

Package ‘Ecfun’

February 25, 2019

Version 0.2-0

Date 2019-02-20

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,
knitr, rmarkdown, invgamma, ipumsr

VignetteBuilder knitr

Imports fda, gdata, RCurl, XML, tis, jpeg, MASS, TeachingDemos,
stringi, methods, xml2

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

LazyData true

License GPL (>= 2)

URL <https://www.r-project.org>

Repository CRAN

Repository/R-Forge/Project ecdat

Repository/R-Forge/Revision 409

Repository/R-Forge/DateTimeStamp 2019-02-22 19:13:31

Date/Publication 2019-02-25 21:10:03 UTC

NeedsCompilation no

R topics documented:

Arrows	3
as.Date1970	4
asNumericDF	5
BoxCox	8

camelParse	14
canbeNumeric	15
checkNames	16
classIndex	18
compareLengths	20
confint.var	22
countByYear	24
countsByYear	26
createMessage	27
createX2matchY	29
Date3to1	30
dateCols	31
Dates3to1	33
financialCrisisFiles	34
getElement2	36
grepNonStandardCharacters	38
Interp	40
interpChar	45
interpPairs	49
logVarCor	57
match.data.frame	59
matchName	61
matchQuote	64
mergeUSHouse.senate	66
mergeVote	68
missing0	70
nchar0	71
Newdata	72
parseCommas	75
parseDollars	76
parseName	77
Ping	80
pmatch2	82
qqnorm2	84
qqnorm2s	88
qqnorm2t	91
rasterImageAdj	93
read.testURLs	95
read.transpose	96
readCookPVI	98
readDates3to1	100
readFinancialCrisisFiles	101
readNIPA	103
readUSHouse	105
readUSSenate	107
readUSstateAbbreviations	108
recode2	109
rgrep	110

sign	112
strsplit1	113
subNonStandardCharacters	114
subNonStandardNames	116
testURLs	119
trimImage	122
truncdist	125
UShouse.senate	130
USsenateClass	131
whichAeqB	133

Index	135
--------------	------------

Arrows	<i>Draw arrows between pairs of points.</i>
--------	---

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 origin 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

For asNumericChar, delete leading blanks and a leading dollar sign plus commas (thousand separators) and drop information after a blank (other than leading blanks), then coerce to numeric or to factors, Dates, or POSIXct as desired.

For a [data.frame](#), apply asNumericChar to all columns and drop columns except those in keep, ignore, factors, Dates, POSIX and MSdates.

Then order the rows by the orderBy column. Some Excel imports include commas as thousand separators; this replaces any commas with char(0), "", before trying to convert to numeric.

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.

For example Table F-1 from the US Census Bureau needs all three of these numeric conversion 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, leadingChar='^\\$',
              suppressChar=',', pctChar='%$')
asNumericDF(x, keep=function(x)any(!is.na(x)),
            orderBy=NA, ignore=NULL, factors=NULL,
            Dates=NULL, POSIX=NULL, MSdates=NULL,
            format., leadingChar='^\\$',
            suppressChar=',', pctChar='%$')

```

Arguments

x	For asNumericChar, this is a character vector to be converted to numeric after <code>gsub(',', '', x)</code> . For asNumericDF, this is a <code>data.frame</code> with all character columns to be converted to numerics.
keep	something to indicate which columns to keep, in addition to columns specified in <code>ignore</code> , <code>factors</code> , <code>Dates</code> , and <code>POSIX</code> .
orderBy	Which columns to order the rows of <code>x[, keep]</code> by. Default is to keep the input order.
ignore	vector identifying columns of <code>x</code> to ignore, i.e., to keep and not attempt to convert to another data type.
factors	vector indicating columns of <code>x</code> to convert to <code>factor</code>
Dates	vector indicating columns of <code>x</code> to convert using <code>as.Date(, format)</code> .
POSIX	vector indicating columns of <code>x</code> to convert using <code>as.POSIXct(, format)</code> .
format.	Character vector of length 1 to pass as argument <code>format</code> to <code>as.Date</code> and / or <code>as.POSIXct</code> for conversion from <code>character</code> . For <code>Dates</code> , <code>as.Date</code> is first tried with <code>format = '%Y-%m-%d'</code> , then with <code>'%Y/%m/%d'</code> , <code>'%m-%d-%Y'</code> , and <code>'%m/%d/%Y'</code> . The conversion with the fewest NAs is kept. If two match for numbers of NAs, the one with the minimum absolute deviations from <code>as.Date1970(0)</code> is used.
MSdates	The names or numbers identifying columns of <code>x</code> identifying dates as integer numbers of days since 1899-12-31. In Microsoft Excel, dates are stored in that format. Tests on 2016-08-10 revealed that <code>read.xls</code> did not read those dates properly (using <code>gdata</code> version 2.17.0), while <code>read.xlsx</code> read them as integers using <code>openxlsx</code> 3.0.0. This capability will convert those integers to <code>Dates</code> using <code>as.Date(, origin = as.Date('1899-12-31'))</code> .
leadingChar	A regular expression passed to <code>grep</code> and <code>sub</code> to replace something like an initial dollar sign with <code>character(0)</code> .
suppressChar	a regular expression passed to <code>gsub</code> to replace all occurrences of something like <code>"</code> , (a thousands separator in the U.S.) with <code>character(0)</code> .
pctChar	A regular expression passed to <code>grep</code> to identify percent columns. <code>pctChar</code> is then passed to <code>sub</code> to replace <code>pctChar</code> with <code>character(0)</code> , and the converted numbers are then divided by 100 to convert them to proportions.

Details

For asNumericChar:

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`

for asNumericDF:

1. Copy `x` to `X`.

2. Confirm that ignore, factors, Dates, and POSIX all refer to columns of x and do not overlap. [*** NOTE: as of 2016-07-21, these checks have only been implemented for ignore.]
3. Convert factors, Dates, and POSIX.
4. Apply asNumericChar to all columns not in ignore, factors, Dates, or POSIX.
5. Keep columns specified by keep.
6. return the result.

Value

a `data.frame`

Author(s)

Spencer Graves

References

"Add (sum) or subtract dates; Applies To: Excel 2013". Microsoft. (accessed 2016-08-11)

See Also

`scan` `gsub` `Quotes` `stripBlanks` `as.numeric`, `factor`, `as.Date`, `as.POSIXct` `read.xlsx`

Examples

