 'FDR' (for BY correction), 'Mval' (M values), 'means' (matrix with mean values for each group of replicates)

addGenes: (matrix or data.frame) additional information to add to output

silent: (logical) suppress messages

callFrom: (character) allow easier tracking of messages produced

debug: (logical) additional messages for debugging

Value

This function returns an object of class "MArrayLM" (from limma) containing/enriched by the testing results

See Also

makeMAList, single line testing lmFit and the eBayes-family of functions in package limma

Examples

set.seed(2014)
dat0 <- rnorm(30) + rep(c(10,15,19,20),c(9,8,7,6))
fa <- factor(rep(letters[1:4],c(9,8,7,6)))
dat2 <- data.frame(facA=rep(c("-","A","-","A"), c(9,8,7,6)),
facB= rep(c("-","-","B","B"), c(9,8,7,6)), dat1=dat0, dat2=runif(30))
grpNa <- sub("-", "", sub("\"","", apply(dat2[,1:2], 1, paste, collapse="")))
test2f <- test2factLimma(t(dat2[,3:4]), dat2$facA, dat2$facB, testS=FALSE)
test2f
# Now you can easily summarize results using topTable from limma
if(requireNamespace("limma", quietly=TRUE)) {
  library(limma)
transpGraySca

Make single vector gray-gradient

Description

This function helps making gray-gradients. Note: The resulting color gradient does not seem linear to the human eye, you may try gray.colors instead.

Usage

transpGraySca(startGray = 0.2, endGrey = 0.8, nSteps = 5, transp = 0.3)

Arguments

- startGray (numeric) gray shade at start
- endGrey (numeric) gray shade at end
- nSteps (integer) number of levels
- transp (numeric) transparency alpha

Value

character vector (of same length as x) with color encoding

See Also

gray.colors

Examples

layout(1:2)
col1 <- transpGraySca(0.8, 0.3, 7, 0.9)
pie(rep(1, length(col1)), col=col1, main="from transpGraySca")
col2 <- gray.colors(7, 0.9, 0.3, alph=0.9)
pie(rep(1, length(col2)), col=col2, main="from gray.colors")
treatTxtDuplicates  Locate duplicates in text and make non-redundant

Description

treatTxtDuplicates locates duplicates in character-vector 'x' and return list (length=3) : with $init (initial), $nRed .. non-redundant text by adding number at end or beginning, and $nrLst .. list-version with indexes per unique entry. Note : NAs (if multiple) will be renamed to NA_1, NA_2

Usage

treatTxtDuplicates(
  x,
  atEnd = TRUE,
  sep = "_",
  onlyCorrectToUnique = FALSE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

  x  (character) vector with character-entries to identify (and remove) duplicates
  atEnd  (logical) decide location of placing the counter (at end or at beginning of ID)  (see correctToUnique)
  sep  (character) separator to add before counter when making non-redundant version
  onlyCorrectToUnique  (logical) if TRUE, return only vector of non-redundant
  silent  (logical) suppress messages
  debug  (logical) additional messages for debugging
  callFrom  (character) allow easier tracking of messages produced

Value

  list with $init, $nRed, $nrLst

See Also

  For simple correction use correctToUnique

Examples

treatTxtDuplicates(c("li0",NA,rep(c("li2","li3"),2)))
correctToUnique(c("li0",NA,rep(c("li2","li3"),2)))
triCoord

**Pairwise x,y combinations**

**Description**

triCoord gets pairwise combinations for 'n' elements; returns matrix with x & y coordinates to form all pairwise groups for 1:n elements

**Usage**

triCoord(n, side = "upper")

**Arguments**

- **n** (integer) number of elements for making all pair-wise combinations
- **side** (character) "upper" or "lower"

**Value**

2-column matrix wiyh indexes for all pairwise combinations of 1:n

**See Also**

lower.tri or upper.tri, simpler version upperMaCoord

**Examples**

triCoord(4)

---

trimRedundText

**Trim redundant text**

**Description**

This function allows trimming/removing redundant text-fragments (redundant from head or tail) out of character vector 'txt'.

