Package 'Ecfun'

October 28, 2014

Version 0.1-2

Date 2013-10-13

Title Functions for Ecdat

Author Spencer Graves <spencer.graves@effectivedefense.org>

Maintainer Spencer Graves <spencer.graves@effectivedefense.org>

Depends R (>= 3.0.1)

Suggests BMA, car, DescTools, Ecdat, maps, grid, gridBase, pryr

Imports fda, gdata, RCurl, XML, tis, jpeg, MASS, TeachingDemos

Description Functions to update datasets in Ecdat and to create,manipulate, plot and analyze those and similar datasets.

LazyData true

License GPL (>= 2)

URL http://www.r-project.org

Repository CRAN

Repository/R-Forge/Project ecdat

Repository/R-Forge/Revision 251

Repository/R-Forge/DateTimeStamp 2014-10-27 19:36:21

Date/Publication 2014-10-28 08:01:16

NeedsCompilation no

R topics documented:

`Arrows` *Draw arrows between pairs of points.*

Description

Generalizes `graphics::arrows` to allow all arguments to be vectors. (As of R 3.1.0, only the first component of the `length` argument is used by `graphics::arrows`; others are ignored without a warning.)

Usage

```
Arrows(x0, y0, x1 = x0, y1 = y0, length = 0.25, angle = 30,
       code = 2, col = par("fg"), lty = par("lty"),
       lwd = par("lwd"), warnZeroLength=FALSE, ...)
```

Arguments

`x0, y0, x1, y1, length, angle, code, col, lty, lwd, ...`	as for `arrows`.
`warnZeroLength`	Issue a warning for zero length arrow? `arrow` does; skip if `FALSE`.

Details

1. Put all arguments in a `data.frame` to force them to shared length.
2. Call `arrows` once for each row.

Author(s)

Spencer Graves

See Also

`arrows`

Examples

```
##
## 1. Simple example:
##    3 arrows, the first with length 0 is suppressed
##
plot(1:3, type='n')
Arrows(1, 1, c(1, 2, 2), c(1, 2:3), col=1:3, length=c(1, .2, .6))

##
## 2.  with an NA
##
plot(1:3, type='n')
```

```
Arrows(1, 1, c(1, 2, 2), c(1, 2, NA), col=1:3, length=c(1, .2, .6))
```

`as.Date1970` *Date from a number of days since the start of 1970.*

Description

`as.Date.numeric` requires orgin to be specified. The present function assumes that this origin is January 1, 1970.

Usage

```
as.Date1970(x, ...)
```

Arguments

`x`	a numeric vector of dates in days since the start of 1970.
`...`	optional arguments to pass to as.Date.

Value

Returns a vector of Dates

Author(s)

Spencer Graves

See Also

`as.Date` `as.POSIXct1970`

Examples

```
days <- c(0, 1, 365)
Dates <- as.Date1970(days)


all.equal(c('1970-01-01', '1970-01-02', '1971-01-01'),
          as.character(Dates))



all.equal(days, as.numeric(Dates))
```

asNumericDF *Coerce to numeric dropping commas and info after a blank*

Description

Delete a leading dollar sign plus commas (thousand separators) and drop information after a blank, then coerce to numeric.

For a data.frame, apply to all columns, drop non-numeric columns, and order the rows by the orderBy. Some Excel imports include commas as thousand separators; this replaces any commas with char(0), ”.

Similarly, if "%" is found as the last character in any field, drop the percent sign and divide the resulting numeric conversion by 100 to convert to proportion.

Also, some character data includes footnote references following the year.

Table F-1 from the US Census Bureau needs all three of these features: It needs orderBy, because the most recent year appears first, just the opposite of most other data sets where the most recent year appears last. It has footnote references following a character string indicating the year. And it includes commas as thousand separators.

Usage

```
asNumericChar(x)
asNumericDF(x, keep=function(x)any(!is.na(x)), orderBy=NA)
```

Arguments

x	For asNumericChar, this is a character vector to be converted to numeric after gsub(',', '', x). For asNumericDF, this is a data.frame with all character columns to be converted to numerics.
keep	something to indicate which columns to keep
orderBy	Which columns to order the rows of x[, keep] by. Default is to keep the input order.

Details

1. Replace commas by nothing
2. strsplit on ’ ’ and take only the first part, thereby eliminating the footnote references.
3. Replace any blanks with NAs
4. as.numeric
5. lapply(x, 1-4)
6. order the rows

Value

all numeric data.frame

Author(s)

Spencer Graves

See Also

scan gsub Quotes

Examples

```
##
## 1.  simple example
##
fakeF1 <- data.frame(yr=c('1948', '1947 (1)'),
                     q1=c('1,234', ''), duh=rep(NA, 2),
                     dol=c('$1,234', ''),
                     pct=c('1%', '2%'))
nF1 <- asNumericDF(fakeF1)

nF1. <- data.frame(yr=asNumericChar(fakeF1$yr),
                   q1=asNumericChar(fakeF1$q1),
                   dol=asNumericChar(fakeF1$dol),
                   pct=c(.01, .02))

nF1c <- data.frame(yr=1948:1947, q1=c(1234, NA),
                   dol=c(1234, NA), pct=c(.01, .02))


all.equal(nF1, nF1.)


all.equal(nF1., nF1c)


##
## 2.  orderBy=1:2
##
nF. <- asNumericDF(fakeF1, orderBy=1:2)


all.equal(nF., nF1c[2:1,])
```

---

camelParse — *Split a character string where a capital letter follows a lowercase letter*

Description

Split a character string where a capital letter follows a lowercase letter.

Usage

```
camelParse(x, except=c('De', 'Mc', 'Mac'))
```

Arguments

x	a character vector
except	character vector giving exceptions: If any of these are found, ignore and look for the next one

Details

Find all places where a lowercase letter is followed by a capital.

Split on those points

Value

list of character vectors

Author(s)

Spencer Graves

See Also

strsplit

Examples

```
tst <- c('Smith, JohnJohn Smith',
         'EducationNational DefenseOther Committee',
         'McCain, JohnJohn McCain')
tst. <- camelParse(tst)


all.equal(tst., list(c('Smith, John', 'John Smith'),
    c('Education', 'National Defense', 'Other Committee'),
    c('McCain, John', 'John McCain') ) )
```

---

checkNames *Check and return names*

Description

Check and return names. If names are not provided or are not unique, write a message and return make.names consistent with warn and unique.

Usage

```
checkNames(x, warn=0, unique=TRUE,
   avoid=character(0),
   message0=head(deparse(substitute(x), 25), 2), ...)
```

Arguments

x	an R object suitable for names
warn	Numeric code for how to treat problems, consistent with the argument warn in options: Negative to ignore, 0 to save and print later, 1 to print as they occur, 2 or greater to convert to errors.
unique	logical: TRUE to check that names(x) are unique. Fix any duplicates with make.names.
avoid	a vector of regular expressions to avoid adding in the output of make.names with a companion replacement when found. Thus, length(avoid) must be a nonnegative even integer, with avoid[2*j-1] providing the pattern for regexpr and sub, and avoid[2*j] providing the replacement. See the second example.
message0	Base to prepend to any message
...	optional arguments for make.names

Details

1. namex <- names(x)
2. Check per warn and unique
3. Return an appropriate version of namex

Value

a character vector of the same length as x. If any problem is found, this character vector will have an attribute message describing the problem found. Message checking considers unique but ignores warn.

Author(s)

Spencer Graves

See Also

names make.names options for warn

Examples

```
##
## 1.  standard operation with no names
##
tst1 <- checkNames(1:2)
```

```
# check
tst1. <- make.names(character(2), unique=TRUE)
attr(tst1., 'message') <- paste(
  "1:2:  names = NULL; returning",
  "make.names(character(length(x))), TRUE)")

all.equal(tst1, tst1.)

##
## 2.  avoid=c('\\.0$', '\\.1$')
##
tst2 <-checkNames(1:2,
    avoid=c('\\.0$', '.2',
            '\\.1$', '.3') )

# check
tst2. <-c('X', 'X.3')
attr(tst2., 'message') <- paste(
  "1:2:  names = NULL; returning",
  "make.names(character(length(x))), TRUE)")

all.equal(tst2, tst2.)
```

`classIndex` *Convert class to an integer 1-8 and vice versa*

Description

`classIndex` converts the class of `x` to an integer:

1. NULL
2. logical
3. integer
4. numeric
5. complex
6. raw
7. character
8. other

`index2class` converts an integer back to the corresponding class.

Usage

```
classIndex(x)
index2class(i, otherCharacter=TRUE)
```

Arguments

x	an object whose class index is desired.
i	an integer to be converted to the name of the corresponding class
otherCharacter	logical: TRUE to convert 8 to "character"; FALSE to convert 8 to "other".

Details

The Writing R Extensions lists six different kinds of "atomic vectors": logical, integer, numeric, complex, character, and raw: See also Wickham (2013, section on "Atomic vectors" in the chapter on "Data structures"). These form a standard heirarchy, except for "raw", in that standard operations combining objects with different atomic classes will create an object of the higher class. For example, `TRUE + 2 + pi` returns a numeric object ((approximately 6.141593). Similarly, `paste(1, 'a')` returns the character string "1 a".

For "interpolation", we might expect users interpolating between objects of class "raw" (i.e., bytes) might most likely prefer "Numeric" to "Character" interpolation, coerced back to type "raw".

The index numbers for the classes run from 1 to 8 to make it easy to convert them back from integers to character strings.

Value

`classIndex` returns an integer between 1 and 7 depending on `class(x)`.

`index2class` returns a character string for the inverse transformation.

Author(s)

Spencer Graves

References

Wickham, Hadley (2014) *Advanced R*, especially Wickham (2013, section on "Atomic vectors" in the chapter on "Data structures").

See Also

`interpChar`

Examples

```
##
## 1.  classIndex
##
x1 <- classIndex(NULL)
x2 <- classIndex(logical(0))
x3 <- classIndex(integer(1))
x4 <- classIndex(numeric(2))
x5 <- classIndex(complex(3))
x6 <- classIndex(raw(4))
x7 <- classIndex(character(5))
x8 <- classIndex(list())
```

```
# check

all.equal(c(x1, x2, x3, x4, x5, x6, x7, x8), 1:8)


##
## 2.  index2class
##
c1 <- index2class(1)
c2 <- index2class(2)
c3 <- index2class(3)
c4 <- index2class(4)
c5 <- index2class(5)
c6 <- index2class(6)
c7 <- index2class(7)
c8 <- index2class(8)
c8o <- index2class(8, FALSE)

# check

all.equal(c(c1, c2, c3, c4, c5, c6, c7, c8, c8o),
          c('NULL', 'logical', 'integer', 'numeric',
            'complex', 'raw', 'character', 'character',
            'other'))
```

`compareLengths` *Compare the lengths of two objects*

Description

Issue a warning or error if the lengths of two objects are not compatible.

Usage

```
compareLengths(x, y,
   name.x=deparse(substitute(x), width.cutoff, nlines=1, ...),
   name.y=deparse(substitute(y), width.cutoff, nlines=1, ...),
   message0='', compFun=c('NROW', 'length'),
   action=c(compatible='', incompatible='warning'),
   length0=c('compatible', 'incompatible', 'stop'),
   width.cutoff=20, ...)
```

Arguments

`x, y`	objects whose lengths are to be compared

name.x, name.y	names of x and y to use in a message. Default = deparse(substitute(.), width.cutoff, nlines=1).
message0	character string to be included with name.x and name.y in a message.
compFun	function to use in the comparison.
action	A character vector of length 2 giving the names of functions to call if the lengths are not equal but are either 'compatible' or 'incompatible'; '' means no action.
length0	If length(x) or length(y) = 0 (but not both), treat this case as specified by length0.
width.cutoff	width.cutoff argument to pass to deparse. This gives the maximum number of characters to use in a name in error and warning messages.
...	optional arguments for deparse

Details

1. If nchar(name.x) = 0 = nchar(name.y), set name.x <- 'x', name.y <- 'y', and append 'in compareLengths:' to message0 for more informative messaging.

2. lenx <- do.call(compFun, list(x)); leny <- do.call(compFun, list(y))

3. if(lenx==leny)return(c('equal', ''))

4. Compatible?

5. Compose the message.

6. "action", as indicated

Value

A character vector of length 2. The first element is either 'equal', 'compatible' or 'incompatible'. The second element is the message composed.

Author(s)

Spencer Graves with help from Duncan Murdoch

See Also

interpChar

Examples

```
##
## 1.  equal
##

all.equal(compareLengths(1:3, 4:6), c("equal", ''))


##
## 2.  compatible
##
a <- 1:2
```

```
b <- letters[1:6]
comp.ab <- compareLengths(a, b, message0='Chk:')
comp.ba <- compareLengths(b, a, message0='Chk:')
# check
chk.ab <- c('compatible',
            'Chk: length(b) = 6 is 3 times length(a) = 2')

all.equal(comp.ab, chk.ab)


all.equal(comp.ba, chk.ab)


##
## 3.  incompatible
##
Z <- LETTERS[1:3]
comp.aZ <- compareLengths(a, Z)
# check
chk.aZ <- c('incompatible',
    ' length(Z) = 3 is not a multiple of length(a) = 2')

all.equal(comp.aZ, chk.aZ)

##
## 4.  problems with name.x and name.y
##
comp.ab2 <- compareLengths(a, b, '', '')
# check
chk.ab2 <- c('compatible',
             'in compareLengths: length(y) = 6 is 3 times length(x) = 2')

all.equal(comp.ab2, chk.ab2)

##
## 5.  zeroLength
##
zeroLen <- compareLengths(logical(0), 1)
# check
zeroL <- c('compatible', ' length(logical(0)) = 0')

all.equal(zeroLen, zeroL)
```

---

createMessage *Compose a message as a single substring from a character vector*

Description

This is a utility function to make it easier to automatically compose informative error and warning messages without using too many characters.

Usage

```
createMessage(x, width.cutoff=45, default='x', collapse='; ',
              endchars='...')
```

Arguments

x	input for paste
width.cutoff	maximum number of characters from x to return in a single string. This differs from the width.cutoff argument in deparse in that the output include here considers endchars, not part of deparse.
default	character string to return if nchar(x) = 0.
collapse	collapse argument for paste
endchars	a character string to indicate that part of the input string(s) was truncated.

Details

x. <- paste(..., collapse='; ') nchx <- nchar(x.) maxch <- (maxchar-nchar(endchar)) if(nchx>maxch) x2 <- substring(x., 1, maxch) x. <- paste0(x2, endchar)

Value

a character string with at most width.cutoff characters.

Author(s)

Spencer Graves

See Also

paste substr nchar

Examples

```
##
## 1.  typical use
##
tstVec <- c('Now', 'is', 'the', 'time')
msg <- createMessage(tstVec, 9, collapse=':',
                     endchars='//')

all.equal(msg, 'Now:is://')

##
## 2.  in a function
##
tstFn <- function(cl)createMessage(deparse(cl), 9)
Cl <- quote(plot(1:3, y=4:6, col='red', main='Title'))
msg0 <- tstFn(Cl)
# check
```

```
msg. <- 'plot(1...'

all.equal(msg0, msg.)


##
## 3.  default
##
y <- createMessage(character(3), default='y')

all.equal(y, 'y')
```

---

createX2matchY *Create X to match Y*

Description

Return a default object of class `index2class(max(classIndex(x), classIndex(y)))` and length = length(y).

For example, suppose `class(x)` == 'numeric', for which `classIndex` = 4. If `class(y)` = 'integer', then an object of class 'numeric' is returned. However, if `class(y)` = 'character', then an object of class 'character' is returned.

Usage

```
createX2matchY(x, y)
```

Arguments

`x, y`	objects of possibly different classes and lengths.

Value

A vector of the same length as `y` whose class is `index2class(max(classIndex(x), classIndex(y)))`.

Author(s)

Spencer Graves

See Also

`interpPairs`

Examples

```
##
## 1.  NULL
## -
null <- createX2matchY(NULL, NULL)
# check

all.equal(null, NULL)

##
## 2.  logical
##
lgcl3 <- createX2matchY(NULL,
           c(FALSE, TRUE, FALSE))

# check

all.equal(lgcl3, logical(3))


##
## 3.  integer
##
int3 <- createX2matchY(integer(0),
           c(FALSE, TRUE, FALSE))
# check

all.equal(int3, integer(3))


##
## 4.  list -> character
##
ch3 <- createX2matchY(integer(0),
         list(a=1, b=2, c=3))
# check

all.equal(ch3, character(3))
```

`financialCrisisFiles` *Files containing financial crisis data*

Description

`FinancialCrisisFiles` in `Ecdat` is an object of class `financialCrisisFiles` created by the `financialCrisisFiles` function to describe files containing data on financial crises downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

Usage

```
financialCrisisFiles(files=c("22_data.xls", "23_data.xls",
       "Varieties_Part_III.xls", "25_data.xls"), ...)
```

Arguments

files	character vector of file names
...	arguments to pass with file and sheet name to read.xls when reading a sheet of an MS Excel file. This is assumed to be the same for all sheets of all files. If this is not the case, the resulting financialCrisisFiles object will have to be edited manually before using it to read the data.