```
##
## 1. an example
##
xDate <- as.Date('1970-01-01')+c(0, 365)
xPOSIX <- as.POSIXct(xDate)+c(1, 99)
xMSdate <- as.Date(1, as.Date('1899-12-31'))+1:2
fakeF1 <- data.frame(yr=c('1948', '1947 (1)'),
  q1=c(' 1,234 ', ''), duh=rep(NA, 2),
  dol=c('$1,234', ''), pct=c('1%', '2%'),
  xDate=as.character(xDate, format='%m-%d-%Y'),
  xPOSIX=as.character(xPOSIX,
    format='%m-%d-%Y %H:%M:%S'),
  xMSdate=2:3, junk=c('this is', 'junk'))

# This converts the last 3 columns to NAs and drops them:

str(nF1.1 <- asNumericChar(fakeF1$yr))
str(nF1.2 <- asNumericChar(fakeF1$q1))
str(nF1.3 <- asNumericChar(fakeF1$duh))

nF1 <- asNumericDF(fakeF1)
nF2 <- asNumericDF(fakeF1, Dates=6, MSdate='xMSdate',
  ignore=c('junk', 'xPOSIX'), format.='%m-%d-%Y')
nF3 <- asNumericDF(nF2, POSIX='xPOSIX',
  ignore=c(5,7:8), format.='%m-%d-%Y %H:%M:%S')
```

```

# check
nF1. <- data.frame(yr=asNumericChar(fakeF1$yr),
                  q1=asNumericChar(fakeF1$q1),
                  dol=asNumericChar(fakeF1$dol),
                  pct=c(.01, .02),
                  xMSdate=2:3)

nF1c <- data.frame(yr=1948:1947, q1=c(1234, NA),
                  dol=c(1234, NA), pct=c(.01, .02),
                  xMSdate=2:3)

all.equal(nF1, nF1.)

all.equal(nF1., nF1c)

nF3c <- data.frame(yr=1948:1947, q1=c(1234, NA),
                  dol=c(1234, NA), pct=c(.01, .02),
                  xDate=xDate, xPOSIX=xPOSIX,
                  xMSdate=xMSdate, junk=fakeF1$junk)

all.equal(nF3, nF3c)

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

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

```

 BoxCox

Box-Cox power transformation and its inverse

Description

Box and Cox (1964) considered the following family of transformations indexed by λ :

$$w = (y^\lambda - 1) / \lambda$$

$$= \expm1(\lambda \log(y)) / \lambda,$$

with the $\lambda=0$ case defined as $\log(y)$ to make w continuous in λ for constant y .

They estimate λ assuming w follows a normal distribution. This raises a theoretical problem in that y must be positive, which means that w must follow a truncated normal distribution conditioned on $\lambda * w > (-1)$.

Bickel and Doksum (1981) removed the restriction to positive y , i.e., to $w > (-1/\lambda)$ by modifying the transformation as follows:

$$w = \begin{cases} (\text{sgn}(y) \cdot \text{abs}(y)^\lambda - 1) / \lambda & \text{if } \lambda \neq 0 \text{ and} \\ \text{sgn}(y) \cdot \log(\text{abs}(y)) & \text{if } \lambda = 0, \end{cases}$$

where $\text{sgn}(y) = 1$ if $y \geq 0$ and -1 otherwise.

NOTE: $\text{sgn}(y)$ is different from $\text{link}[\text{base}]\{\text{sign}\}(y)$, which is 0 for $y = 0$. A two-argument update to the sign function in the base package has been added to this Ecfun package, so `sign(y, 1) = sgn(y)`.

If $y < 0$, this transformation is discontinuous at $\lambda = 0$. To see this, we rewrite this as

$$\begin{aligned} w &= (\text{sgn}(y) \cdot \exp(\lambda \cdot \log(\text{abs}(y))) + (\text{sgn}(y) - 1)) / \lambda \\ &= \text{sgn}(y) \cdot (\log(\text{abs}(y)) + O(\lambda)) + (\text{sgn}(y) - 1) / \lambda, \end{aligned}$$

where $O(\lambda)$ indicates a term that is dominated by a constant times λ .

If $y < 0$, this latter term $(\text{sgn}(y) - 1) / \lambda = (-2) / \lambda$ and becomes Inf as $\lambda \rightarrow 0$.

In practice, we assume that $y > 0$, so this distinction has little practical value. However, the BoxCox function computes the Bickel-Doksum version.

Box and Cox further noted that proper estimation of λ should include the Jacobian of the transformation in the $\log(\text{likelihood})$. Doing this can be achieved by rescaling the transformation with the n th root of the Jacobian, which can be written as follows:

$$j(y, \lambda) = J(y, \lambda)^{1/n} = \text{GeometricMean}(y)^{(\lambda - 1)}.$$

With this the rescaled power transformation is as follows:

$$z = \begin{cases} (y^\lambda - 1) / (\lambda \cdot j(y, \lambda)) & \text{if } \lambda \neq 0 \\ \text{GeometricMean}(y) \cdot \log(y) & \text{if } \lambda = 0. \end{cases}$$

In addition to facilitating estimation of λ , rescaling has the advantage that the units of z are the same as the units of y .

The output has class 'BoxCox', which has attributes that allow the input to be recovered using `invBoxCox`. The default values of the arguments of `invBoxCox` are provided by the corresponding [attributes](#) of z .

Usage