**Usage**

trimRedundText(

txt,
minNchar = 1,
side = "both",
spaceElim = FALSE,
silent = TRUE,
callFrom = NULL,
debug = FALSE
)

Arguments

- **txt**: character vector to be treated
- **minNchar**: (integer) minimum number of characters that must remain
- **side**: (character) may be either 'both', 'left' or 'right'
- **spaceElim**: (logical) optional removal of any heading or tailing white space
- **silent**: (logical) suppress messages
- **callFrom**: (character) allow easier tracking of messages produced
- **debug**: (logical) display additional messages for debugging

Value

This function returns a modified character vector

See Also

Inverse: Find/keep common text *keepCommonText*; you may also look for related functions in package *stringr*

Examples

```r
txt1 <- c("abcd_ccc","bcd_ccc","cde_ccc")
trimRedundText(txt1, side="right")  # trim from right

txt2 <- c("ddd_ab","ddd_bcd","ddd_cde")
trimRedundText(txt2, side="left")   # trim from left
```

tTestAllVal

Run t.test on each indiv value of x against all its neighbours (=remaining values of same vector) in order to test if this value is likely to belong to vector x. This represents a repeated leave-one-out testing. Multiple choices for multiple testing correction are available.

Usage

```r
tTestAllVal(
  x,
  alph = 0.05,
  alternative = "two.sided",
  p.adj = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```
Arguments

- **x**: matrix or data.frame
- **alph**: (numeric) threshold alpha (passed to `t.test`)
- **alternative**: (character) will be passed to `t.test` as argument 'alternative', may be "two.sided"...
- **p.adj**: (character) multiple test correction: may be NULL (no correction), "BH", "BY", "holm", "hochberg" or "bonferroni" (but not 'fdr' since this may be confounded with local false discovery rate), see `p.adjust`
- **silent**: (logical) suppress messages
- **debug**: (logical) additional messages for debugging
- **callFrom**: (character) allow easier tracking of messages produced

Value

This function returns a numeric vector with p-values or FDR (depending on argument `p.adj`)

See Also

- `t.test`, `p.adjust`

Examples

```r
set.seed(2016); x1 <- rnorm(100)
allTests1 <- tTestAllVal(x1)
hist(allTests1,breaks="FD")
```

Description

The aim of this function is to provide help in automatically harmonizing enumerators at the end of sample-names. When data have same grouped setup/design, many times this is reflected in their names, eg 'A_sample1', 'A_sample2' and 'B_sample1'. However, human operators may use multiple similar (but not identical) ways of expressing the same meaning, eg writing 'A_Samp_1'. This function allows testing a panel of different extensions of enumerators and (if recognized) to replace them by a user-defined standard text/enumerator. Please note that the more recent function `rmEnumeratorName` offers better/more flexible options.

Usage

```r
unifyEnumerator(
  x,
  refSep = "_",
  baseSep = c("\", "\", "\".),
  suplEnu = c("Repl", "Rep", "R", "Number", "No", "Sample", "Samp"),
)```
Arguments

Arguments

- `x` (character) main input
- `refSep` (character) separator for output
- `baseSep` (character) basic separators to test (you have to protect special characters)
- `suplEnu` (character) additional text
- `stringentMatch` (logical) decide if enumerator text has to be found in all instances or only once
- `silent` (logical) suppress messages
- `debug` (logical) display additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced

Details

This function has been developed for matching series of the same samples passing in parallel through different evaluation software (see R package wrProteo). The way human operators may name things may easily leave room for surprises and this function allows testing only a limited number of common ways of writing. Thus, in any case, the user is advised to inspect the results by eye and - if needed- to adjust the parameters.