Details

Reinhart and Rogoff (http://www.reinhartandrogoff.com) provide numerous data sets analyzed in their book, "This Time Is Different: Eight Centuries of Financial Folly". Of interest here are data on financial crises of various types for 70 countries spanning the years 1800 - 2010, downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

The function financialCrisisFiles produces a list of class financialCrisisFiles describing four different Excel files in very similar formats with one sheet per Country and a few extra descriptor sheets. The data object FinancialCrisisFiles is the default output of that function.

It does this in several steps:

1. Read the first sheet of each file
2. Extract the names of the Countries from that first sheet.
3. Elimiate any blank spaces in the names to convert, e.g., "Costa Rica" to "CostaRica".
4. Find the sheets corresponding to each of the compressed names.
5. Construct the output list.

Value

The function financialCrisisFiles returns a list of class financialCrisisFiles. This is a list with components carrying the names of files to be read. Each component is a list of optional arguments to pass to do.call(read.xls, ...) to read the sheet with name = name of that component.

The default value returned by financialCrisisFiles is the data object FinancialCrisisFiles. This corresponds to the files downloaded from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/ in January 2013 (except for the fourth, which was not available there because of an error with the web site but instead was obtained directly from Prof. Reinhart).

Author(s)

Spencer Graves

Source

http://www.reinhartandrogoff.com

References

Carmen M. Reinhart and Kenneth S. Rogoff (2009) This Time Is Different: Eight Centuries of Financial Folly, Princeton U. Pr.

See Also

read.xls

Examples

```
Ecdat.demoFiles <- system.file('demoFiles', package='Ecdat')
Ecdat.xls <- dir(Ecdat.demoFiles, pattern='xls$',
                 full.names=TRUE)
if(require(gdata)){
  tst <- financialCrisisFiles(Ecdat.xls)
}
## Not run:
# check
\dontshow{stopifnot(}
all.equal(tst, data(FinancialCrisisFiles))
\dontrun{)}

## End(Not run)
```

`getElement2` *Extract a named element from an object with a default*

Description

Get element `name` of `object`. If `object` does not have an element `name`, return `default`.

If the `name` element of `object` is `NULL` the result depends on `warn.NULL`: If `TRUE`, issue a warning and return `default`. Otherwise, return `NULL`

Usage

```
getElement2(object, name=1, default=NA, warn.NULL=TRUE,
            envir=list(), returnName)
```

Arguments

`object`	object from which to extract component `name`.
`name`	Name or index of the element to extract
`default`	default value if `name` is not part of `object`.
`warn.NULL`	logical to decide how to treat cases where `object` has a component `name`: If `TRUE`, return `default` with a warning. Otherwise, return `NULL`.
`envir`	Supplemental list beyond `object` in which to look for names in case `object[[name]]` is a language object that must be evaluated.

returnName — logical: TRUE to return as.character of any name found as an element of object.
FALSE to eval any name found in the environment of object.
Default = TRUE if name == 1 or a character string matching the name of the first element of object.

Details

1. If is.numeric(name) In <- (1 <= name <= length(object))
2. else In <- if(name %in% names(object))
3. El <- if(In) object[[name]] else default
4. warn.NULL?
5. if(returnName) return(as.character(El)) else return(eval(El, envir=object))

Value

an object of the form of object[[name]]; if object does not have an element or slot name, return default.

Author(s)

Spencer Graves with help from Marc Schwartz and Hadley Wickham

See Also

getElement, which also can return slots from S4 objects.

Examples

```
##
## 1.  name in object, return
##
e1 <- getElement2(list(ab=1), 'ab', 2) # 1
# check

all.equal(e1, 1)


##
## 2.  name not in object, return default
##
eNA <- getElement2(list(), 'ab') # default default = NA
# check

all.equal(eNA, NA)


e0 <- getElement2(list(), 'ab', 2) # name not in object

all.equal(e0, 2)
```

```
e2 <- getElement2(list(ab=1), 'a', 2) # partial matching not used

all.equal(e2, 2)


##
## 3.  name NULL in object, return default
##
ed <- getElement2(list(a=NULL), 'a',2) # 2 with a warning

all.equal(ed, 2)


e. <- getElement2(list(a=NULL), 'a', 2, warn.NULL=FALSE) # NULL

all.equal(e., NULL)


eNULL <- getElement2(list(a=NULL), 'a', NULL) # NULL

all.equal(eNULL, NULL)


##
## 4.  Language:  find, eval, return
##
Qte <- quote(plot(1:4, y=x, col=c2))
if(require(pryr)){
  Qt <- pryr::standardise_call(Qte) # add the name 'x'
  fn <- getElement2(Qt)
  eQuote <- getElement2(Qt, 'y')
  Col2 <- getElement2(Qt, 'col', envir=list(c2=2))
# check

  all.equal(fn, 'plot')


  all.equal(eQuote, 1:4)


  all.equal(Col2, 2)

}
```

---

grepNonStandardCharacters

grep for nonstandard characters

Description

Return the indices of elements of x containing characters that are not in standardCharacters.

Usage

```
grepNonStandardCharacters(x, value=FALSE,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   ... )
```

Arguments

x	character vector in which it is desired to identfy elements containing characters not in standardCharacters.
value	logical: TRUE to return the values found in x, FALSE to return their indices.
standardCharacters	Characters to overlook in x to identify anything not in standardCharacters.
...	optional arguments for regexpr

Details

1. x. <- strsplit(x, ''): convert the input character vector to a list of vectors of character vectors with nchar(x.[i]) == 1 for i in 1:length(x).

2. sapply(x., ...) to identify all elements for which any element of x[[i]] is not in standardCharacters.

Value

an integer vector identifying all elements of x containing a character not in standardCharacters.

Author(s)

Spencer Graves

See Also

grep, regexpr, subNonStandardCharacters, showNonASCII

Examples

```
Names <- c('Raul', 'Ra`l', 'Torres,Raul', 'Torres, Raul')
#  confusion in character sets can create
#  names like Names[2]

chk <- grepNonStandardCharacters(Names)


all.equal(chk, 2)
```

```
chkv <- grepNonStandardCharacters(Names, TRUE)


all.equal(chkv, 'Ra`l')
```

`Interp` *Interpolate between numbers or numbers of characters*

Description

Numeric interpolation is defined in the usual way:

`xOut <- x*(1-proportion) + y*proportion`

Character interpolation does linear interpolation on the number of characters of `x` and `y`. If `length(proportion) == 1`, interpolation is done on `cumsum(nchar(.))`. If `length(proportion) > 1`, interpolation is based on `nchar`. In either case, the interpolant is rounded to an integer number of characters. `Interp` then returns `substring(y, ...)` unless `nchar(x) > nchar(y)`, when it returns `substring(x, ...)`.

Character interpolation is used in two cases: (1) At least one of `x` and `y` is character. (2) At least one of `x` and `y` is neither logical, integer, numeric, complex nor raw, and `class(unclass(.))` is either integer or character.

In all other cases, numeric interpolation is used.

NOTE: This seems to provide a relatively simple default for what most people would want from the six classes of atomic vectors (logical, integer, numeric, complex, raw, and character) and most other classes. For example, `class(unclass(factor))` is integer. The second rule would apply to this converting it to character. The `coredata` of an object of class `zoo` could be most anything, but this relatively simple rule would deliver what most people want in most case. An exception would be an object with integer coredata. To handle this as numeric, a `Interp.zoo` function would have to be written.

Usage

```
Interp(x, ...)
## Default S3 method:
Interp(x, y, proportion,
        argnames=character(3), message0=character(0), ...)
InterpChkArgs(x, y, proportion,
        argnames=character(3), message0=character(0), ...)
InterpChar(argsChk, ...)
InterpNum(argsChk, ...)
```

Arguments

`x, y`	two vectors of the same class or to be coerced to the same class.
`proportion`	A number or numeric vector assumed to be between 0 and 1.

argnames	a character vector of length 3 giving args name.x, name.y, and proportion to pass to compareLengths to improve the value of any diagnostic message in case lengths are not compatible.
message0	A character string to be passed with argnames to compareLengths to improve the value of any diagnostic message in case lengths are not compatible.
argsChk	a list as returned by interpChkArgs
...	optional arguments for compareLengths

Details

Interp is an S3 generic function to allow users to easily modify the behavior to interpolate between special classes of objects.

Interp has two basic algorithms for "Numeric" and "Character" interpolation.

The computations begin by calling InterpChkArgs to dispose quickly of simple cases (e.g, x or y missing or length 0 or if proportion is <= 0 or >= 1 or missing). It returns a list.

If the list contains a component named "xout", Interp returns that value with no further computations.

Otherwise, the list returned by InterpChkArgs includes components "algorithm", "x", "y", "proportion", "pLength1" (defined below), "raw", and "outclass". The "algorithm" component must be either "Numeric" or "Character". That algoritm is then performed as discussed below using arguments "x", "y", and "proportion"; all three will have the same length. The class of "x" and "y" will match the algorithm. The list component "raw" is logical: TRUE if the output will be raw or such that class(unclass(.)) of the output will be raw. In that case, a "Numeric" interpolation will be transformed back into "raw". "outclass" will either be a list of attributes to apply to the output or NA. If a list, "xout" will be added as component ".Data" to the list "outclass" and then then processed as do.call('structure', outclass) to produce the desired output.

These two basic algorithms ("Numeric" and "Character") are the same if proportion is missing or not numeric: In that case Interp throws an error.

We now consider "Character" first, because it's domain of applicability is easier to describe. The "Numeric" algorithm is used in all other cases

1. "CHARACTER"

* 1.1. The "CHARACTER" algorithem is used when at least one of x and y is neither logical, integer, numeric, complex nor raw and satisfies one of the following two additional conditions:

** 1.1.1. Either x or y is character.

** 1.1.2. class(unclass(.)) for at least one of x and y is either character or integer.

NOTE: The strengths and weaknesses of 1.1.2 can be seen in considering factors and integer vectors of class zoo: For both, class(unclass(.)) is integer. For factors, we want to use as.character(.). For zoo objects with coredata of class integer, we would want to use numeric interpolation. This is not allowed with the current code but could be easily implemented by writing Interp.zoo.

* 1.2. If either x or y is missing or has length 0, the one that is provided is returned unchanged.

* 1.3. Next determine the class of the output. This depends on whether neither, one or both of x and y have one of the six classes of atomic vectors (logical, integer, numeric, complex, raw, character):

** 1.3.1. If both x and y have one of the six atomic classes and one is character, return a character object.

** 1.3.2. If only one of x and y have an atomic class, return an object of the class of the other.

** 1.3.3. If neither of x nor y have a basic class, return an object with the class of y.

* 1.4. Set `pLength1 <- (length(proportion) == 1)`:

** 1.4.1. If(pLength1) do the linear interpolation on `cumsum(nchar(.))`.

** 1.4.2. Else do the linear interpolation on `nchar`.

* 1.5. Next check x, y and `proportion` for comparable lengths: If all have length 0, return an object of the appropriate class. Otherwise, call `compareLengths(x, proportion)`, `compareLengths(y, proportion)`, and `compareLengths(x, y)`.

* 1.6. Extend x, y, and `proportion` to the length of the longest using `rep`.

* 1.7. nchOut <- the number of characters to output using numeric interpolation and rounding the result to integer.

* 1.8. Return `substring(y, 1, nchOut)` except when the number of characters from x exceed those from y, in which case return `substring(x, 1, nchOut)`. [NOTE: This meets the naive end conditions that the number of characters matches that of x when `proportion` is 0 and matches that of y when `proportion` is 1. This can be used to "erase" characters moving from one frame to the next in a video. See the examples.

2. "NUMERIC"

* 2.1. Confirm that this does NOT satisfy the condition for the "Character" algorithm.

* 2.2. If either x or y is missing or has `length` 0, return the one provided.

* 2.3. Next determine the class of the output. As for "Character" described in section 1.3, this depends on whether neither, one or both of x and y have a basic class other than character (logical, integer, numeric, complex, raw):

** 2.3.1. If `proportion` <= 0, return x unchanged. If `proportion` >= 1, return y unchanged.

** 2.3.2. If neither x nor y has a basic class, return an object of class equal that of y.

** 2.3.3. If exactly one of x and y does not have a basic class, return an object of class determined by `class(unclass(.))` of the non-basic argument.

** 2.3.4. When interpolating between two objects of class raw, convert the interpoland back to class raw. Do this even when 2.3.2 or 2.3.3 applies and `class(unclass(.))` of both x and y are of class raw.

* 2.4. Next check x, y and `proportion` for comparable lengths: If all have length 0, return an object of the appropriate class. Otherwise, call `compareLengths(x, proportion)`, `compareLengths(y, proportion)`, and `compareLengths(x, y)`.

* 2.5. Compute the desired interpolation and convert it to the required class per step 2.3 above.

Value

`Interp` returns a vector whose class is described in "* 1.3" and "* 2.3" in "Details" above.

`InterpChkArgs` returns a list or throws an error as described in "Details" above.

Author(s)

Spencer Graves

References

The *Writing R Extensions* manual (available via `help.start()`) lists six different classes of atomic vectors: `logical`, `integer`, `numeric`, `complex`, `raw` and `character`. See also Wickham, Hadley (2014) *Advanced R*, especially Wickham (2013, section on "Atomic vectors" in the chapter on "Data structures").

See Also

`classIndex` `interpPairs`

Many other packages have functions with names like "interp", "interp1", and "interpolate". Some do one-dimensional interpolation. Others do two-dimensional interpolation. Some offer different kinds of interpolation beyond linear. At least one is a wrapper for `approx`.

Examples

```
##
## 1.  numerics
##
# 1.1.  standard
xNum <- interpChar(1:3, 4:5, (0:3)/4)
# answer
xN. <- c(1, 2.75, 3.5, 4)

all.equal(xNum, xN.)


# 1.2.  with x but not y:
# return that vector with a warning

xN1 <- Interp(1:4, p=.5)
# answer
xN1. <- 1:4

all.equal(xN1, xN1.)


##
## 2.  Single character vector
##


i.5 <- Interp(c('a', 'bc', 'def'), character(0), p=0.3)
# with y = NULL or character(0),
# Interp returns x

all.equal(i.5, c('a', 'bc', 'def'))
```

```
i.5b <- Interp('', c('a', 'bc', 'def'), p=0.3)
# Cumulative characters (length(proportion)=1):
#     0.3*(total 6 characters) = 1.2 characters
i.5. <- c('a', 'b', '')

all.equal(i.5b, i.5.)


##
## 3.  Reverse character example
##
i.5c <- Interp(c('a', 'bc', 'def'), '', 0.3)
# check:  0.7*(total 6 characers) = 4.2 characters
i.5c. <- c('a', 'bc', 'd')

all.equal(i.5c, i.5c.)


##
## 4.  More complicated example
##
xCh <- Interp('', c('Do it', 'with R.'),
              c(0, .5, .9))
# answer
xCh. <- c('', 'with', 'Do i')

all.equal(xCh, xCh.)

##
## 5.  Still more complicated
##
xC2 <- Interp(c('a', 'fabulous', 'bug'),
                  c('bigger or', 'just', 'big'),
                  c(.3, .3, 1) )
x.y.longer <- c('bigger or', 'fabulous', 'big')
# use y with ties
# nch smaller        1          4         3
# nch larger         9          8         3
# d.char             8,         4,        0
# prop              .3,        .7,        1
# prop*d.char      2.4,       2.8,        0
# smaller+p*d        3,         7,        3
xC2. <- c('big', 'fabulou', 'big')

all.equal(xC2, xC2.)


##
## 6.  with one NULL
##
null1 <- Interp(NULL, 1, .3)
```

```
all.equal(null1, 1)


null2 <- Interp('abc', NULL, .3)

all.equal(null2, 'abc')

##
## 7.  length=0
##
log0 <- interpChar(logical(0), 2, .6)

all.equal(log0, 1.2)


##
## 8.  Date
##
Jan1.1980 <- as.Date('1980-01-01')

Jan1.1972i <- Interp(0, Jan1.1980, .2)
# check
Jan1.1972 <- as.Date('1972-01-01')


all.equal(Jan1.1972, round(Jan1.1972i))


##
## 9.  POSIXct
##
Jan1.1980c <- as.POSIXct(Jan1.1980)

Jan1.1972ci <- Interp(0, Jan1.1980c, .2)
# check
Jan1.1972ct <- as.POSIXct(Jan1.1972)


abs(difftime(Jan1.1972ct, Jan1.1972ci,
             units="days"))<0.5
```

`interpChar` *Interpolate between numbers or numbers of characters*

Description

For x and y logical, integer, numeric, Date or POSIX:

```
xOut <- x*(1-.proportion) + y*.proportion
```

Otherwise, coerce to character and return a substring of x or y with number of characters interpolating linearly between nchar(x) and nchar(y); see details.

*** NOTE: This function is currently in flux. The results may not match the documentation and may change in the future.

The current version does character interpolation on the cumulative number of characters with defaults with only one argument that may not be easy to understand and use. Proposed:

old: interpolate on number of characters in each string with the default for a missing argument being character(length(x)) [or character(length(y)) or numeric(length(x)) or ...]

2014-08-08: default with either x or y missing should be to set the other to the one we have, so interpChar becomes a no op – except that values with .proportion outside ("validProportion" = [0, 1] by default) should be dropped.

Usage

```
interpChar(x, ...)
## S3 method for class 'list'
interpChar(x, .proportion,
        argnames=character(3), message0=character(0), ...)
## Default S3 method:
interpChar(x, y, .proportion,
        argnames=character(3), message0=character(0), ...)
```

Arguments

x	either a vector or a list. If a list, pass the first two elements as the first two arguments of interpChar.default.
y	a vector
.proportion	A number or numeric vector assumed to be between 0 and 1.
argnames	a character vector of length 3 giving args name.x, name.y, and .proportion to pass to compareLengths to improve the value of any diagnostic message in case lengths are not compatible.
message0	A character string to be passed with argnames to compareLengths to improve the value of any diagnostic message in case lengths are not compatible.
...	optional arguments for compareLengths

Details

1. x, y and .proportion are first compared for compatible lengths using compareLengths. A warning is issued if the lengths are not compatible. They are then all extended to the same length using rep.

2. If x and y are both numeric, interpChar returns the standard linear interpolation (described above).

3. If x, y, and .proportion are all provided with at least one of x and y not being numeric or logical, the algorithm does linear interpolation on the difference in the number of characters between x and y. It returns characters from y except when nchar(x) > nchar(y), in which case it returns characters from x. This meets the end conditions that the number of characters matches that of x

when .proportion is 0 and matches that of y when .proportion is 1. This can be used to "erase" characters moving from one frame to the next in a video. See the examples.

4. If either x or y is missing, it is replaced by a default vector of the same type and length; for example, if y is missing and x is numeric, y = numeric(length(x)). (If the one supplied is not numeric or logical, it is coerced to character.)

Value

A vector: Numeric if x and y are both numeric and character otherwise. The length = max length of x, y, and .proportion.

Author(s)

Spencer Graves

See Also

interpPairs, which calls interpChar

classIndex, which is called by interpChar to help decide the class of the interpoland.

Examples

```
##
## 1.  numerics
##
# 1.1.  standard
xNum <- interpChar(1:3, 4:5, (0:3)/4)
# answer
xN. <- c(1, 2.75, 3.5, 4)

all.equal(xNum, xN.)


# 1.2.  list of length 1 with a numeric vector:
#       return that vector with a warning
xN1 <- interpChar(list(a.0=1:4), .5)
# answer
xN1. <- 1:4

all.equal(xN1, xN1.)


##
## 2.  Single character vector
##
i.5 <- interpChar(list(c('a', 'bc', 'def')), .p=0.3)
# If cumulative characters:
#        0.3*(total 6 characters) = 1.8 characters
#
# However, the current code does something different,
# returning "a", "bc", "d" <- like using 1-.p?
```

```
# This is a problem with the defaults with a single
# argument;  ignore this issue for now.
# 2014-06-04
i.5. <- c('a', 'b', '')

#all.equal(i.5, i.5.)


##
## 3.  Reverse character example
##
i.5c <- interpChar(c('a', 'bc', 'def'), '', 0.3)
# check:  0.7*(total 6 characers) = 4.2 characters
i.5c. <- c('a', 'bc', 'd')

all.equal(i.5c, i.5c.)


#  The same thing specified in a list
i.5d <- interpChar(list(c('a', 'bc', 'def'), ''), 0.3)

all.equal(i.5d, i.5c.)


##
## 4.  More complicated example
##
xCh <- interpChar(list(c('Do it', 'with R.')),
                  c(0, .5, .9))
# answer
xCh. <- c('', 'with', 'Do ')
# With only one input, it's assumed to be y.
# It is replicated to length(.proportion),
# With nchar = 5, 7, 5, cum = 5, 12, 17.

all.equal(xCh, xCh.)

##
## 5.  Still more complicated
##
xC2 <- interpChar(c('a', 'fabulous', 'bug'),
                  c('bigger or', 'just', 'big'),
                  c(.3, .3, 1) )
# answer
x.y.longer <- c('bigger or', 'fabulous', 'big')
# use y with ties
# nch smaller        1          4         3
# nch larger         9          8         3
# d.char             8,         4,        0
# cum characters     8,        12,       12
# prop              .3,        .7,        1
# prop*12          3.6,       8.4,       12
# cum.sm             1,         5,        8
```

```
# cum.sm+prop*12     5,         13,       20
#   -cum(larger[-1]) 5,          4,        3
xC2. <- c('bigge', 'fabu', 'big')

all.equal(xC2, xC2.)


##
## 6.  with one NULL
##
null1 <- interpChar(NULL, 1, 1)

all.equal(null1, 1)


null2 <- interpChar('abc', NULL, .3)

all.equal(null2, 'ab')

##
## 7.  length=0
##
log0 <- interpChar(logical(0), 2, .6)

all.equal(log0, 1.2)


##
## 8.  Date
##



##
## 9.  POSIXct
##
```

`interpPairs` *interpolate between pairs of vectors in a list*

Description

This does two things:

1. Computes a `.proportion` interpolation between `pairs` by passing each pair with `.proportion` to `interpChar`. `interpChar` does standard linear interpolation with numerics and interpolates based on the number of characters with non-numerics.

2. Discards rows of interpolants for which `.proportion` is outside `validProportion`. If `object` is a `list`, corresponding rows of other vectors of the same length are also discarded.

   NOTE: There are currently discrepancies between the documentation and the code over defaults when one but not both elements of a pair are provided. The code returns an answer. If that's not acceptable, provide the other half of the pair. After some experience is gathered, the question of defaults will be revisited and the code or the documentation will change.

Usage

```
interpPairs(object, ...)
## S3 method for class 'call'
interpPairs(object,
    nFrames=1, iFrame=nFrames,
    endFrames=round(0.2*nFrames),
    envir = parent.frame(),
    pairs=c('1'='\\.0$', '2'='\\.1$', replace0='',
                replace1='.2', replace2='.3'),
    validProportion=0:1, message0=character(0), ...)
## S3 method for class 'function'
interpPairs(object,
    nFrames=1, iFrame=nFrames,
    endFrames=round(0.2*nFrames),
    envir = parent.frame(),
    pairs=c('1'='\\.0$', '2'='\\.1$', replace0='',
                replace1='.2', replace2='.3'),
    validProportion=0:1, message0=character(0), ...)
## S3 method for class 'list'
interpPairs(object,
    .proportion, envir=list(),
        pairs=c('1'='\\.0$', '2'='\\.1$', replace0='',
                replace1='.2', replace2='.3'),
        validProportion=0:1, message0=character(0), ...)
```

Arguments

`object`	A `call`, `function`, `list` or `data.frame` with names possibly matching `pairs[1:2]`. When names matching both of `pairs[1:2]`, they are converted to potentially common names using `sub(pairs[i], pairs[3], ...)`. When matches are found among the potentially common names, they are passed with `.proportion` to `interpChar` to compute an interpolation. The matches are removed and replaced with the interpolant, shortened by excluding any rows for which `.proportion` is outside `validProportion`. Elements with "common names" that do not have a match are replaced by elements with the common names that have been shortened by omitting rows with `.proportion` outside `validProportion`. Thus, if `x.0` is found without `x.1`, `x.0` is removed and replaced by `x`.
`nFrames`	number of distinct plots to create.
`iFrame`	integer giving the index of the single frame to create. Default = nFrames.

An error is thrown if both `iFrame` and `.proportion` are not `NULL`.

`endFrames` Number of frames to hold constant at the end.

`.proportion` a numeric vector assumed to lie between 0 and 1 specifying how far to go from suffixes[1] to suffixes[2]. For example, if `x.0` and `x.1` are found and are numeric, `x = x.0 + .proportion * (x.1 - x.0)`. Rows of `x` and any other element of `object` of the same length are dropped for any `.proportion` outside `validProportion`.

An error is thrown if both `iFrame` and `.proportion` are not `NULL`.

`envir` environment / list to use with codeobject, which can optionally provide other variables to compute what gets plotted; see the example below using this argument.

`pairs` a character vector of two regular expressions to identify elements of `object` between which to interpolate and three replacements.

(1) The first of the three replacements is used in `sub` to convert each `pairs[1:2]` name found to the desired name of the interpolate. Common names found are then passed with `.proportion` to `interpChar`, which does the actual interpolation.

(2, 3) `interpPairs` also calls `checkNames(object, avoid = pairs[c(1, 3, 2, 5)])`. This confirms that `object` has `names`, and all such names are unique. If `object` does not have names or has some duplicate names, the `make.names` is called to fix that problem, and any new names that match `pairs[1:2]` are modified using `sub` to avoid creating a new match. If the modification still matches `pairs[1:2]`, it generates an error.

`validProportion`

Range of values of `.proportion` to retain, as noted with the discussion of the `object` argument.

`message0` a character string passed to `interpChar` to improve the value of diagnostic messages

`...` optional arguments for `sub`

Details

*** FUNCTION ***

First `interpPairs.function` looks for arguments `firstFrame`, `lastFrame`, and `Keep`. If any of these are found, they are stored locally and removed from the function. If `iFrame` is provided, it is used with with these arguments plus `nFrames` and `endFrames` to compute `.proportion`.

If `.proportion` is outside `validProportion`, `interpPairs` does nothing, returning `enquote(NULL)`.

If `any(.proportion)` is inside `validProportion`, `interpPairs.function` next uses `grep` to look for arguments with names matching `pairs[1:2]`. If any are found, they are passed with `.proportion` to `interpChar`. The result is stored in the modified `object` with the common name obtained from `sub(pairs[i], pairs[3], ...)`, `i` = 1, 2.

The result is then evaluated and then returned.

*** LIST ***

1. ALL.OUT: if(none(0<=.proportion<=1))return 'no.op' = list(fun='return', value=NULL)

2. FIND PAIRS: Find names matching `pairs[1:2]` using `grep`. For example, names like `x.0` match the default `pairs[1]`, and names like `x.1` match the default `pairs[1]`.

3. MATCH PAIRS: Use `sub(pairs[i], pairs[3], ...)` for i = 1:2, to translate each name matching `pairs[1:2]` into something else for matching. For example, he default `pairs` thus translates, e.g., `x.0` and `x.1` both into `x`. In the output, `x.0` and `x.1` are dropped, replaced by `x = interpChar(x.0, x.1, .proportion, ...)`. Rows with `.proportion` outside `validProportion` are dropped in `x`. Drop similar rows of any numeric or character vector or `data.frame` with the same number of rows as `x` or `.proportion`.

4. Add component `.proportion` to `envir` to make it available to `eval` any `language` component of `object` in the next step.

5. Loop over all elements of `object` to create `outList`, evaluating any expressions and computing the desired interpolation using `interpChar`. Computing `xleft` in this way allows `xright` to be specified later as `quote(xleft + xinch(0.6))`, for example. This can be used with a call to `rasterImageAdj`.

6. Let `N` = the maximum number of rows of elements of `outList` created by interpolation in the previous step. If `.proportion` is longer, set `N = length(.proportion)`. Find all vectors and `data.frame`s in `outList` with `N` rows and delete any rows for which `.proportion` is outside `validProportion`.

7. Delete the raw pairs found in steps 1-3, retaining the element with the target name computed in steps 4 and 5 above. For other elements of `object` modified in the previous step, retain the shortened form. Otherwise, retain the original, unevaluated element.

Value

a `list` with elements containing the interpolation results.

Author(s)

Spencer Graves

See Also

`interpChar` for details on interpolation. `compareLengths` for how lengths are checked and messages composed and written.

`enquote`

Examples

```
###
###
### 1.  interpPairs.function
###
###

##
## 1.1.  simple
##
plot0 <- quote(plot(0))
```

```
plot0. <- interpPairs(plot0)
# check

all.equal(plot0, plot0.)


##
## 1.2.  no op
##
noop <- interpPairs(plot0, iFrame=-1)
# check

all.equal(noop, enquote(NULL))


##
## 1.3.  a more typical example
## example function for interpPairs
tstPlot <- function(){
  plot(1:2, 1:2, type='n')
  lines(firstFrame=1:3,
        lastFrame=4,
        x.1=seq(1, 2, .5),
        y.1=x,
        z.0=0, z.1=1,
        txt.1=c('CRAN is', 'good', '...'),
        col='red')
}
tstbo <- body(tstPlot)
iPlot <- interpPairs(tstbo[[2]])
# check
iP <- quote(plot(1:2, 1:2, type='n'))

all.equal(iPlot, iP)


iLines <- interpPairs(tstbo[[3]], nFrames=5, iFrame=2)
# check:
# .proportion = (iFrame-firstFrame)/(lastFrame-firstFrame)
#  = c(1/3, 0, -1/3)




# if x.0 = 0 and y.0 = 0 by default:
iL <- quote(linex(x=c(1/3, 0), y=c(1/9, 0), z=c(1/3, 0),
          tst=c('CR', '')))
##
##**** This example seems to give the wrong answer
##**** 2014-06-03:  Ignore for the moment
##

#all.equal(iLines, iL)
```

```
##
## 1.4.  Don't throw a cryptic error with NULL
##
ip0 <- interpPairs(quote(text(labels.1=NULL)))


###
###
### 2.  interpPairs.list
###
###

##
## 2.1.  (x.0, y.0, x.1, y.1) -> (x,y)
##
tstList <- list(x.0=1:5, y.0=5:9, y.1=9:5, x.1=9,
                ignore=letters, col=1:5)
xy <- interpPairs(tstList, 0.1)
# check
xy. <- list(ignore=letters, col=1:5,
            x=1:5 + 0.1*(9-1:5),
            y=5:9 + 0.1*(9:5-5:9) )
# New columns, 'x' and 'y', come after
# columns 'col' and 'ignore' already in tstList

all.equal(xy, xy.)


##
## 2.2.  Select the middle 2:
##      x=(1-(0,1))*3:4+0:1*0=(3,0)
##
xy0 <- interpPairs(tstList[-4], c(-Inf, -1, 0, 1, 2) )
# check
xy0. <- list(ignore=letters, col=3:4, x=c(3,0), y=7:6)


all.equal(xy0, xy0.)

##
## 2.3.  Null interpolation because of absence of y.1 and x.0
##
xy02 <- interpPairs(tstList[c(2, 4)], 0.1)
# check
#### NOT the current default answer;  revisit later.
xy02. <- list(y=5:9, x=9)

# NOTE:  length(x) = 1 = length(x.1) in testList

#all.equal(xy02, xy02.)
```

```
##
## 2.4.  Select an empty list (make sure this works)
##
x0 <- interpPairs(list(), 0:1)
# check
x0. <- list()
names(x0.) <- character(0)

all.equal(x0, x0.)


##
## 2.5.  subset one vector only
##
xyz <- interpPairs(list(x=1:4), c(-1, 0, 1, 2))
# check
xyz. <- list(x=2:3)

all.equal(xyz, xyz.)


##
## 2.6.  with elements of class call
##
xc <- interpPairs(list(x=1:3, y=quote(x+sin(pi*x/6))), 0:1)
# check
xc. <- list(x=1:3, y=quote(x+sin(pi*x/6)))

all.equal(xc, xc.)


##
## 2.7. text
##
#  2 arguments
j.5 <- interpPairs(list(x.0='', x.1=c('a', 'bc', 'def')), 0.5)
# check
j.5. <- list(x=c('a', 'bc', ''))

all.equal(j.5, j.5.)


##
##  2.8.  text, 1 argument as a list
##
j.50 <- interpPairs(list(x.1=c('a', 'bc', 'def')), 0.5)
# check

all.equal(j.50, j.5.)


##
```

```
## 2.9.  A more complicated example with elements to eval
##
logo.jpg <- paste(R.home(), "doc", "html", "logo.jpg",
                  sep = .Platform$file.sep)
if(require(jpeg)){
  Rlogo <- readJPEG(logo.jpg)
# argument list for a call to rasterImage or rasterImageAdj
  RlogoLoc <- list(image=Rlogo,
    xleft.0 = c(NZ=176.5,CH=172,US=171,  CN=177,RU= 9.5,UK= 8),
    xleft.1 = c(NZ=176.5,CH=  9,US=-73.5,CN=125,RU= 37, UK= 2),
    ybottom.0=c(NZ=-37,  CH=-34,US=-34,  CN=-33,RU= 48, UK=47),
    ybottom.1=c(NZ=-37,  CH= 47,US= 46,  CN= 32,RU=55.6,UK=55),
    xright=quote(xleft+xinch(0.6)),
    ytop = quote(ybottom+yinch(0.6)),
    angle.0 =0,
    angle.1 =c(NZ=0,CH=3*360,US=5*360, CN=2*360,RU=360,UK=360)
    )

  RlogoInterp <- interpPairs(RlogoLoc,
            .proportion=rep(c(0, -1), c(2, 4)) )
# check

all.equal(names(RlogoInterp),
   c('image', 'xright', 'ytop', 'xleft', 'ybottom', 'angle'))

# NOTE:  'xleft', and 'ybottom' were created in interpPairs,
# and therefore come after 'xright' and 'ytop', which were
# already there.

##
## 2.10.  using envir
##
  RlogoDiag <- list(x0=quote(Rlogo.$xleft),
                  y0=quote(Rlogo.$ybottom),
                  x1=quote(Rlogo.$xright),
                  y1=quote(Rlogo.$ytop) )

  RlogoD <- interpPairs(RlogoDiag, .p=1,
                      envir=list(Rlogo.=RlogoInterp) )



all.equal(RlogoD, RlogoDiag)

}
##
## 2.11.  assign;  no interp but should work
##
tstAsgn <- as.list(quote(op <- (1:3)^2))
intAsgn <- interpPairs(tstAsgn, 1)

# check
intA. <- tstAsgn
```

```
names(intA.) <- c('X', 'X.3', 'X.2')

all.equal(intAsgn, intA.)


#   op <- par(...)
tstP <- quote(op <- par(mar=c(5, 4, 2, 2)+0.1))
tstPar <- as.list(tstP)
intPar <- interpPairs(tstPar, 1)

# check
intP. <- list(quote(`<-`), quote(op),
              quote(par(mar=c(5, 4, 2, 2)+0.1)) )
names(intP.) <- c("X", 'X.3', 'X.2')

all.equal(intPar, intP.)


intP. <- interpPairs(tstP)

all.equal(intP., tstP)


##
## NULL
##

all.equal(interpPairs(NULL), quote(NULL))
```

`match.data.frame` *Identify the row of* `y` *best matching each row of* `x`

Description

For each row of `x[, by.x]`, find the best matching row of `y[, by.y]`, with the best match defined by `grep.` and `split`.