Basically enumerator separators can be constructed by combing a base-separator `baseSep` (like `-`, `\_` etc) and an enumerator-abbreviation `suplEnu`. Then, all possible combinations will be tested if they occur in the text `x`. Furthermore, the text searched has to be followd by on or multiple digits at the end of text-entry (decimal comma-separators etc are not allowed). Thus, if there is other ‘free text’ following to the right after the enumerator-text this function will not find any enumerators to replace.

The argument `stringentMatch` allows defining if this text has to be found in all text-entries of `x` or just one of them. Whe using `stringentMatch=FALSE` there is risk that other text not meant to design enumerators may be picked up and modified.

Please note, that with large data-sets (ie many columns) testing/checking a larger panel of enumerator-abbreviations may result in slower performance. In cases of larger data-sets it may be more effective to first study the data and then run simple subsitions using sub targeted for this very case.

Value

This function returns a character vector of same length as input `x`, with it’s content as adjusted enumerators

See Also

rmEnumeratorName for better/more flexible options; grep or sub(), etc if exact and consistent patterns are known
Examples

```r
unifyEnumerator(c("ab-1","ab-2","c-3"))
unifyEnumerator(c("ab-R1","ab-R2","c-R3"))
unifyEnumerator(c("ab-1","c3-2","dR3"), strin=FALSE);
```

---

**uniqCountReport**  
*Report number of unique and redundant elements (optional figure)*

**Description**

Make report about number of unique and redundant elements of vector 'dat'. Note: fairly slow for long vectors!!

**Usage**

```r
uniqCountReport(
  dat, 
  frL = NULL, 
  plotDispl = FALSE, 
  tit = NULL, 
  col = NULL, 
  radius = 0.9, 
  sizeTo = NULL, 
  clockwise = FALSE, 
  silent = FALSE, 
  debug = FALSE, 
  callFrom = NULL 
)
```

**Arguments**

- `dat` (character or numeric vector) main input where number of unique (and redundant) should be determined
- `frL` (logical) optional (re-)introducing results from duplicated to shorten time of execution
- `plotDispl` (logical) decide if pie-type plot should be produced
- `tit` (character) optional title in plot
- `col` (character) custom colors in pie
- `radius` (numeric) radius passed to pie
- `sizeTo` (numeric or character) optional reference group for size-population relative adjusting overall surface of pie
- `clockwise` (logical) argument passed to pie
- `silent` (logical) suppress messages
- `debug` (logical) additional messages for debugging
- `callFrom` (character) allow easier tracking of messages produced
Value

vector with counts of \( n \) (total), \( n \text{Unique} \) (wo any repeated), \( n \text{HasRepeated} \) (first of repeated), \( n \text{Redundant} \), optional figure

See Also

correctToUnique, unique

Examples

```
layout(1:2)
uniqCountReport(rep(1:7,1:7),plot=TRUE)
uniqCountReport(rep(1:3,1:3),plot=TRUE,sizeTo=rep(1:7,1:7))
```

```
upperMaCoord(n)
```

Arguments

\( n \) (integer) number of elements for making all pair-wise combinations

Value

2-column matrix with indexes for all pairwise combinations of 1:\( n \)

See Also

lower.tri, more evolved version triCoord

Examples

```
upperMaCoord(4)
```
withinRefRange  Check for values within range of reference

Description

withinRefRange checks which values of numeric vector 'x' are within range +/- 'fa' x 'ref' (ie within range of reference).

Usage

withinRefRange(x, fa, ref = NULL, absRef = TRUE, asInd = FALSE)

Arguments

x    matrix or data.frame
fa   (numeric) absolute or relative tolerance value (numeric, length=1), interpreted according to 'absRef' as absolute or relative to 'x'(ie fa*ref)
ref  (numeric) (center) reference value for comparison (numeric, length=1), if not given mean of 'x' (excluding NA or non-finite values) will be used
absRef (logical) return result as absolute or relative to 'x'(ie fa*ref)
asInd (logical) if TRUE return index of which values of 'x' are within range, otherwise return values if 'x' within range