`grep.` and `split` must either be `missing` or have the same length as `by.x` and `by.y`. If `grep.[i]` and `split[i]` are NA, do a complete match of `x[, by.x[i]]` and `y[, by.y[i]]`. Otherwise, for each row `j`, look for a match for `strsplit(x[j, by.x[i]], split[i])[[1]][1]` among `strsplit(y[, by.y[i]], split[i])`. See details.

Usage

```
match.data.frame(x, y, by, by.x=by, by.y=by, grep., split, sep=':')
```

Arguments

x, y	data.frames
by, by.x, by.y	names of columns of x and y to match.
grep.	a character vector of the type of match for each element of by.x and by.y. If NA, require a perfect match. Alternatives are grep and agrep to find a match for the first segment in strsplit(x, split=split[i]) among any of the segments of strsplit(y, split=split[i]). Use fixed=TRUE with the calls to these functions. NOTE: These alternatives are not examined if a unique match is found betweed x[, by.x[is.na(grep.) & is.na(split)]] and the corresponding columns of y.
split	A character vector of split characters to pass to strsplit; strsplit is not called if is.na(split).
sep	a sep argument to use with paste to produce a matching key for the columns of x and y for which perfect matches are required. If(missing(sep) && not(missing(grep.))) sep <- ' ' except where grep. = NA.

Details

1. Check by.x, by.y, grep. and split. If((missing(by.x) | missing(by.y)) && missing(by)) by <- names(x)

2. fullMatch <- (is.na(grep.) & is.na(split)). Create keyfx and keyfy by by pasting columns of x[, by.x[fullMatch]] and y[, by.y[fullMatch]]. Also create x. and y. = strsplit of x[, by.x[!fullMatch]].

3. Iterate over rows of x looking for the best match. This includes an inner loop over columns of x[, by.x[!fullMatch]], stopping on the first unique match. Return (-1) if no unique match is found.

Value

an integer vector of length nrow(x) containing the index of the best matching row of y or NA if no adequate match was found.

Author(s)

Spencer Graves

See Also

strsplit, is.na grep, agrep match, row.match, join, match_df classify

Examples

```
newdata <- data.frame(state=c("AL", "MI","NY"),
                      surname=c("Rogers", "Rogers", "Smith"),
                      givenName=c("Mike R.", "Mike K.", "Al"),
                      stringsAsFactors=FALSE)
reference <- data.frame(state=c("NY", "NY", "MI", "AL", "NY", "MI"),
                      surname=c("Smith", "Rogers", "Rogers (MI)",
                                "Rogers (AL)", "Smith", 'Jones'),
```

```
                      givenName=c("John", "Mike", "Mike", "Mike",
                                  "T. Albert", 'Al Thomas'),
                      stringsAsFactors=FALSE)
newInRef <- match.data.frame(newdata, reference,
       grep.=c(NA, 'agrep', 'agrep'))


all.equal(newInRef, c(4, 3, 5))
```

`matchQuote` *Match isolated quotes across records*

Description

Look for unmatched quotes in a character vector. If found, look for a matching quote starting the next characer string in the vector, possibly after a blank line. If found, merge the two strings and return the resulting shortened character vector.

Usage

```
matchQuote(x,  Quote='"', sep=' ', maxChars2append=2, ...)
```

Arguments

`x`	a character vector to scan for unmatched `Quotes`.
`Quote`	the `Quote` character that should appear in pairs
`sep`	`sep` argument passed to `paste` to combine pairs of successive lines with unmatched quotes.
`maxChars2append`	maximum number of characters in the following string to concatonate two adjacent strings (possibly separated by a blank line) with unmatched `Quotes`.
`...`	optional arguments for `gsub`

Details

This function was written to help parse data from the US Department of Health and Human Services on cyber-security breaches affecting 500 or more individuals. As of 2014-06-03 the csv version of these data included commas in quotes that are not `sep` characters, quotes that are not matched, lines with zero characters, followed by lines with 3 characters being a quote and a comma. This function was written to drop the blank lines and append the quote-comma line to the preceeding line so it contained matching quotes.

Value

The input character vector possibly shortened with the following attributes explaining what was found:

- unmatchedQuotes indices of the input `x` with an unmatched `Quote`.
- blankLinesDropped indices of the input `x` that were dropped because they (1) followed an unmatched `Quote` and (2) contained no non-blank characters.

  `quoteLinesAppended` indices of the input `x` that were concatonated with a preceeding line because the two lines contained unmatched `Quote` characters, and concatonating them produced a line with all `Quotes` matched.

  `ncharsAppended` an integer vector of the same length as `quoteLinesConcatonated` giving the number of characters in the second line concatonated onto the previous line.

Author(s)

Spencer Graves

See Also

`strsplit1` `delimMatch`

Examples

```
chvec <- c('abc', 'de"f', ' ', '",', 'g"h', 'matched"quotes"', '')
ch. <- matchQuote(chvec)

# check
chv. <- c('abc', 'de"f ",', 'g"h', 'matched"quotes"', '')
attr(chv., 'unmatchedQuotes') <- c(2, 4, 5)
attr(chv., 'blankLinesDropped') <- 3
attr(chv., 'quoteLinesAppended') <- 4
attr(chv., 'ncharsAppended') <- 2

all.equal(ch., chv.)
```

---

mergeUShouse.senate *Expand a dataset on some members of the US Congress to the entire membership*

Description

Merge a `data.frame` regarding some members of the US Congress with a `data.frame` with general information on all members.

Usage

```
mergeUShouse.senate(x, UScongress=UShouse.senate(),
        newrows="amount0",
        default=list(member=FALSE, amount=0, vote="notEligible",
                     incumbent=TRUE) )
```

Arguments

x	a data.frame to be merged with UScongress
UScongress	a data.frame to be merged with x.
newrows	name of a logical column to add that is TRUE for rows added to x and FALSE otherwise.
default	default values for columns of x identified by regexpr(names(default)[i], tolower(names(x))).

Details

1. keyx <- with(x, paste(houseSenate, state, District, sep=":"))
2. keyy <- with(UScongress(houseSenate, state, District, sep=":"))
3. notx <- !is.elememt(keyy, keyx)
4. Y <- UScongress[notx,]
5. add default columns to Y
6. if(!newrows is not in names(x))x <- cbind(x, newrows=FALSE)
7. Y[, newrows] <- TRUE
8. xY <- rbind(x, Y[c(names(x)])
9. replace 'Democrat' with 'Democratic' in xY[['Party']]
10. Look for NAs in "incumbent" who are nevertheless in UScongress; fix. Thus, if x[['incumbent']] is TRUE or FALSE, this value is not checked in UScongress; it's checked only if NA. The check consists of comparing names for a given Office:state:district between strsplit(x[['surname']], ' ')[[1]][1] and strsplit(UScongress[['surname']], ' ')[[1]][1] and similarly for givenName. This allows 'Rogers' in x[['surname']] to match 'Rogers (AL)' in UScongress[['surname']], etc. The algorithm is not perfect, but errors should be rare – and could be fixed manually.

Value

a data.frame combining x and UScongress as desired

Author(s)

Spencer Graves

See Also

merge UShouse.senate

Examples

```
tst <- data.frame(Office=factor(rep(c('House', 'Senate'), c(4, 2))),
         State=factor(c('Missouri', 'Minnesota', 'Tennessee',
                        'New York', rep('South Carolina', 2))),
         state=factor(c('MO', 'MN', 'TN', 'NY', 'SC', 'SC')),
         district=as.character(c(4, 1, 8, 18, 2, 3)),
         surname=c('Hartzler', 'Walz', 'Fincher', 'Maloney',
                   'Graham', 'DeMint'),
         givenName=c('Vicky', 'Timothy J.', 'Stephen Lee',
                   'Sean Patrick', 'Lindsey', 'Jim'),
         Party=c('Republican', 'Democrat', 'Republican', 'Democrat',
                 'Republican', 'Democrat'),
         CommitteeMember=rep(c(TRUE, FALSE), c(4, 2)),
         amount=c(5000, 2000, 29500, 1000, 1000, 11500),
         xvote=c('Y', 'N', 'Y', 'Y', 'notEligible', 'notEligible'),
         incumbent=NA, stringsAsFactors=FALSE )
tst2 <- mergeUShouse.senate(tst)

# A couple of simple tests;  don't test too much,
# because the results of UShouse.senate change,
# and we don't want this test to fail
# due to changes that don't affect Ecdat code

tst3 <- tst2[!tst2$amount0, c(1, 4:6, 8:10)]
row.names(tst) <- row.names(tst3)

## Not run:
all.equal(tst[c(1, 4:6, 8:10)], tst3)

## End(Not run)
# tst3[2] = state = factor with 56 levels,
# and tst[2] only has 5;  compare without this
```

`mergeVote` *Merge Roll Call Vote*

Description

Merge roll call vote record with a `data.frame` containing other information. The `vote` records are typically incomplete, so match first on `houseSenate` and `surname`. If this match is incomplete, try using `givenName`. If that fails, try `state` and `district`, which may not always be present in `vote`.

Usage

```
mergeVote(x, vote, Office="House", vote.x, check.x=TRUE)
```

Arguments

x	a data.frame whose columns include Office, surname, and givenName.
vote	a data.frame with column names which when forced tolower would match surname, givenname, and vote. However, the givenname may not be complete, so use it only if the surname is not sufficient.
Office	Either "House" or "Senate"; ignored if vote includes a column Office.
vote.x	name of a column of x containing a vote to be updated with the vote column of the vote data.frame. If missing and x has a column with a name matching "vote", then vote.x is that column. If missing but x has no such column, then append a column to x with the name of the vote column of the vote data.frame.
check.x	logical: If TRUE, check for rows of x[, vote.x] that are NOT in vote and throw an error if found.

Details

1. Parse vote.x to get the name of the column of x into which to write the vote column of the vote data.frame.

2. If the vote data.frame contains a column Office, ignore the Office argument. Otherwise, add the argument houseSenate as a column of vote.

3. Create keyx <- with(x, paste(Office, surname, sep=":")), keyx2 <- paste(keyx, givenName, sep=":"), keyx. <- paste(houseSenate, state, district, sep=":"), and similarly keyv, leyv2, and keyv. from vote.

4. Look for keyv in keyx. When a unique match is found, transfer the vote the vote column of x. When no match is found, try for keyv2 in keyx2 or keyv. in keyx. If those fail, print an error message with the information from vote on all failures and ask the user to add state and district informuation.

5. if(check.x), check for rows in x[, vote.x] that are NOT "notEligible" but are also not in vote: Throw an error if any are found.

Value

a data.frame with the same columns as x with its vote column modified per the vote argument.

Author(s)

Spencer Graves

See Also

mergeUShouse.senate

Examples

```
##
## 1.  Test good cases
##
votetst <- data.frame(
  surName=c('Smith', 'Jones', 'Graves', 'Jsn', 'Jsn', 'Gay'),
  givenName=c("Sam", "", "", "John", "John", ''),
  votex=factor(c('Y', 'N', 'abstain', 'Y', 'Y', 'Y')),
  State=factor(rep(c("CA", "", "SC", "NY"), c(1, 2, 1, 2))),
  district=rep(c("13", "1", "2", "1"), c(1, 2, 2, 1)),
  stringsAsFactors=FALSE )

x1 <- data.frame(
  Office=factor(rep(c("House", "Senate"), e=8)),
  state=rep(c("NY", "SC", "SD", "CA", "AK", "AR", "NY", "NJ"), 2),
  District=rep(c("2", "2", "At Large", "13", "1", "9", "1", "3"), 2),
  surname=rep(c('Jsn', 'Jsn', 'Smith', 'Smith', 'Jones',
       'Graves', 'Rx', 'Agnew'), 2),
  givenName=rep(c("John D.", "John J.",
    "Samual", "Samual", "Mary", "Mary", "Susan", 'Spiro'), 2),
  don=1:16, stringsAsFactors=FALSE)

x1. <- mergeVote(x1, votetst)

x2 <- cbind(x1, votex=factor( rep(
   c('Y', 'notEligible', 'Y', 'N', 'abstain', 'Y', 'notEligible'),
   c(2,1,1,1,1,1,9) ) ) )


all.equal(x1., x2)


##
## 2.  Test a case with a vote error in x
##

x1a <- cbind(x1, voterr=rep(
     c('notEligible', 'Y', 'notEligible'), c(7, 1, 8)))

x1a. <- try(mergeVote(x1a, votetst))


class(x1a.)=='try-error'
```

`missing0` *Missing or length 0*

Description

TRUE if x is missing or if length(x) is 0.

Usage

```
missing0(x)
```

Arguments

x	a formal argument as for missing

Details

Only makes sense called from within another function

Value

logical: TRUE if x is missing or if length(x) is 0.

Author(s)

Spencer Graves

See Also

missing

Examples

```
tstFn <- function(x)missing0(x)
# missing

all.equal(tstFn(), TRUE)


# length 0

all.equal(tstFn(logical()), TRUE)


# supplied

all.equal(tstFn(1), FALSE)
```

nchar0 *Zero characters or NULL*

Description

Returns TRUE if (is.null(x) || (length(x) == 0) || (max(nchar(x)) == 0)).

Usage

```
nchar0(x, ...)
```

Arguments

x	a character vector or something that can be coerced to mode character
...	optional arguments to be passed to nchar

Value

TRUE if x is either NULL or max(nchar(x)) == 0. FALSE otherwise.

Author(s)

Spencer Graves

See Also

nchar

Examples

```
all.equal(nchar0(NULL), TRUE)


all.equal(nchar0(character(0)), TRUE)


all.equal(nchar0(character(3)), TRUE)


all.equal(nchar0(c('a', 'c')), FALSE)
```

parseCommas *Convert character string with Dollar signs and commas to numerics*

Description

as.numeric of character strings after suppressing commas and dollar signs. This is a generalization of parseDollars.

Usage

```
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
## Default S3 method:
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
## S3 method for class 'data.frame'
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
```

Arguments

x	vector of character strings to be converted to numerics
pattern	regular expression to be replaced by replacement
replacement	Character string to substitute for each occurrence of pattern
acceptableErrorRate	number indicating the proportion of new NAs to that can be introduced and still assume it's numeric
...	optional arguments to pass to gsub

Details

as.numeric(gsub(x, ...))

The data.frame method outputs another data.frame with character or factor columns connverted to numerics using parseDollars whenever that can be done without creating NAs.

Value

Numeric vector converted from the character strings in x or a data.frame with columns that are obviously numbers in character format converted to numerics.