Value

numeric vector (containing only the values within range of reference)

Examples

## within 2.5 +/- 0.7
withinRefRange(-5:6,fa=0.7,ref=2.5)
## within 2.5 +/- (0.7*2.5)
withinRefRange(-5:6,fa=0.7,ref=2.5,absRef=FALSE)

writeCsv  Write (and convert) csv files

Description

This functions is absed on write.csv allows for more options when writing data into csv-files. The main input may be given as R-object or read from file 'input'. Then, one can (re-)write using specified conversions. An optional filter to select columns (column-name specified via 'filterCol') is available. The output may be simultaneaously written to multiple formats, as specified in 'expTy', tabulation characters may be converted to avoid accidentally split/shift text to multiple columns. Note: Mixing '.' and ',' as comma separators via text-columns or fused text&data may cause problems lateron, though.
writeCsv

Usage

writeCsv(
  input,
  inPutFi = NULL,
  expTy = c("Eur", "US"),
  importTy = "Eur",
  filename = NULL,
  quote = FALSE,
  filterCol = NULL,
  replMatr = NULL,
  returnOut = FALSE,
  SYLKprevent = TRUE,
  digits = 22,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

Arguments

input either matrix or data.frame
inPutFi (character or NULL) file-name to be read (format as US or Euro-type may specified via argument importTy)
expTy (character) 'US' and/or 'Eur' for separator and decimal type in output
importTy (character) default 'Eur' (otherwise set to 'US')
filename (character) optional new file name(s)
quote (logical) will be passed to function write.csv
filterCol (integer or character) optionally, to export only the columns specified here
replMatr optional, matrix (1st line:search, 2nd li:use for replacing) indicating which characters need to be replaced
returnOut (logical) return output as object
SYLKprevent (logical) prevent difficulty when opening file via Excel. In some cases Excel presumes (by error) the SYLK format and produces an error when trying to open files: To prevent this, if necessary, the 1st column-name will be changed from 'ID' to 'Id'.
digits (integer) limit number of signif digits in output (ie file)
silent (logical) suppress messages
debug (logical) for bug-tracking: more/enhanced messages
callFrom (character) allow easier tracking of messages produced

Value

This function writes a file to disk and returns NULL unless returnOut=TRUE
See Also

write.csv in write.table, batch reading using this package readCsvBatch

Examples

dat1 <- data.frame(ini=letters[1:5],x1=1:5,x2=11:15,t1=c("10,10","20.20","11,11","21,21","33,33"),
t2=c("10,11","20.21","kl;kl","az,az","ze.ze"))
fiNa <- file.path(tempdir(), paste("test",1:2,".csv",sep=""))
writeCsv(dat1, filename=fiNa[1])
dir(path=tempdir(), pattern="cs")

(writeCsv(dat1, replM=rbind(bad=c(";",""), replBy="--"), expTy=c("Eur"),
          returnOut=TRUE, filename=fiNa[2]))

XYToDiffPpm

Express difference as ppm

Description

This function transforms offset (pairwise-difference) between 'x' & 'y' to ppm (as normalized
difference ppm, parts per million, ie (x-y)/y ). This type of expressioning differences is used eg
in mass-spectrometry.

Usage

XYToDiffPpm(x, y, nSign = NULL, silent = FALSE, debug = FALSE, callFrom = NULL)

Arguments

x (numeric) typically for measured variable

y (numeric) typically for theoretical/expected value (vector must be of same length
as 'x')

nSign (integer) number of significant digits in output

silent (logical) suppress messages

debug (logical) additional messages for debugging

callFrom (character) allow easier tracking of messages produced

Value

This function returns a numeric vector of (ratio-) ppm values

See Also

ratioToPpm for classical ppm
Examples

```r
set.seed(2017); aa <- runif(10,50,900)
cbind(x=aa,y=aa+1e-3,ppm=XYToDiffPpm(aa,aa+1e-3,nSign=4))
```
Index