Author(s)

Spencer Graves

See Also

gsub as.numeric parseDollars

Examples

```
##
## 1.  a character vector
##
X2 <- c('-$2,500', '$5,000.50')
x2 <- parseDollars(X2)


all.equal(x2, c(-2500, 5000.5))


##
## A data.frame
##
chDF <- data.frame(let=letters[1:2], Dol=X2, dol=x2)
numDF <- parseCommas(chDF)

chkDF <- chDF
chkDF$Dol <- x2

all.equal(numDF, chkDF)
```

`parseDollars` *Convert character string with Dollar signs and commas to numerics*

Description

as.numeric of character strings after suppressing commas and dollar signs. This is a special case of `parseCommas`.

Usage

```
parseDollars(x, pattern='\$|,', replacement='', ...)
```

Arguments

`x`	vector of character strings to be converted to numerics
`pattern`	regular expression to be replaced by `replacement`
`replacement`	Character string to substitute for each occurrence of `pattern`
`...`	optional arguments to pass to `gsub`

Details

as.numeric(gsub(x, ...)). See also `parseCommas`.

Value

Numeric vector converted from `x`.

Author(s)

Spencer Graves

See Also

gsub as.numeric parseCommas

Examples

```
##
## 1.  a character vector
##
X2 <- c('-$2,500', '$5,000.50')
x2 <- parseDollars(X2)


all.equal(x2, c(-2500, 5000.5))


##
## A data.frame
##
chDF <- data.frame(let=letters[1:2], Dol=X2, dol=x2)
numDF <- parseCommas(chDF)

chkDF <- chDF
chkDF$Dol <- x2

all.equal(numDF, chkDF)
```

---

parseName *Parse surname and given name*

Description

Identify the presumed surname in a character string assumed to represent a name and return the result in a character matrix with "surname" followed by "givenName".

Usage

```
parseName(x, surnameFirst=(median(regexpr(',', x))>0),
          suffix=c('Jr.', 'I', 'II', 'III', 'IV', 'Sr.'),
          fixNonStandard=subNonStandardNames, ...)
```

Arguments

x	a character vector
surnameFirst	logical: If TRUE, the surname comes first followed by a comma (","), then the given name. If FALSE, parse the surname from a standard Western "John Smith, Jr." format. If missing(surnameFirst), use TRUE if half of the elements of x contain a comma.
suffix	character vector of strings that are NOT a surname but might appear at the end without a comma that would otherwise identify it as a suffix.
fixNonStandard	function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software. Use identity if this is not desired.
...	optional arguments passed to fixNonStandard

Details

If surnameFirst is FALSE:

1. If the last character is ")" and the matching "(" is 3 characters earlier, drop all that stuff. Thus, "John Smith (AL)" becomes "John Smith".

2. Look for commas to identify a suffix like Jr. or III; remove and call the rest x2.

3. split <- strsplit(x2, " ")

4. Take the last as the surname.

5. If the "surname" found per 3 is in suffix, save to append it to the givenName and recurse to get the actual surname.

NOTE: This gives the wrong answer with double surnames written without a hyphen in the Spanish tradition, in which, e.g., "Anistasio Somoza Debayle", "Somoza Debayle" give the (first) surnames of Anistasio's father and mother, respectively: The current algorithm would return "Debayle" as the surname, which is incorrect.

6. Recompose the rest with any suffix as the givenName.

Value

a character matrix with two columns: surname and givenName

Author(s)

Spencer Graves

See Also

strsplit identity

Examples

```
##
## 1.  Parse standard first-last name format
##
tst <- c('Joe Smith (AL)', 'Teresa Angelica Sanchez de Gomez',
         'John Brown, Jr.', 'John Brown Jr.',
         'John W. Brown III', 'John Q. Brown,I',
         'Linda Rosa Smith-Johnson', 'Anastasio Somoza Debayle',
         'Ra_l Vel_zquez')
library(Ecdat)
parsed <- parseName(tst)

tst2 <- matrix(c('Smith', 'Joe', 'Gomez', 'Teresa Angelica Sanchez de',
  'Brown', 'John, Jr.', 'Brown', 'John, Jr.',
  'Brown', 'John W., III', 'Brown', 'John Q., I',
  'Smith-Johnson', 'Linda Rosa', 'Debayle', 'Anastasio Somoza',
  'Velazquez', 'Raul'),
  ncol=2, byrow=TRUE)
# NOTE:  This second to last example is in the Spanish tradition
# and is handled incorrectly by the current algorithm.
# The correct answer should be "Somoza Debayle", "Anastasio".
# However, fixing that would complicate the algorithm excessively for now.
colnames(tst2) <- c("surname", 'givenName')


all.equal(parsed, tst2)


##
## 2.  Parse "surname, given name" format
##
tst3 <- c('Smith (AL),Joe', 'Sanchez de Gomez, Teresa Angelica',
     'Brown, John, Jr.', 'Brown, John W., III', 'Brown, John Q., I',
     'Smith-Johnson, Linda Rosa', 'Somoza Debayle, Anastasio',
     'Vel_zquez, Ra_l')
tst4 <- parseName(tst3)

tst5 <- matrix(c('Smith', 'Joe', 'Sanchez de Gomez', 'Teresa Angelica',
  'Brown', 'John, Jr.', 'Brown', 'John W., III', 'Brown', 'John Q., I',
  'Smith-Johnson', 'Linda Rosa', 'Somoza Debayle', 'Anastasio',
  'Velazquez', 'Raul'),
  ncol=2, byrow=TRUE)
colnames(tst5) <- c("surname", 'givenName')


all.equal(tst4, tst5)
```

Ping *ping a Uniform resource locator (URL)*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

ping a Uniform resource locator (URL) or Internet Protocol (IP) address.

NOTE: Some Internet Service Providers (ISPs) play games with "ping". That makes the results of `Ping` unreliable.

Usage

```
Ping(url, pingArgs='', warn=NA,
     show.output.on.console=FALSE)
```

Arguments

`url`	a character string of a URL or IP address to ping. If `url` is a vector of length greater than 1, only the first component is used.
`pingArgs`	arguments to pass to the `ping` command of typical operating systems via pingResult <- system(paste('ping', pingArgs, url), intern=TRUE, ...)
`warn`	value for `options('warn')` during the call to `system`. `NA` to not change `options('warn')` during this call.
`show.output.on.console`	argument for `system`.

Details

1. urlSplit0 <- strsplit(url, '://')[[1]]
2. urlS0 <- urlSplit0[min(2, length(urlSplit0))]
3. host <- strsplit(urlS0, '/')[[1]][1]
4. pingCmd <- paste('ping', pingArgs, host)
5. system(pingCmd, intern=TRUE, ...)

Value

list with the following components:

`rawResults`	character vector of the raw results from the ping command
`rawNumbers`	numeric vector of the times measured
`counts`	numeric vector of numbers of packets sent, received, and lost
`p.lost`	proportion lost = lost / sent
`stats`	numeric vector of min, avg (mean), max, and mdev (standard deviation) of the measured round trip times

Author(s)

Spencer Graves

See Also

system, options

Examples

```
##
## Some ISPs play games with ping.
## Therefore, the results are not reliable.
##
## Not run:
##
## good
##
(google <- Ping('http://google.com/ping works on host not pages'))

\dontshow{stopifnot(}
with(google, (counts[1]>0) && (counts[3]<1))
\dontshow{)}

##
## ping oops <<-- at one time, this failed.
##      However, with some ISPs, it works, so don't test it.
##

##
(couldnotfindhost <- Ping('oops'))

\dontshow{stopifnot(}
with(couldnotfindhost,
     length(grep('could not find host', rawResults))>0)
\dontshow{)}

##
## impossible, but not so obvious
##
(requesttimedout <- Ping('requesttimedout.com'))

\dontshow{stopifnot(}
with(requesttimedout, (counts[1]>0) && (counts[2]<1) &&
         (counts[3]>0))
\dontshow{)}

## End(Not run)
```

---

qqnorm2 *Normal Probability Plot with Multiple Symbols*

Description

Create a normal probability plot with different symbols for the values of another variable. qqnorm2 produces an object of class qqnorm2, whose plot method produces the plot.

Usage

```
qqnorm2(y, z, plot.it=TRUE, datax=TRUE, pch=NULL, ...)
## S3 method for class 'qqnorm2'
plot(x, y, ...)
## S3 method for class 'qqnorm2'
lines(x, ...)
## S3 method for class 'qqnorm2'
points(x, ...)
```

Arguments

y	For qnorm2, y is a numeric vector for which a normal probability plot is desired. For plot.qqnorm2, y is ignored; it is included, because the generic plot function requires it.
z	A variable to indicate different plotting symbols.
plot.it	logical: Should the result be plotted?
datax	The datax argument of qqnorm: If TRUE, the data are displayed on the horizontal rather than the vertical axis. (The default value for datax is the opposite of that for qqnorm.)
x	an object of class qqnorm2.
pch	a named vector of the plotting symbols to be used with names corresponding to the levels of z. By default, if z takes levels FALSE and TRUE (or 0 and 1), pch=c(4, 1) to plot a "x" for FALSE and "o" for TRUE. If z assumes integer values between 0 and 255, by default, the symbols are chosen as described with points. Otherwise, by default, z is coerced to character, and the result is plotted. If pch is provided, it must eitehr have names corresponding to levels of z, or z must be integers between 1 and length(pch).
...	Optional arguments. For plot.qqnorm2, they are passed to plot. For qqnorm2, they are passed to qqnorm and to plot.qqnorm2.

Details

For qqnorm2:

1. q2 <- qqnorm(y, datax=datax, ...)

2. q2[["z"]] <- z

3. q2[["pch"]] gets whatever pch decodes to.

4. Silently return(list(x, y, z, pch)), where "x" and "y" are as returned by qqnorm in step 1 above.

For plot.qqnorm2, plot(x, y, pch=pch[z], ...). For lines.qqnorm2, lines(x, y, pch=pch[z], ...). For points.qqnorm2, points(x, y, pch=pch[z], ...).

Value

qqnorm2 returns a list with components, x, y, z, and pch.

Author(s)

Spencer Graves

See Also

qqnorm, qqnorm2s, plot points lines

Examples

```
##
## a simple test data.frame to illustrate the plot
## but too small to illustrate qqnorm concepts
##
tstDF <- data.frame(y=1:3, z1=1:3, z2=c(TRUE, TRUE, FALSE),
                    z3=c('tell', 'me', 'why'), z4=c(1, 2.4, 3.69) )
# plotting symbols circle, triangle, and "+"
qn1 <- with(tstDF, qqnorm2(y, z1))

# plotting symbols "x" and "o"
qn2 <- with(tstDF, qqnorm2(y, z2))

# plotting with "-" and "+"
qn. <- with(tstDF, qqnorm2(y, z2, pch=c('FALSE'='-', 'TRUE'='+')))

# plotting with "tell", "me", "why"
qn3 <- with(tstDF, qqnorm2(y, z3))

# plotting with the numeric values
qn4 <- with(tstDF, qqnorm2(y, z4))

##
## test plot, lines, points
##
plot(qn4, type='n') # establish the scales
lines(qn4)          # add a line
points(qn4)         # add points

##
## Check the objects created above
##
# check qn1
qn1. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn1.$xlab <- 'y'
qn1.$ylab <- 'Normal scores'
qn1.$z <- tstDF$z1
qn1.$pch <- 1:3
names(qn1.$pch) <- 1:3
qn11 <- qn1.[c(3:4, 1:2, 5:6)]
```

```
class(qn11) <- 'qqnorm2'

all.equal(qn1, qn11)


# check qn2
qn2. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn2.$xlab <- 'y'
qn2.$ylab <- 'Normal scores'
qn2.$z <- tstDF$z2
qn2.$pch <- c('FALSE'=4, 'TRUE'=1)
qn22 <- qn2.[c(3:4, 1:2, 5:6)]
class(qn22) <- 'qqnorm2'

all.equal(qn2, qn22)


# check qn.
qn.. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn..$xlab <- 'y'
qn..$ylab <- 'Normal scores'
qn..$z <- tstDF$z2
qn..$pch <- c('FALSE'='-', 'TRUE'='+')
qn.2 <- qn..[c(3:4, 1:2, 5:6)]
class(qn.2) <- 'qqnorm2'

all.equal(qn., qn.2)


# check qn3
qn3. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn3.$xlab <- 'y'
qn3.$ylab <- 'Normal scores'
qn3.$z <- as.character(tstDF$z3)
qn3.$pch <- as.character(tstDF$z3)
names(qn3.$pch) <- qn3.$pch
qn33 <- qn3.[c(3:4, 1:2, 5:6)]
class(qn33) <- 'qqnorm2'

all.equal(qn3, qn33)


# check qn4
qn4. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn4.$xlab <- 'y'
qn4.$ylab <- 'Normal scores'
qn4.$z <- tstDF$z4
qn44 <- qn4.[c(3:4, 1:2, 5)]
qn44$pch <- NULL
class(qn44) <- 'qqnorm2'

all.equal(qn4, qn44)
```

qqnorm2s *Normal Probability Plot with Multiple Lines and Multiple Symbols*

Description

Create a normal probability plot with multiple lines for different variables and different symbols for the values of another variable. qqnorm2s produces an object of class qqnorm2s, whose plot method produces the plot.

Usage

```
qqnorm2s(y, z, data., plot.it=TRUE, datax=TRUE, outnames=y,
         pch=NULL, col=c(1:4, 6), legend.=NULL, ...)
## S3 method for class 'qqnorm2s'
plot(x, y, ...)
```

Arguments

y	For qnorm2s, y is a character vector of names of columns of data. for which normal probability plots are desired. data. is either a data.frame or a list of data.frames of the same length as y, with y[i] being the name of a column of the data.frame data.[[i]]. z is a similar character vector of names of columns of data., which identify symbols for plotting different points in a normal probability plot. The lengths of y, and z must match the number of data.frames in data.; if not, the lengths of the shorter are replicated to the length of the longest before computations begin. For plot.qqnorm2, y is ignored; it is included, because the generic plot function requires it.
z	A character vector giving the names of columns of data. to indicate different plotting symbols. z should be the same length as y and must equal the number of data.frames in the list data. of data.frames. If not, the shorter are replicated to the length of the longer.
data.	a data.frame or a list of data.frames with columns named in y and z.
plot.it	logical: Should the result be plotted?
datax	The datax argument of qqnorm: If TRUE, the data are displayed on the horizontal rather than the vertical axis. (The default value for datax is the opposite of that for qqnorm.)
outnames	Names for the components of the qqnorm2s object returned by the qqnorm2s function.
pch	a named vector of the plotting symbols to be used with names corresponding to the levels of z. By default, if z takes levels FALSE and TRUE (or 0 and 1), pch=c(4, 1) to plot a "x" for FALSE and "o" for TRUE.

If z assumes integer values between 0 and 255, by default, the symbols are chosen as described with points.

Otherwise, by default, z is coerced to character, and the result is plotted.

If pch is provided, it must eitehr have names corresponding to levels of z, or z must be integers between 1 and length(pch).

col — A vector indicating the colors corresponding to each element of y. Defaults to rep(c(1:4, 6), length=length(y)), with 1:4 and 6 being black, red, green, blue, and pink.

x — an object of class qqnorm2.

legend. — A list with components pch and col providing information for legend to identify the plotting symbols (pch) and colors (col).

By default, pch = list(x='right', legend=names(qq2s[[1]][['pch']]), pch=qq2s[[1]][['pch']]), where qq2s is described below in details.

Similarly, by default, lines = list(x='bottomright', legend=y, lty=1, pch=NA, col=qq2s[[1]][['col']]).

... — Optional arguments.

For plot.qqnorm2s, they are passed to plot.

For qqnorm2s, they are passed to qqnorm2 and to plot.qqnorm2s.

Details

For qqnorm2s:

1. Create qq2s = a list of objects of class qqnorm2
2. Add legend. to qq2s.
3. class(qq2s) <- 'qqnorm2s'
4. if(plot.it)plot(qq2s, ...)
5. Silently return(qq2s).

For plot.qqnorm2s, create a plot with one line for each variable named in y.

Value

qqnorm2s returns a named list with components of class qqnorm2 with names = y with each component having an additional component col plus one called "legend.".

Author(s)

Spencer Graves

See Also

qqnorm2 plot

Examples

```
##
## One data.frame
##
tstDF2 <- data.frame(y=1:3, y2=3:5, z2=c(TRUE, TRUE, FALSE),
                     z3=c('tell', 'me', 'why'), z4=c(1, 2.4, 3.69) )
# produce the object and plot it
Qn2 <- qqnorm2s(c('y', 'y2'), 'z2', tstDF2)

# plot the object previously created
plot(Qn2)

# Check the object
qy <- with(tstDF2, qqnorm2(y, z2, type='b'))
qy$col <- 1
qy2 <- with(tstDF2, qqnorm2(y2, z2, type='b'))
qy2$col <- 2
legend. <- list(pch=list(x='right', legend=c('FALSE', 'TRUE'),
                pch=c('FALSE'=4, 'TRUE'= 1)),
                col=list(x='bottomright', legend=c('y', 'y2'),
                lty=1, col=1:2))
Qn2. <- list(y=qy, y2=qy2, legend.=legend.)
class(Qn2.) <- 'qqnorm2s'


all.equal(Qn2, Qn2.)

##
## Two data.frames
##
tstDF2b <- tstDF2
tstDF2b$y <- c(0.1, 0.1, 9)
Qn2b <- qqnorm2s('y', 'z2', list(tstDF2, tstDF2b),
           outnames=c('ok', 'oops'), log='x' )
```

`rasterImageAdj` *rasterImage adjusting to zero distortion*

Description

Call `rasterImage` to plot `image` from `(xleft, ybottom)` to either `xright` or `ytop`, shrinking one toward the center to avoid distortion.