* character
  - pasteC, 192
  - addLetterWoLast, 7
  - allRatioMatr1to2, 8
  - allRatios, 9
  - arrLstMean, 9
  - arrLstSEM, 10
  - asDF2, 11
  - breakInSer, 12
  - bringToCtr, 13
  - checkArgNa, 13
  - checkConsistentArrList, 14
  - checkConv2Vect, 15
  - checkFactor, 16
  - checkFileNameExtensions, 17
  - checkLegendLoc, 17
  - checkLmConfInt, 18
  - checkRegrArguments, 19
  - chooseGrpCol, 19
  - combineListAnnot, 20
  - compareByDiff, 21
  - compareByLogRatio, 22
  - compareByPPM, 22
  - complCols, 23
  - composeCallName, 24
  - convertMatrToNum, 25
  - convertNa, 25
  - corDupItemsByIncrem, 26
  - cutAtSearch, 27
  - cutStr, 28
  - datSlope, 28
  - extrNAneighb, 29
  - extrNumHeadingCap, 30
  - extrNumHeadingSepChar, 31
  - filtSize, 33
  - filterNetw, 31
  - filterSw, 32
  - findBorderOverlaps, 34
  - firstMin, 35
  - fuse2ArrBy2ndDim, 35
  - getAmean, 36
  - getAmean2, 37
  - getMValue2, 37
  - growTree, 38
  - insp1dimByClustering, 39
  - inspectHeader, 40
  - keepCenter1d, 41
  - keepFiniteCol, 42
  - mayBeNum, 43
  - medianSpecGrp, 43
  - mergeMatrices, 44
  - minDif, 45
  - neighbDis, 46
  - normConstSlope, 48
  - normalize, 47
  - offCenter, 49
  - pasteCols, 49
  - plotCountPie, 50
  - plusLowerCaps, 51
  - predRes, 52
  - raiseColLowest, 52
  - removeCol, 53
  - removeEmptyCol, 54
  - replSpecChar, 55
  - retain1stPart, 56
  - rowGrpCV, 56
  - rowGrpMeans, 57
  - rowGrpSds, 58
  - rowGrpSums, 58
  - rowNorm, 59
  - rowNormFact, 60
  - scale01, 62
  - scaleSpecGrp, 62
  - scaleXY, 63
  - seqCutStr, 64
  - setLowestTo, 65
  - sortMid, 65
  - stackArray, 66
INDEX

.. summarizeCols, 67
.... trimFromEnd, 68
.... trimFromStart, 68
.... trimLeft, 69
.... trimRight, 70
.... uniqueWName, 71
.... vector2Matr, 72
addBeforeFileExtension, 72
adjBy2ptReg, 73, 183, 184, 208, 209, 226
anova, 234
aov, 234
append, 19, 29, 75
appendNR, 74
array, 36, 187
arrayCV, 57–59, 75, 215, 220
as.data.frame, 12
asSepList, 76, 191
boot, 89, 224, 240
buildTree, 38, 77, 101, 110, 233
cbind, 79, 128, 151, 154
cbindNR, 78
chartr, 14, 51
checkAvSd, 79
checkGrpOrder, 80, 83, 186
checkGrpOrderSEM, 81, 81
checkSimValueInSer, 21–23, 66, 82, 87, 114,
131, 133, 156
checkStrictOrder, 83, 186
checkVectLength, 84
cleanReplicates, 85
closeMatchMatrix, 86, 133
coinPermTest, 88
colMedSds, 89
colorAccording2, 20, 89
colSds, 91, 228
colSums, 146, 220–222, 224, 228, 241
combinatIntTable, 91
combineAsN, 92
combineByEitherFactor, 94
combineOverlapInfo, 95
combineRedBasedOnCol, 96, 180, 181
combineReplFromListToMatr, 97
combineSingleT, 98
combn, 92, 211
completeArrLst, 99
cconfInt, 100
confint, 100
contribToContigPerFrag, 78, 101, 110, 233
conv01toColNa, 102
convColorToTransp, 102
convMatr2df, 43, 103
convToNum, 104, 160
coordOfFilt, 106
correctToUnique, 33, 51, 107, 126, 130, 142,
180, 181, 247, 253
correctWinPath, 108
countCloseToLimits, 21–23, 109, 131
countSameStartEnd, 78, 110, 233
cut, 90, 112
cutArrayInCluLike, 10, 11, 15, 21, 111
cutAtMultSites, 8, 111, 176, 177
cutToNgrp, 112
date, 243
diff, 49, 113, 114, 169
diffCombin, 113
diffPPM, 114
dist, 12, 13, 46, 54, 66, 191
duplicated, 26, 33, 126, 130, 135, 136, 180,
213, 231
elemCloseCoord, 115
equLenNumber, 15, 116
exclExtrValues, 116
exponNormalize, 118, 183, 184, 226
extr1chan, 98, 99, 119
extractLast2numericParts, 120
extrColsDeX, 121, 126
extrNumericFromMatr, 122
extrSpcText, 122
factor, 16, 55
fdrtool, 196
file.path, 108
filt3dimArr, 33, 124
filterLiColDeList, 33, 124, 125
filterList, 33, 121, 124, 126
filterNetw, 32, 127, 190
filtSizeUniq, 33, 107, 129
findCloseMatch, 21–23, 86, 87, 109, 115,
130, 132, 133
findRepeated, 97, 131, 142, 181
findSimilFrom2sets, 132
findUsableGroupRange, 134
firstLineOfDat, 134, 181
<table>
<thead>
<tr>
<th>Function</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstOfRepeated</td>
<td>30, 50, 115, 135, 135, 136, 137, 141, 179–181</td>
</tr>
<tr>
<td>firstOfRepLines</td>
<td>79, 97, 117, 136, 136, 181</td>
</tr>
<tr>
<td>fisher.test</td>
<td>198</td>
</tr>
<tr>
<td>fread</td>
<td>202, 203</td>
</tr>
<tr>
<td>fuseAnnotMatri</td>
<td>137</td>
</tr>
<tr>
<td>fuseCommonListElem</td>
<td>138</td>
</tr>
<tr>
<td>fusePairs</td>
<td>139</td>
</tr>
<tr>
<td>get1stOfRepeatedByCol</td>
<td>117, 131, 132, 140, 179, 181</td>
</tr>
<tr>
<td>getValuesByUnique</td>
<td>141</td>
</tr>
<tr>
<td>getDataUrl</td>
<td>142</td>
</tr>
<tr>
<td>gray.colors</td>
<td>246</td>
</tr>
<tr>
<td>grep</td>
<td>8, 9, 27, 30, 121, 145, 151, 157, 160, 173, 218, 251</td>
</tr>
<tr>
<td>head</td>
<td>40</td>
</tr>
<tr>
<td>htmlSpecCharConv</td>
<td>143</td>
</tr>
<tr>
<td>is.finite</td>
<td>42, 55</td>
</tr>
<tr>
<td>justvsn</td>
<td>183, 184, 226</td>
</tr>
<tr>
<td>keepCommonText</td>
<td>68–71, 144, 249</td>
</tr>
<tr>
<td>kmeans</td>
<td>211</td>
</tr>
<tr>
<td>legend</td>
<td>18</td>
</tr>
<tr>
<td>levIndex</td>
<td>146</td>
</tr>
<tr>
<td>linModelSelect</td>