`angle` specifies a rotation around the midpoint `((xleft+xright)/2, (ybottom+ytop)/2)`. This is different from `rasterImage`, which rotates around `(xleft, ybottom)`.

NOTE: The code may change in the future. The visual image with rotation looks a little off in the examples below, but the code seems correct. If you find an example where this is obviously off, please report to the maintainer – especially if you find a fix for this.

Usage

```
rasterImageAdj(image, xleft=par('usr')[1], ybottom=par('usr')[3],
     xright=par('usr')[2], ytop=par('usr')[4], angle = 0,
     interpolate = TRUE, xsub=NULL, ysub=NULL, ...)
```

Arguments

image	a raster object, or an object that can be coerced to one by as.raster.
xleft	a vector (or scalar) of left x positions.
ybottom	a vector (or scalar) of bottom y positions.
xright	a vector (or scalar) of right x positions.
ytop	a vector (or scalar) of top y positions.
angle	angle of rotation in degrees, anti-clockwise about the centroid of image. NOTE: rasterImage rotates around (xleft, ybottom). rasterImage rotates around the center ((xleft+xright)/2, (ybottom+ytop)/2). See the examples.
interpolate	a logical vector (or scalar) indicating whether to apply linear interpolation to the image when drawing.
xsub, ysub	subscripts to subset image
...	graphical parameters (see par).

Details

1. imagePixels = number of (x, y) pixels in image. Do this using dim(as.raster(image))[2:1], because the first dimension of image can be either x or y depending on class(image). For example link[EBImage]{Image} returns dim with x first then y and an optional third dimension for color. A simple 3-dimensional array is assumed by rasterImage to have the y dimension first. as.raster puts all these in a standard format with y first, then x.

2. imageUnits <- c(x=xright-xleft, ytop-ybottom)

3. xyinches = (x, y) units per inch in the current plot, obtained from xyinch.

4. Compute pixel density (pixels per inch) in both x and y dimension: pixelsPerInch <- imagePixels * xyinches / imageUnits.

5. Compute imageUnitsAdj solving 4 for imageUnits and replacing pixelsPerInch by the max pixel density: imageUnitsAdj <- imagePixels * xyinches / max(pixelsPerInch).

6. (dX, dY) = imageUnitsAdj/2 = half of the (width, height) in plotting units.

7. cntr = (xleft, ybottom) + (dX, dY).

xleft0 = cntr[1]+sin((angle-90)*pi/180)*dX*sqrt(2); ybottom0= cntr[2]-cos((angle-90)*pi/180)*dY*sqrt(2);

(xright0, ytop0) = (upper right without rotation about lower left) xright0 = xleft0+imageUnitsAdj[2] ytop0 = ybottom0+imageUnitsAdj[2]

8. rasterImage(image, xleft0, ybottom0, xright0, ytop0, angle, interpolate, ...)

Value

a named vector giving the values of `xleft`, `ybottom`, `xright`, and `ytop` passed to `rasterImage`. (`rasterImage` returns `NULL`, at least for some inputs.) This shows the adjustment, shrinking toward the center and rotating as desired.

Author(s)

Spencer Graves

See Also

`rasterImage`

Examples

```
# something to plot
logo.jpg <- paste(R.home(), "doc", "html", "logo.jpg",
                  sep = .Platform$file.sep)
if(require(jpeg)){
##
## 1.  Shrink as required
##
  Rlogo <- readJPEG(logo.jpg)

  all.equal(dim(Rlogo), c(76, 100, 3))


  plot(1:2)
# default
  rasterImageAdj(Rlogo)

  plot(1:2, type='n', asp=0.75)
# Tall and thin
  rasterImage(Rlogo, 1, 1, 1.2, 2)
# Fix
  rasterImageAdj(Rlogo, 1.2, 1, 1.4, 2)

# short and wide
  rasterImage(Rlogo, 1.4, 1, 2, 1.2)
# Fix
  rasterImage(Rlogo, 1.4, 1.2, 2, 1.4)
##
## 2.  rotate
##
#  2.1.  angle=90:  rasterImage left of rasterImageAdj
  plot(0:1, 0:1, type='n', asp=1)
  rasterImageAdj(Rlogo, .5, .5, 1, 1, 90)
  rasterImage(Rlogo, .5, .5, 1, 1, 90)
#  2.2.  angle=180:  rasterImage left and below
  plot(0:1, 0:1, type='n', asp=1)
  rasterImageAdj(Rlogo, .5, .5, 1, 1, 180)
  rasterImage(Rlogo, .5, .5, 1, 1, 180)
```

```
#  2.3.  angle=270:  rasterImage below
  plot(0:1, 0:1, type='n', asp=1)
  rasterImageAdj(Rlogo, .5, .5, 1, 1, 270)
  rasterImage(Rlogo, .5, .5, 1, 1, 270)
##
## 3.  subset
##
dim(Rlogo)
# 76 100 3
Rraster <- as.raster(Rlogo)
dim(Rraster)
# 76 100:
# x=1:100, left to right
# y=1:76, top to bottom
rasterImageAdj(Rlogo, 0, 0, .5, .5, xsub=40:94)
}
```

`read.testURLs` *Read a file produced by testURLs*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

`read.table(file.)` and return the result as an object of class c('testURLs', 'data.frame').

Usage

```
read.testURLs(file.='testURLresults.csv', ...)
```

Arguments

`file.`	Name of a CSV file to read
`...`	optional arguments for `read.csv`.

Details

dat <- read.csv(file., ...)

class(dat) <- c('testURLsFile', 'data.frame')

Value

a `data.frame` from the file written by `testURLs`, of the same format as the `testResults` attribute of the `testURLs` object returned by `testURLs`.

Author(s)

Spencer Graves

See Also

read.csv

Examples

```
# Test only 2 web sites, not the default 4,
# and test only twice, not the default 10 times:
tst <- testURLs(c(
 PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov/representatives"),
    n=2, maxFail=2)

# The above should have created a file 'testURLresults.csv'
# in the working directory.  Read it.

dat <- read.testURLs()
```

---

read.transpose *Read a data table in transpose form*

Description

Read a text (e.g., csv) file, find rows with more than 3 sep characters. Parse the initial contiguous block of those into a matrix. Add attributes headers, footers, and a summary.

The initial application for this function is to read Table 6.16. Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce.

Usage

```
read.transpose(file, header=TRUE, sep=',',
               na.strings='---', ...)
```

Arguments

file	the name of a file from which the data are to be read.
header	Logical: Is the second column of the identified data matrix to be interpreted as variable names?
sep	The field space separator charactor.
na.strings	character string(s) that translate into NA
...	optional arguments for strsplit

Details

1. txt <- readLines(file)
2. Split into fields.
3. Identifiy headers, Data, footers.
4. Recombine the second component of each Data row if necessary so all have the same number of fields.
5. Extract variable names
6. Numbers?
7. return the transpose

Value

A matrix of the transpose of the rows with the max number of fields with attributes 'headers', 'footers', 'other', and 'summary'. If this matrix can be coerced to numeric with no NAs, it will be. Otherwise, it will be left as character.

Author(s)

Spencer Graves

References

Table 6.16. Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce. To get this table from www.bea.gov, under "U.S. Economic Accounts", first select "Corporate Profits" under "National". Then next to "Interactive Tables", select, "National Income and Product Accounts Tables". From there, select "Begin using the data...". Under "Section 6 - income and employment by industry", select each of the tables starting "Table 6.16". As of February 2013, there were 4 such tables available: Table 6.16A, 6.16B, 6.16C and 6.16D. Each of the last three are available in annual and quarterly summaries. The `USFinanceIndustry` data combined the first 4 rows of the 4 annual summary tables.

See Also

`read.table` `readLines` `strsplit`

Examples

```
#  Find demoFiles/*.csv
demoDir <- system.file('demoFiles', package='Ecdat')
(demoCsv <- dir(demoDir, pattern='csv$', full.names=TRUE))

# Use the fourth example
# to ensure the code will handle commas in a name
# and NAs
nipa6.16D <- read.transpose(demoCsv[4])
str(nipa6.16D)
```

readCookPVI *Read Cook Partisan Voting Index*

Description

Read tables of the Cook Partisan Voting Index and returns a list with components 'House' and 'Senate'. readCookPVI. returns the tables with the names of the current incumbents per readUShouse and readUSsenate; readCookPVI tables do not include the names of the incumbents.

Usage

```
readCookPVI(url.=
"http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index")
readCookPVI.(url.=
"http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
      UShouse=readUShouse(), USsenate=readUSsenate(), ...)
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired lists.
UShouse, USsenate	data.frames as returned by readUShouse and readUSsenate, respectively.
...	optional arguments passed to readUShouse and readUSsenate.

Details

The primary source for these data is the Cook Political Report web site. However, the current URL we have for these data on that web site includes "2012" in the title. If and when the numbers are updated, we would expect that file name to change.

To avoid that problem the code is currently set to read from the Wikipedia article on "Cook Partisan Voting Index".

The algorithm reads the web site into a list, finds the desired tables on the list, then parses and formats them as desired. Then it merges the results with UShouse and USsenate.

Value

A list with components "House" and "Senate". Each contains a data.frame. The "House" data.frame returned by readCookPVI includes the following columns:

State	name of the state
District	District, e.g, 1st, 2nd, At-Large
PVInum	PVI as a number ranging from roughly 50 to 150. 100 means that the vote split in that district was within 0.5 percent of the national average. 101 means that it tilts 1 percent (after rounding) to Republican. 98 means that it tilts 1 percent to Democratic; 99 is not used.

`PVIchar` PVI rating in character format. For example, 'D+1' means that the vote tilted 1 percent toward Democratic more than the national average. 'R+1' means that it tilted 1 percent toward Republican.

`PartyOfRepresentative` Party of the incumbent, either 'Republican' or 'Democratic'

The 'Senate' data.frame includes the following columns:

`State` name of the state

`PVInum` PVI numeric, as for 'House'

`PVIchar` PVI rating in character format, as for 'House'

`PartyOfGovernor` Party of the Governor of the state

`PartyInSenate` party of the incumbent senators, either 'Republican', Democratic', or 'Both'.

`houseBalanceNum` House balance as a number with 0 = 100 percent Democratic, 99.9 = 100 percent Republican, and 500 for the same number of Republicans as Democrats.

`houseBalanceChar` Count by pary in the house delegation for that state, e.g., '6R, 1D' for 6 Republicans and 1 Democrat.

`readCookPVI.` adds to the above the information returned by `readUShouse` and `readUSsenate`.

Author(s)

Spencer Graves

Source

Wikipedia, "Cook Partisan Voting Index" The Cook Political Report

See Also

`readUShouse`, `readUSsenate`

Examples

```
## Not run:
CookPVI <- readCookPVI()

## End(Not run)

if(!fda::CRAN()){
CookPVI. <- readCookPVI.()
}
```

readFinancialCrisisFiles *banking crisis data and function to read financial crisis files*

Description

Read financial crisis data in files described by an object of class financialCrisisFiles. This is designed to read Excel files describing financial crises since 1800 downloaded from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

bankingCrises is a data.frame created by readFinancialCrisisFiles() using 3 files downloaded from http://www.reinhartandrogoff.com and 1 obtained from Prof. Reinhart in January 2013.

Usage

```
readFinancialCrisisFiles(files, crisisType=7, ...)
```

Arguments

files	an object of class financialCrisisFiles.
crisisType	an integer (vector) between 1 and 8 indicating the type of data to be retrieved: 1=independence year (not a crisis but an indicator), 2=currency, 3=inflation, 4=stock market, 5=domestic soveregn debt crisis, 6=external sovereign debt crisis, 7=banking, 8=tally. ("Type" 1 = year.) These are all 0 or 1 indicating the presence of the event in the given year. Type 8 = sum of types 2 through 7.
...	arguments to pass with file and sheet name to read.xls when reading a sheet of an MS Excel file. This is assumed to be the same for all sheets of all files. If this is not the case, the resulting financialCrisisFiles object will have to be edited manually before using it to read the data.

Details

Reinhart and Rogoff (http://www.reinhartandrogoff.com) provide numerous data sets analyzed in their book, "This Time Is Different: Eight Centuries of Financial Folly". Of interest here are data on financial crises of various types for 70 countries spanning the years 1800 - 2010, downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

The function financialCrisisFiles produces a list of class financialCrisisFiles describing different Excel files in very similar formats with one sheet per Country and a few extra descriptor sheets. The data object FinancialCrisisFiles is the default output of that function.

readFinancialCrisisFiles reads the sheets for the individual countries.

Value

If length(crisisType) == 1, a data.frame is returned with the first column being year, and with one other column containing the data for that crisisType for each country.

If length(crisisType) > 1, a list is returned containing a data.frame for each country.

Author(s)

Spencer Graves

Source

http://www.reinhartandrogoff.com

References

Carmen M. Reinhart and Kenneth S. Rogoff (2009) This Time Is Different: Eight Centuries of Financial Folly, Princeton U. Pr.

See Also

read.xls financialCrisisFiles

Examples

```
##
## Recreate / update the data object BankingCrises
##
library(Ecdat)

## Not run:
bankingCrises <- readFinancialCrisisFiles(FinancialCrisisFiles)

## End(Not run)

##
## Toy example using local data to check the code
## and illustrate returning all the data not just one crisisType
##
Ecdat.demoFiles <- system.file('demoFiles', package='Ecdat')
Ecdat.xls <- dir(Ecdat.demoFiles, pattern='xls$',
                 full.names=TRUE)
if(require(gdata)){
  tst <- financialCrisisFiles(Ecdat.xls)

  bankingCrises.tst <- readFinancialCrisisFiles(tst)
# optional tests if not CRAN
  if(!fda::CRAN()){
  allCrises.tst <- readFinancialCrisisFiles(tst, 1:8)

# Manually construct tst from allCrises.tst
  tst2 <- data.frame(year=1800:1999)
  tst2$Algeria <- as.numeric(allCrises.tst$Algeria[-(1:12), 8])
  tst2$CentralAfricanRep <- as.numeric(
                   allCrises.tst$CentralAfricanRep[-(1:12), 8])
  tst2$Taiwan <- as.numeric(allCrises.tst$Taiwan[-(1:11), 8])
  tst2$UK <- as.numeric(allCrises.tst$UK[-(1:11), 8])
```

```
all.equal(bankingCrises.tst, tst2)


# check
data(bankingCrises)

all.equal(bankingCrises.tst,
   bankingCrises[1:200, c('year', 'Algeria', 'CentralAfricanRep',
                          'Taiwan', 'UK')])

}
}
```

---

readNIPA *Read a National Income and Product Accounts data table*

Description

Read multiple files with data in rows using read.transpose and combine the initial columns.

Usage

```
readNIPA(files, sep.footnote='/', ...)
```

Arguments

files	A character vector of names of files from which the data are to be read using read.transpose.
sep.footnote	a single character to identify footnote references in the variable names in some but not all of files.
...	optional arguments for read.transpose

Details

This is written first and foremost to facilitate updating USFinanceIndustry from Table 6.16: Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce. As of February 2013, this table can be obtained from http://www.bea.gov: Under "U.S. Economic Accounts", first select "Corporate Profits" under "National". Then next to "Interactive Tables", select, "National Income and Product Accounts Tables". From there, select "Begin using the data...". Under "Section 6 - income and employment by industry", select each of the tables starting "Table 6.16". As of February 2013, there were 4 such tables available: Table 6.16A, 6.16B, 6.16C and 6.16D. Each of the last three are available in annual and quarterly summaries. The USFinanceIndustry data combined the first 4 rows of the 4 annual summary tables.

This is available in 4 separate files, which must be downloaded and combined using readNIPA. The first three of these are historical data and are rarely revised. For convenience and for testing, they are provided in the demoFiles subdirectory of this Ecdat package.

It has not been tested on other data but should work for annual data with a sufficiently similar structure.

The algorithm proceeds as follows:

1. Data <- lapply(files, read.transpose)

2. Is `Data` a list of numeric matrices? If no, print an error.

3. cbind common initial variables, averaging overlapping years, reporting percent difference

4. attributes: stats from files and overlap. Stats include the first and last year and the last revision date for each file, plus the number of years overlap with the previous file and the relative change in the common files kept between those two files.

Value

a `matrix` of the common variables

Author(s)

Spencer Graves

References

United States Department of Commerce Bureau of Economic Analysis National Income and Product Account tables

See Also

`read.table` `readLines` `strsplit`

Examples