<td>147</td>
</tr>
<tr>
<td>linRegrParamAndPVal</td>
<td>149</td>
</tr>
<tr>
<td>listBatchReplace</td>
<td>150</td>
</tr>
<tr>
<td>listGroupsByNames</td>
<td>151</td>
</tr>
<tr>
<td>lm</td>
<td>150, 153, 198, 232</td>
</tr>
<tr>
<td>lmFit</td>
<td>126, 171, 172, 234, 245</td>
</tr>
<tr>
<td>lmSelClu</td>
<td>152</td>
</tr>
<tr>
<td>LocationTests</td>
<td>88</td>
</tr>
<tr>
<td>lower.tri</td>
<td>248, 253</td>
</tr>
<tr>
<td>lrbind</td>
<td>19, 29, 75, 153</td>
</tr>
<tr>
<td>makeMAList</td>
<td>8, 9, 36–38, 154, 244, 245</td>
</tr>
<tr>
<td>makeNRedMatr</td>
<td>155</td>
</tr>
<tr>
<td>MAplotW</td>
<td>229</td>
</tr>
<tr>
<td>match</td>
<td>157, 158, 175, 186</td>
</tr>
<tr>
<td>matchMatrixLinesToRef</td>
<td>156</td>
</tr>
<tr>
<td>matchNamesWithReverseParts</td>
<td>158</td>
</tr>
<tr>
<td>matchSmpToPairw</td>
<td>159</td>
</tr>
<tr>
<td>matr2list</td>
<td>160</td>
</tr>
<tr>
<td>matrix</td>
<td>25, 72</td>
</tr>
<tr>
<td>merge</td>
<td>45, 138, 162–168</td>
</tr>
<tr>
<td>mergeMatrices</td>
<td>45, 161, 163</td>
</tr>
<tr>
<td>mergeMatrixList</td>
<td>45, 162, 162</td>
</tr>
<tr>
<td>mergeSelCol</td>
<td>163, 166</td>
</tr>
<tr>
<td>mergeSelCol3</td>
<td>164, 165</td>
</tr>
<tr>
<td>mergeVectors</td>
<td>166</td>
</tr>
<tr>
<td>mergeW2</td>
<td>167</td>
</tr>
<tr>
<td>minDiff</td>
<td>169</td>
</tr>
<tr>
<td>moderTest2grp</td>
<td>126, 147, 148, 170, 172</td>
</tr>
<tr>
<td>moderTestXgrp</td>
<td>147, 148, 160, 171, 229</td>
</tr>
<tr>
<td>multiCharReplace</td>
<td>173</td>
</tr>
<tr>
<td>multiMatch</td>
<td>174</td>
</tr>
<tr>
<td>NA</td>
<td>242</td>
</tr>
<tr>
<td>na.fail</td>
<td>176, 215</td>
</tr>
<tr>
<td>naOmit</td>
<td>175, 215</td>
</tr>
<tr>
<td>nchar</td>
<td>30, 31, 34, 53, 65, 178</td>
</tr>
<tr>
<td>nFragments</td>
<td>112, 176</td>
</tr>
<tr>
<td>nFragments0</td>
<td>112, 177</td>
</tr>
<tr>
<td>nNonNumChar</td>
<td>177</td>
</tr>
<tr>
<td>Node</td>
<td>78</td>
</tr>
<tr>
<td>nonAmbiguousMat</td>
<td>178</td>
</tr>
<tr>
<td>nonAmbiguousNum</td>
<td>30, 50, 79, 136, 137, 141, 179</td>
</tr>
<tr>
<td>nonRedDataFrame</td>
<td>180</td>
</tr>
<tr>
<td>nonRedundLines</td>
<td>181</td>
</tr>
<tr>
<td>Normal</td>
<td>219</td>
</tr>
<tr>
<td>normalizeThis</td>
<td>47, 48, 74, 119, 182</td>
</tr>
<tr>
<td>normalizeWithinArrays</td>
<td>155</td>
</tr>
<tr>
<td>numeric</td>
<td>43, 104, 106</td>
</tr>
<tr>
<td>numPairDeColNames</td>
<td>184</td>
</tr>
<tr>
<td>order</td>
<td>83, 148</td>
</tr>
<tr>
<td>orderMatrToRef</td>
<td>185</td>
</tr>
<tr>
<td>organizeAsListOfRepl</td>
<td>97–99, 120, 187</td>
</tr>
<tr>
<td>p.adjust</td>
<td>171, 196, 250</td>
</tr>
<tr>
<td>packageDownloadStat</td>
<td>188</td>
</tr>
<tr>
<td>pairsAsPropensMatr</td>
<td>189</td>
</tr>
<tr>
<td>par</td>
<td>103, 233</td>
</tr>
<tr>
<td>partialDist</td>
<td>190</td>
</tr>
<tr>
<td>partUnlist</td>
<td>76, 191</td>
</tr>
<tr>
<td>paste</td>
<td>8, 24, 192</td>
</tr>
<tr>
<td>pasteC</td>
<td>28, 64, 192</td>
</tr>
<tr>
<td>pie</td>
<td>207</td>
</tr>
<tr>
<td>presenceFilt</td>
<td>49, 193, 195, 242</td>
</tr>
<tr>
<td>presenceGrpFilt</td>
<td>194, 194</td>
</tr>
<tr>
<td>protectSpecChar</td>
<td>195</td>
</tr>
<tr>
<td>pVal2lfdr</td>
<td>196</td>
</tr>
<tr>
<td>randIndex</td>
<td>197</td>
</tr>
</tbody>
</table>
INDEX

261

randIndFx, 197
rankToContigTab, 198
ratioAllComb, 199
ratioToPpm, 200, 256
read.delim, 203
read.table, 121, 202, 204, 205
read_excel, 206
readCsvBatch, 201, 203, 256
readTabulatedBatch, 202
readVarColumns, 204
readXlsxBatch, 202, 205
regrBy1or2point, 208, 209
regrMultBy1or2point, 208, 209
renameColumns, 42, 55, 210
reorgByCluNo, 210
replicateStructure, 93, 157, 212, 237
replNAbyLow, 213
replPlateCV, 57–59, 75, 215, 220
rgb, 20, 103
rmDupl2colMatr, 216
rmEnumeratorName, 216, 250, 251
rnormW, 218
rowCVs, 57–59, 75, 215, 219, 220, 223
rowGrpCV, 57–59, 75, 220, 220
rowGrpMeans, 80, 221, 222–224
rowGrpNA, 222
rowGrpSds, 223, 224
rowGrpSums, 223
rowMedSds, 224
rowNormalize, 24, 60, 61, 183, 184, 225
rowSds, 146, 220–222, 224, 227, 228
rowSEMs, 223, 227
sampNoDeMArrayLM, 228
scale, 62, 63, 230, 238, 239
scaleXY, 62, 63, 229
sd, 91, 223, 227
searchDataPairs, 230
searchLinesAtGivenSlope, 18, 39, 41, 52, 231
simpleFragFig, 78, 110, 233
singleLineAnova, 234
sortBy2CategorAnd1IntCol, 235
sortByNRpeated, 236
sprintf, 116
stableModel, 237
standardW, 238
stdErrMedBoot, 239
strsplit, 112, 160, 175, 185
sub, 143
substr, 28, 56, 64, 123
summarizeCols, 66, 156, 240
sumNaperGroup, 241
Sweave, 244
Sys.time, 243
sysDate, 242
t, 231
t.test, 250
table, 93, 207, 237
tableToPlot, 243
TDist, 100
tempfile, 108
tst2FactLimma, 155, 234, 244
transpGrayScal, 246
treatTxtDuplicates, 107, 142, 247
triCoord, 248, 253
trimRedundText, 68–71, 145, 157, 213, 248
tTestAllVal, 249
unifyEnumerator, 250
uniqCountReport, 51, 252
unique, 30, 33, 50, 51, 71, 107, 126, 130, 135, 137, 142, 180, 253
unlist, 15, 76, 139, 191, 216
upperMaCoord, 248, 253
VolcanoPlotW, 229
which, 106
which.min, 35, 44, 63
withinRefRange, 83, 254
write.table, 256
writeCsv, 202, 254
XYToDiffPpm, 200, 256