```
#  Find demoFiles/*.csv
demoDir <- system.file('demoFiles', package='Ecdat')
(demoCsv <- dir(demoDir, pattern='csv$', full.names=TRUE))

nipa6.16 <- readNIPA(demoCsv)
str(nipa6.16)
```

`readUShouse` *Read the list of representatives in the United States House of Representatives*

Description

Read the list of representatives in the United States House of Representatives.

Usage

```
readUShouse(url.="http://house.gov/representatives/",
   nonvoting=c('American Samoa', 'District of Columbia',
               'Guam', 'Northern Mariana Islands', 'Puerto Rico',
               'Virgin Islands'),
   fixNonStandard=subNonStandardNames, ...)
```

Arguments

`url.`	Universal resource locator to be read and processed to obtain the desired list
`nonvoting`	Character vector of the names of US territories that send a nonvoting delegate to the US House.
`fixNonStandard`	function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software. Use `identity` if this is not desired.
`...`	optional arguments passed to `fixNonStandard`

Details

1. `House.gov <- readHTMLTable(url)`. As of April 2013, this is a list of 80 tables. The first 56 are for the 50 states and 6 territories. The remaining 24 are for the first letter of the last name of the representatives.

2. Use `rbind` to collapse these into 2 tables. The first has the district as a number without identifying the state (because that was with the names of the first 56 tables in House.gov). The second has the state names but with the district numbers in a form not easily parsed.

3. Obtain the state names from the second table to match the names of the representatives in the first.

4. Add a `nonvoting` column for those "States" in `nonvoting`.

5. Look for and fix `surname` and `givenName` with nonstandard characters using `fixNonStandard`.

Value

`readUShouse` returns a `data.frame` with the following columns:

`State`	A factor identifying the state or territory the person represents
`state`	2-letter US Postal Service abbreviation for the state or territory
`district`	the character vector identifying the district each person represents. This is either an integer in character format or 0 for "At Large".
`Name`	A character vector giving the name of each representative (in surname, given name format)
`party`	a factor identifying the party affiliation of each representative ("D" or "R").
`Room`	character vector identifying the room number of the office
`Phone`	character vector giving the phone number
`Committees`	a character vector giving the committee assignments of each representative

surname	character vector giving the surname of each representative
givenName	given name of each representative (possibly with middle name or initial, a nickname, and a suffix like "Jr.")

Author(s)

Spencer Graves

See Also

getURL readHTMLTable readUSsenate UShouse.senate parseName readUSstateAbbreviations subNonStandardNames readCookPVI

Examples

```
if(!fda::CRAN()){
UShouse <- readUShouse()
}
```

---

readUSsenate *Read the list of elected officials in the United States Senate*

Description

Read the list of elected officias in the United States Senate.

Usage

```
readUSsenate(url.=
"http://en.wikipedia.org/wiki/List_of_current_United_States_Senators",
    stateAbbreviations=Ecdat::USstateAbbreviations,
   fixNonStandard=subNonStandardNames, ...)
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired list NOTE: On April 26, 2013 the obvious naive use of readHTMLTable worked with the Wikipedia article on the US Senate but did not work with "senate.gov".
stateAbbreviations	a data.frame giving names and alternative codes for US states and territories. This must have a column named "Name" giving the names of all 50 states as they appear on the url and another whose name includes "USPS" giving the correspoding 2-letter codes.
fixNonStandard	function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software.
...	optional arguments passed to fixNonStandard

Details

1. `Senate <- readHTMLTable(url)`
2. Use `camelParse` to remove duplication in `Name`.
3. Look for and fix `surname` and `givenName` with nonstandard characters using `fixNonStandard`.

Value

`readUSsenate` returns a `data.frame` with the following columns:

`State`	A factor identifying the state the person represents
`state`	A factor giving the 2-letter USPS code for the state represented
`Class`	"1", "2", or "3" for election in the 6-year cycle including 2008, 2010, or 2012, respectively.
`Name`	A character vector giving the name of each representative (in surname, given name format)
`Party`	a factor identifying the party affiliation of each representative ("Democrat", "Republican", or "Independent").
`Experience`	character vector highlighting prior experience.
`assumedOffice`	character vector giving the date assumed office
`Born`	a character vector giving the year of birth
`endOffice`	a character vector giving the last day in the present term.
`surname`	character vector giving the surname of each representative
`givenName`	given name of each representative (possibly with middle name or initial, a nickname, and a suffix like "Jr.")

Author(s)

Spencer Graves

See Also

`getURL` `readHTMLTable` `camelParse` to remove duplication in `Name` `readUShouse` `UShouse.senate` `parseName` `subNonStandardNames`

Examples

```
if(!fda::CRAN()){
USsenate <- readUSsenate()
}
```

readUSstateAbbreviations *Read a list of abbreviations of states and territories of the United States*

Description

Read the list of abbreviations of states and territories of the United states from the relevant Wikipedia article

Usage

```
readUSstateAbbreviations(url.=
"http://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations",
      clean=TRUE, Names=c('Name', 'Status', 'ISO', 'ANSI.letters',
          'ANSI.digits', 'USPS', 'USCG', 'Old.GPO', 'AP', 'Other') )
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired list
clean	logical: If TRUE, clean the data using subNonStandardCharacters and strsplit(x, "\[")
Names	names for the columns of the data matrix read; ignored with a warning if the lengths do not match

Details

Wrapper for readHTMLTable.

NOTE: readHTMLTable(url) returns a list of length 7, only one of which is the table we want. Moreover, that table contains some duplicates, which are removed by readUSstateAbbreviations. For example, 'NB' is an "Obsolete postal code" for Nebraska. If you need this, please consult the Wikipedia article.

Value

readUSstateAbbreviatons returns a data.frame from the table in the Wikipedia article on "List of U.S. state abbreviations".

Author(s)

Spencer Graves

See Also

getURL readHTMLTable make.names USstateAbbreviations

Examples

```
if(!fda::CRAN()){
abbreviations <- readUSstateAbbreviations()
}
```

---

recode2 *bivariate recode*

Description

Recode x1 and x2 per the lexical codes table.

Usage

```
recode2(x1, x2, codes)
```

Arguments

x1, x2	vectors of the same length assuming a discrete number of levels
codes	a 2-dimensional matrix indexed by the levels of x1 and x2. If dimnames(codes) are not provided, they are assumed to unique(x1) (or unique(x2)).

Details

1. If length(x1) != length(x2), complain.
2. if(is.logical(x1)) l1 <- c(FALSE, TRUE) else l1 <- unque(x1); ditto for x2.
3. If(missing(codes)) codes <- outer(unique(x1), unique(x2))
4. if(is.null(dim(codes))) dim(codes) <- c(length(unique(x1)), length(unique(x2)))
5. If is.null(rownames(codes)), set as follows: If nrow(codes) == length(unique(x1)), rownames(codes) <- unique(x1). Else, if nrow(codes) = max(x1), set rownames(codes) <- seq(1, max(x1). Else throw an error. Ditto for colnames, ncol, and x2.
6. codes[x1, x2]

Value

a vector of the same length as x1 and x2.

Author(s)

Spencer Graves

See Also

dim rownames link{colnames}

Examples

```
contrib <- c(-1, 0, 0, 1)
contrib0 <- c(FALSE, FALSE, TRUE, FALSE)

contribCodes <- recode2(contrib>0, contrib0,
   c('returned', 'received', '0', 'ERR') )

cC <- c('returned', 'returned', '0', 'received')

all.equal(contribCodes, cC)
```

`strsplit1` *Split the first field*

Description

Split the first field from `x`, identified as all the characters preceeding the first unquoted occurrence of `split`.

Usage

```
strsplit1(x, split=',', Quote='"', ...)
```

Arguments

`x`	a character vector to be split
`split`	the split character
`Quote`	a quote character: Occurrences of `split` between pairs of `Quote` are ignored.
`...`	optional arguments for grep

Details

This function was written to help parse data from the US Department of Health and Human Services on cyber-security breaches affecting 500 or more individuals. As of 2014-06-03 the csv version of these data included commas in quotes that are not `sep` characters. this function was written to split the fields one at a time to allow manual processing to make it easier to correct parsing errors.

Algorithm:

1. spl1 <- regexpr(split, x, ...)

2. Qt1 <- regexpr(Quote, x, ...)

3. For any (Qt1<spl1), look for Qt2 <- regexpr(Quote, substring(x, Qt1+1)), then look for spl1 <- regexpr(split, substring(x, Qt1+Qt2+1))

4. out <- list(substr(x, 1, spl1-1), substr(x, spl1+1))

Value

A list of length 2: The first component of the list contains the character strings found before the first unquoted occurrence of `split`. The second component contains the character strings remaining after the characters up to the identified `split` are removed.

Author(s)

Spencer Graves

See Also

`strsplit` `substring` `grep`

Examples

```
chars2split <- c(qs00='abcdefg', qs01='abc,def',
   qs10a='"abcdefg', qs10b='abc"defg',
   qs1.1='"abc,def', qs20='"abc" def',
   qs2.1='"ab,c" def', qs21='"abc", def', qs22.1='"a,b",c')

split <- strsplit1(chars2split)

# answer
split. <- list(c(qs00='abcdefg', qs01='abc', qs10a='"abcdefg',
   qs10b='abc"defg', qs1.1='"abc,def', qs20='"abc" def',
   qs2.1='"ab,c" def', qs21='"abc"', qs22.1='"a,b"'),
               c(qs00='', qs01='def', qs10a='',
   qs10b='', qs1.1='', qs20='', qs2.1='',
   qs21=' def', qs22.1='c') )

all.equal(split, split.)
```

---

subNonStandardCharacters

sub nonstandard characters with replacement

Description

Find the first and last character not in `standardCharacters` and replace all between them with `replacement`. For example, a string like "Ruben" where "e" carries and accent and is mangled by some software would become something like "Rub_n" using the default values for `standardCharacters` and `replacement`.

Usage

```
subNonStandardCharacters(x,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   replacement='_',
   gsubList=list(list(pattern='\\\\\\\\\|\\\\',
      replacement='\"')),
   ... )
```

Arguments

x	character vector in which it is desired to find the first and last character not in standardCharacters and replace that substring by replacement.
standardCharacters	a character vector of acceptable characters to keep.
replacement	a character to replace the subtring starting and ending with characters not in standardCharacters.
gsubList	list of lists of pattern and replacement arguments to be called in succession before looking for nonStandardCharacters
...	optional arguments passed to strsplit

Details

1. for(il in 1:length(gsubList))x <- gsub(gsubList[[il]][["pattern"]], gsublist[[il]][['replacement']], x)

2. nx <- length(x)

3. x. <- strsplit(x, "", ...)

4. for(ix in 1:nx) find the first and last standardCharacters in x.[ix] and substitute replacement for everything in between.

Value

a character vector with everthing between the first and last character not in standardCharacters replaced by replacement.

Author(s)

Spencer Graves

See Also

sub, strsplit, grepNonStandardCharacters, subNonStandardNames encoded_text_to_latex subNonStandardNames

Examples

```
# Consider Names = Ruben, Avila and Jose, where "e" and "A" in
#    these examples carry an accent.  With the default values
#    for standardCharacters and replacement, these would become
#    Rub_en, _vila, and Jos_.
#    (The standard checks for R packages complains about
#    non-standard characters, so none are included here.)
#
Names <- c('Ra`l', 'Ra`', '`l', 'Torres, Raul',
           "Robert C. \\Bobby\\\\")
#  confusion in character sets can create
#  names like Names[2]
Name2 <- subNonStandardCharacters(Names)

Name2. <- c('Ra_l', 'Ra_', '_l', Names[4],
            'Robert C. "Bobby"')


all.equal(Name2, Name2.)
```

---

subNonStandardNames *sub for nonstandard names*

Description

sub(nonEnglishData["nonEnglish"], nonEnglishData["English"], x)

Usage

```
subNonStandardNames(x,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   replacement='_',
   gsubList=list(list(pattern='\\\\\\\\\|\\\\',
      replacement='\"')),
   removeSecondLine=TRUE,
   nonStandardNames=Ecdat::nonEnglishNames, ...)
```

Arguments

x	character vector in which it is desired replace `nonStandardNames[, 1]` in `subNonStandardCharacters(x, ...)` with the corresponding element of `nonStandardNames[, 2]`.
standardCharacters, replacement, gsubList, ...	arguments passed to `subNonStandardCharacters`
removeSecondLine	logical: If TRUE, delete anything following "\n" and return it as an attribute "secondLine"

nonStandardNames
: data.frame or character matrix with two columns: Replace any substring of x matching nonStandardNames[, 1] with the corresponding elemeng of nonStandardNames[, 2]

Details

1. removeSecondLine
2. x. <- subNonStandardCharacters(x, standardCharacters, replacement, ...)
3. Loop over all rows of nonStandardNames substituting anything matching nonEnglishData[i, 1] with nonEnglishData[i, 2].
4. Eliminate leading and trailing blanks.

Value

a character vector with all nonStandardCharacters replaced first by replacement and then by the second column of nonStandardNames for any that match the first column.

Author(s)

Spencer Graves

See Also

sub nonEnglishNames subNonStandardCharacters

Examples

```
Names <- c('Raul', 'Ra`l', 'Torres,Raul', 'Torres, Raul',
           "Robert C. \\Bobby\\\\", 'Ed  \n --Vacancy')
#  confusion in character sets can create
#  names like Names[2]

library(Ecdat)

Name2 <- subNonStandardNames(Names)
Name2

Name2. <- c('Raul', 'Raul', Names[3:4],
            'Robert C. "Bobby"', 'Ed')
attr(Name2., 'secondLine') <- c(rep(NA, 5), ' --Vacancy')
Name2.


all.equal(Name2, Name2.)
```

testURLs *Test URLs for intermittent download problems*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

`try(getURL(...))` to read each element of `urls`. After each try, write a row to `file.` indicating which of `urls` was tested, the test time in seconds, and any error message. Repeat any failures up to `maxFail` times. After testing each element of `urls` once, repeat `n` times.

If(ping), preceed each test with "ping url[i]". NOTE: Some Internet Service Providers seem to block some attepts to use "ping" or return fraudulet replies to "ping". It is included in the code, because it seemed like an obvious test. However, it is not executed by default because the results do not necessarily reflect what people might expect from "ping".

Return a list of the last successful version read if any from each element of `urls` with two attributes: (1) "urls" containing the `urls` argument. (2) "testResults" being an object of class `c('testURLs', 'data.frame')` of the test results written to `file.`.

This function was written to diagnose a download problem with a particular Internet Service Provider (ISP). For other tools for testing an ISP, see measurementlab.net or the "Test your ISP" software discussion by the Electronic Frontier Foundation at the URL mentioned in references below.

Usage

```
testURLs(urls=c(
 wiki="http://en.wikipedia.org",
 wiki.PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov",
 house.reps="http://house.gov/representatives"),
         file.='testURLresults.csv',
         n=10, maxFail=10, warn=-1, tzone='GMT', ping=FALSE, ...)
```

Arguments

`urls`	a character vector assumed to be universal resource locators to pass to `getURL` for testing. The default was selected to provide a 2 x 2 experiment with two different web sites (en.wikipedia.org and house.gov) vs. the landing page and a subordinate page for each site.
`file.`	Name of a CSV file to which to write the results. If the file already exists, new results are appended to it.
`n`	number of times to repeat the cycle testing each member of `urls`.
`maxFail`	max tests for a continually failing URL. This is designed to make it relatively easy to determine determine dependencies between failures. If the failure rate is constant, the number of consecutive failures will follow a Poisson distribution. Otherwise, it may be possible to evaluate various effects using, e.g., state space

techniques for non-normal time series. This could include daily and weekly cycles possibly with holiday effects and trends as well as drifts suggesting abnormal drifts in web traffic congestion.

`warn` `warn` argument to pass to `Ping`.

`tzone` Time zone for `Time`. Defaults to GMT (UTC). tzone=NULL will use the current locale.

`ping` logical: TRUE to include `Ping`, FALSE otherwise.

`...` optional arguments for `Ping`.

Details

for(i in 1:n):

1. pingi <- Ping(urls[i], ...)

2. The time for each call to `getURL` is computed by computing `start.time <- proc.time()` before calling `try(getURL(.))`, then computing the following after:

```
elapsed.time <- max(proc.time() - start.time, na.rm=TRUE)
```

After each of the `urls` is tested, a summary of the results is appended to `file.`. This includes the pingi[['stats']], `elapsed.time` and the error message if the download failed.

The Electronic Frontier Foundation provides a table of existing software to "Test your ISP"; see the references below. This table includes a column noting whether the software is "active" (sending test traffic) or "passive" (observing the way the network treats natural traffic). The current `testURLs` function is "active", because it asks for a copy of the code at the indicated URL.

Value

an object of class `testURLs`, which in this case is a list of the last successful result returned by `getURL` for each element of `urls` with the following attributes:

`urls` the `urls` argument used for this call

`testURLresults` an object of class `c('testURLs', 'data.frame')` of the data written to `file.`. This has the following columns:

- Time `date()` for the time a particular test started
- URL the name in `urls` of the URL tested
- ping statistics several columns with the `count` and `stats` returned by `Ping`.
- readTime time in seconds for the attempt to read the URL (getURL(urls[j])) to complete.
- error character: " if the read attempt was successful; the error message if not.

Author(s)

Spencer Graves

References

measurementlab.net "Test your ISP" software discussion by the Electronic Frontier Foundation "active" (sending test traffic) or "passive" (observing the way the network treats natural traffic).

See Also

`try` `getURL` `Ping`

Examples

```
# Test only 2 web sites, not the default 4,
# and test only twice, not the default 10 times:
tst <- testURLs(c(
 PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov/representatives"),
    n=2, maxFail=2)


(class(tst) == 'testURLs') &&
all(names(tst) == c('PVI', 'house')) &&
all(names(attributes(tst)) ==
    c('names', 'urls', 'testURLresults', 'class'))
```

---

trimImage *Trim zero rows or columns from an object of class* Image.

Description

Identify rows or columns of a matrix or 3-dimensional array that are all 0 and remove them.

Usage

```
trimImage(x, max2trim=.Machine$double.eps, na.rm=TRUE,
          returnIndices2Keep=FALSE, ...)
```

Arguments

x	a numeric matrix or 3-dimensional array or an object with subscripting defined so it acts like such.
max2trim	a single number indicating the max absolute numeric value to trim.
na.rm	logical: If TRUE, NAs will be ignored in determining the max absolute value for the row. If a row or column is all NA, it will be treated as all 0 in deciding whether to trim. If FALSE, any row or column containing an NA will be retained.

`returnIndices2Keep`
: if TRUE, return a list with 2 integer vectors giving row and column indices to use in selecting the desired subset of `x`. This allows an array `y` to be trimmed to match `x`.

If FALSE, return the desired trimmed version of `x`.

If this is a list with two two integer vectors, use them to trim `x`.

`...`
: Optional arguments; not currently used.

Details

1. Check arguments: 2 <= length(dim(x)) <= 3? is.logical(na.rm)? returnIndices2Keep = logical or list of 2 integer vectors, all the same sign, not exceeding dim(x)?

2. if(is.list(returnIndices2Keep)) check that returnIndices2Keep is a list with 2 integer vectors, all the same sign, not exceeding dim(x). If yes, return x appropriately subsetted.

3. if(!is.logical(returnIndices2Keep)) throw an error message.

4. Compute indices2Keep.

5. If(returnIndices2Keep) return (indices2Keep) else return x appropriately susetted.

Value

if(returnIndices2Keep==TRUE) return a list with 2 integer vectors to use as subscripts in trimming objects like x.

Otherwise, return an object like x appropriately trimmed.

Author(s)

Spencer Graves

See Also

`trim` trims raster images, similar to `trimImage`.

`trim` trims leading and trailing spaces from character strings and factors. Similar `trim` functions exist in other packages but without obvious, explicit consideration of `factors`.

Examples

```
##
## 1.  trim a simple matrix
##
tst1 <- matrix(.Machine$double.eps, 3, 3,
          dimnames=list(letters[1:3], LETTERS[1:3]))
tst1[2,2] <- 1
tst1t <- trimImage(tst1)

# check
tst1. <- matrix(1, 1, 1,
          dimnames=list(letters[2], LETTERS[2]))
```

```
all.equal(tst1t, tst1.)


##
## 2.  returnIndices2Keep
##
tst2i <- trimImage(tst1, returnIndices2Keep=TRUE)
tst2a <- trimImage(tst1, returnIndices2Keep=tst2i)

tst2i. <- list(index1=2, index2=2)


# check

all.equal(tst2i, tst2i.)


all.equal(tst2a, tst1.)


##
## 3.  trim 0's only
##
tst3 <- array(0, dim=3:5)
tst3[2, 2:3, ] <- 0.5*.Machine$double.eps
tst3[3,,] <- 1

tst3t <- trimImage(tst3, 0)

# check
tst3t. <- tst3[2:3,, ]

# check

all.equal(tst3t, tst3t.)


##
## 4.  trim NAs
##
tst4 <- tst1
tst4[1,1] <- NA
tst4[3,] <- NA

tst4t <- trimImage(tst4)
# tst4o == tst4
tst4o <- trimImage(tst4, na.rm=FALSE)

# check

all.equal(tst4t, tst1[2, 2, drop=FALSE])
```

```
all.equal(tst4o, tst4)


##
## 5.  trim all
##
tst4a <- trimImage(tst1, 1)

tst4a. <- matrix(0,0,0,
     dimnames=list(NULL, NULL))


all.equal(tst4a, tst4a.)
```

`truncdist` *Truncated distribution*

Description

The cumulative distribution function for a truncated distribution is `0` for `x <= truncmin`, `1` for `truncmax < x`, and in between is as follows:

`(pdist(x, ...) - pdist(truncmin, ...)) / (pdist(truncmax, ...) - pdist(truncmin, ...))`

The density, quantile, and random number generation functions are similarly defined from this.

Usage

```
dtruncdist(x, ..., dist='norm', truncmin=-Inf, truncmax=Inf)
ptruncdist(q, ..., dist='norm', truncmin=-Inf, truncmax=Inf)
qtruncdist(p, ..., dist='norm', truncmin=-Inf, truncmax=Inf)
rtruncdist(n, ..., dist='norm', truncmin=-Inf, truncmax=Inf)
```

Arguments

`x, q`	numeric vector of quantiles
`p`	numeric vector of probabilities
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`...`	other arguments to be passed to the corresponding function for the indicated `dist`
`dist`	Standard `R` name for the family of functions for the desired distribution. By default, this is "norm", so the corresponding function for `dtruncdist` is `dnorm`, the corresponding function for `ptrunctist` is `pnorm`, etc.
`truncmin, truncmax`	lower and upper truncation points, respectively.

Details

NOTE: This is different from "censoring", where it's known that an observation lies between certain limits; it's just not known exactly where it lies between those limits.

For a truncated distrubion, observations below `truncmin` and `truncmax` are not observed at all. Thus, it's not known how many observations like outside the given range, `truncmin` to `truncmax`, if any.

1. Setup

dots <- list(...)

2. For `dtruncdist`, return 0 for all `x` outside `truncmin` and `truncmax`. For all others, compute as follows:

dots$x <- truncmin ddist <- paste0('d', dist) pdist <- paste0('p', dist) p.min <- do.call(pdist, dots) dots$x <- truncmax p.max <- do.call(pdist, dots) dots$x <- x dx <- do.call(ddist, dots)

return(dx / (p.max-p.min))

NOTE: Adjustments must be made if 'log' appears in `names(dots)`

3. The computations for `ptruncdist` are similar.

4. The computations for `qtruncdist` are complementary.

5. For `rtruncdist`, use `qtruncdist(runif(n), ...)`.

Value

`dtruncdist` gives the density, `ptruncdist` gives the distribution function, `qtruncdist` gives the quantile function, and `rtruncdist` generates random deviates.

The length of the result is determined by `n` for `rtruncdist` and is the maximum of the lengths of the numerical arguments for the other functions.

Author(s)

Spencer Graves

See Also

`Distributions` `Normal`

Examples

```
##
## 1.  dtruncdist
##
#  1.1.  Normal
dx <- dtruncdist(1:4)

# check

all.equal(dx, dnorm(1:4))
```

```
#  1.2.  Truncated normal between 0 and 1
dx01 <- dtruncdist(seq(-1, 2, .5), truncmin=0, truncmax=1)

# check
dx01. <- c(0, 0, 0, dnorm(c(.5, 1))/(pnorm(1)-pnorm(0)),
           0, 0)

all.equal(dx01, dx01.)


#  1.3.  lognormal meanlog=log(100), sdlog = 2, truncmin=500
x10 <- 10^(0:9)
dx10 <- dtruncdist(x10, log(100), 2, dist='lnorm',
                   truncmin=500)

# check
dx10. <- (dtruncdist(log(x10), log(100), 2,
                    truncmin=log(500)) / x10)

all.equal(dx10, dx10.)


#  1.4.  log density of the previous example
dx10log <- dtruncdist(x10, log(100), 2, log=TRUE,
                  dist='lnorm', truncmin=500)

all.equal(dx10log, log(dx10))


##
## 2.  ptruncdist
##
#  2.1.  Normal
px <- ptruncdist(1:4)

# check

all.equal(px, pnorm(1:4))


#  2.2.  Truncated normal between 0 and 1
px01 <- ptruncdist(seq(-1, 2, .5), truncmin=0, truncmax=1)

# check
px01. <- c(0, 0, (pnorm(c(0, .5, 1)) - pnorm(0))
                    /(pnorm(1)-pnorm(0)), 1, 1)

all.equal(px01, px01.)


#  2.3.  lognormal meanlog=log(100), sdlog = 2, truncmin=500
x10 <- 10^(0:9)
px10 <- ptruncdist(x10, log(100), 2, dist='lnorm',
```

```
                  truncmin=500)

# check
px10. <- (ptruncdist(log(x10), log(100), 2,
                     truncmin=log(500)))

all.equal(px10, px10.)


#  2.4.  log of the previous probabilities
px10log <- ptruncdist(x10, log(100), 2, log=TRUE,
                  dist='lnorm', truncmin=500)

all.equal(px10log, log(px10))


##
## 3.  qtruncdist
##
#  3.1.  Normal
qx <- qtruncdist(seq(0, 1, .2))

# check

all.equal(qx, qnorm(seq(0, 1, .2)))


#  3.2.  Normal truncated outside (0, 1)
qx01 <- qtruncdist(seq(0, 1, .2), truncmin=0, truncmax=1)

# check
pxmin <- pnorm(0)
pxmax <- pnorm(1)
unp <- (pxmin + seq(0, 1, .2)*(pxmax-pxmin))
qx01. <- qnorm(unp)

all.equal(qx01, qx01.)


#  3.3.  lognormal meanlog=log(100), sdlog=2, truncmin=500
qlx10 <- qtruncdist(seq(0, 1, .2), log(100), 2,
                   dist='lnorm', truncmin=500)

# check
plxmin <- plnorm(500, log(100), 2)
unp. <- (plxmin + seq(0, 1, .2)*(1-plxmin))

qlx10. <- qlnorm(unp., log(100), 2)

all.equal(qlx10, qlx10.)


#  3.4.  previous example with log probabilities
```

```
qlx10l <- qtruncdist(log(seq(0, 1, .2)), log(100), 2,
                  log.p=TRUE, dist='lnorm', truncmin=500)

# check

all.equal(qlx10, qlx10l)


##
## 4.  rtruncdist
##
#  4.1.  Normal
set.seed(1)
rx <- rtruncdist(9)

# check
set.seed(1)

all.equal(rx[1], rnorm(1))

# Only the first observation matches;  check that.

#  4.2.  Normal truncated outside (0, 1)
set.seed(1)
rx01 <- rtruncdist(9, truncmin=0, truncmax=1)

# check
pxmin <- pnorm(0)
pxmax <- pnorm(1)
set.seed(1)
rnp <- (pxmin + runif(9)*(pxmax-pxmin))
rx01. <- qnorm(rnp)

all.equal(rx01, rx01.)


#  4.3.  lognormal meanlog=log(100), sdlog=2, truncmin=500
set.seed(1)
rlx10 <- rtruncdist(9, log(100), 2,
                  dist='lnorm', truncmin=500)

# check
plxmin <- plnorm(500, log(100), 2)
set.seed(1)
rnp. <- (plxmin + runif(9)*(1-plxmin))

rlx10. <- qlnorm(rnp., log(100), 2)

all.equal(rlx10, rlx10.)
```

UShouse.senate *Create a list of members of the US House and Senate*

Description

Combine the output of readUShouse and readUSsenate.

Usage

```
UShouse.senate(house=readUShouse(), senate=readUSsenate())
```

Arguments

house, senate	data.frames as returned by the functions readUShouse and readUSsenate, respectively.

Details

Convert the two into a common format and rbind.

Value

a data.frame with the following columns:

Office	A factor identifying "House" vs. "Senate", indicating whether the person is in the US House or Senate
state	A factor identifying the state using the USPS 2-letter state code (all caps)
district	"0" or "At-Large" for members of the US House representing an entire state or integers in character format indicating the district. For the Senate, this contains the "class", which codes the year of the next election for that seat is an integer multiple of 6 years after 2012, 2008, or 2010 for class "1", "2", or "3", respectively.
Party	a factor identifying the party affiliation of each representative, e.g., 'Democratic', 'Republican', 'Democratic-Farmer-Labor', 'Independent'.
surname	family name
givenname	first name with possibly a middle name, nickname, and suffix (e.g., Jr., III).

Author(s)

Spencer Graves

See Also

readUShouse readUSsenate

Examples

```
if(!fda::CRAN()){
house <- readUShouse()

USreps <- UShouse.senate(house)
}
```

`USsenateClass` *Election Class given state and surname of a US Senator*

Description

For all indiviuals in `x` with `houseSenate` == "Senate", look up their `state` and `surname` in the reference table `senate` and return their `Class`. For indivuals not found in `senate`, return `x[[district]]`.

Senate classes 1, 2 and 3 have their normal elections in 6-year cycles including 2000, 2002, and 2004 (or 2012, 2008, and 2010), respectively. When vacancies occur out of cycle, the vacancy is first filled with appointment by the governor of the state, and an election to fill that seat occurs in the next even-numbered year; the class of that seat does not change.

For example, South Carolina Senator Jim DeMint resigned effective January 1, 2013. South Carolina Governor Nikki Haley appointed Tim Scott to serve until a special election in 2014. This is a Class 3 seat, which means that another election for that seat will occur in 2016.

Usage

```
USsenateClass(x, senate=readUSsenate(),
   Office='Office', state='state',
   surname='surname', district='district', senatePattern='^Senate')
```

Arguments

`x`	`data.frame` with character or factor columns `Office`, `state`, `surname`, and `district`.
`senate`	`data.frame` as returned by `readUSsenate`.
`Office`	name of a character or factor variable `x` in which the members of the US Senate can be identifed by `grep(senatePattern, x[, Office])`.
`state`	Standard 2-letter abbreviation for the state of the US
`surname`	the name of a column of `x` containing the surname
`district`	name of a column of `x` containing the number of the district in the US House. For states with only one representataive, this may be 0.
`senatePattern`	a regular expression for identifying the senators from `x[, Office]`.

Details

The current algorithm may fail if both senators in a `state` have the same `surname`.

Value

a data.frame with one row for each row of x and the following columns:

incumbent — logical vector: NA if Office == 'house'. If Office == 'senate', then TRUE if state:surname found in senate and FALSE otherwise.

District — a character vector containing the desired Class for all US Senators found in senate or a guess at the Class for non-incumbents. For members of the House, this returned the previous content of x[[District]].

NOTES:

1. Incumbents can be missed if the spelling of the surname is different between x and senate. This can occur with, for example, Spanish surnames containing an accent.

2. If one but not two incumbents is found, others are currently assigned to the class of an incumbent not found. This could be a mistake, because the person could be a previous incumbent or could have lost to the incumbent in the last election.

Author(s)

Spencer Graves

See Also

readUSsenate

Examples

```
tst <- data.frame(Office=factor(c("House", "Senate", "Senate", 'Senate')),
                  state=factor(c('SC', 'SC', 'SC', 'NY')),
                  surname=c("Jones", "DeMint", "Graham", 'Smith'),
                  district=c("9", NA, NA, NA),
                  stringsAsFactors=FALSE)
if(!fda::CRAN()){
tst. <- USsenateClass(tst)

chk <- data.frame(incumbent=c(NA, FALSE, TRUE, FALSE),
                  district=c("9", "3", "2", "1 or 3"),
                  stringsAsFactors=FALSE)

all.equal(tst., chk)


##
## test with names different from the default
##
tst2 <- tst
names(tst2) <- letters[1:4]
tst2. <- USsenateClass(tst2, Office='a',
           state='b', surname='c', district='d')
```

```
all.equal(tst., tst2.)

}
```

---

whichAeqB *Index of a single match*

Description

Return which(A %in% B) if it has length 1; give an error message otherwise.

Usage

```
whichAeqB(A, B, errNoMatch='no match',
                      err2Match='more than one match')
```

Arguments

A	A vector which may have a single match in B.
B	A vector of possible matches for A.
errNoMatch	a character string: error message if no match found.
err2Match	a character string: error message if multiple matches found.

Value

a single integer giving the index of the match in A.

Author(s)

Spencer Graves

See Also

interpPairs

Examples

```
a2b <- whichAeqB(letters, 'b')


all.equal(a2b, 2)
```

Index