```
BoxCox(y, lambda, rescale=TRUE, na.rm=rescale)
invBoxCox(z, lambda, sign.y, GeometricMean, rescale)
```

Arguments

<code>y</code>	a numeric vector for which the power transform is desired
<code>lambda</code>	A numeric vector of length 1 or 2. The first component is the power. If the second component is provided, y is replaced by $y + \text{lambda}[2]$.
<code>rescale</code>	logical or numeric. If logical: For <code>BoxCox</code> , this is <code>TRUE</code> to return the power transform with <code>rescale</code> , z , above, and <code>FALSE</code> to return the power transform without the n th root of the Jacobian, w , above. This defaults to <code>TRUE</code> , because this will give z the same units as y .

	For <code>invBoxCox</code> , this is TRUE if the input argument <code>z</code> is assumed to have been rescaled by the <code>nth</code> root of the Jacobian of the transformation. This defaults to a <code>rescale</code> attribute of <code>z</code> if present or to TRUE if absent. If numeric, it is assumed to be the geometric mean of another set of <code>y</code> values to use with new <code>y</code> 's.
<code>na.rm</code>	logical: TRUE to remove NAs from <code>y</code> before computing the geometric mean. FALSE to compute NA for the geometric mean if <code>any(is.na(y))</code> . NOTE: If <code>na.rm = FALSE</code> , the output will be all NA if <code>rescale = TRUE</code> . This could produce non useable answers in most cases. To avoid that, the default for <code>na.rm</code> is TRUE whenever <code>rescale = TRUE</code> . Conversely, applications using <code>na.rm = FALSE</code> will likely also want <code>rescale = FALSE</code> to avoid returning a non-answer in these cases. This explains the default <code>na.rm = rescale</code> .
<code>z</code>	a numeric vector or an object of class <code>BoxCox</code> for which the inverse Box-Cox transform is desired.
<code>sign.y</code>	an optional logical vector giving <code>sign(y-lambda[2])</code> of the data values that presumably generated <code>z</code> . Defaults to a <code>sign.y</code> attribute of <code>z</code> or to <code>rep(1, length(z))</code> if no such attribute is present.
<code>GeometricMean</code>	an optional numeric scalar giving the geometric mean of the data values that presumably generated <code>z</code> . Defaults to a <code>GeometricMean</code> attribute of <code>z</code> or to 1 if no such attribute is present.

Details

Box and Cox (1964) discussed

$$w(y, \lambda) = (y^\lambda - 1)/\lambda.$$

They noted that `w` is continuous in `lambda` with `w(y, lambda) = log(y)` if `lambda = 0` (by l'Hopital's rule).

They also discussed

$$z(y, \lambda) = (y^\lambda - 1)/(\lambda * g^{(\lambda-1)}),$$

where `g` = the geometric mean of `y`.

They noted that proper estimation of `lambda` should include the Jacobian of `w(y, lambda)` with the likelihood. They further showed that a naive normal likelihood using `z(y, lambda)` as the response without a Jacobian is equivalent to the normal likelihood using `w(y, lambda)` adjusted appropriately using the Jacobian. See Box and Cox (1964) or [the Wikipedia article on "Power transform"](#).

Bickel and Doksum (1981) suggested adding `sign(y)` to the transformation, as discussed above.

NUMERICAL ANALYSIS:

Consider the Bickel and Doksum version described above:

$$w <- (\text{sign}(y) * \text{abs}(y)^\lambda - 1) / \lambda$$

`if(any(y==0)), GeometricMean(y) = 0`. This creates a problem with the above math.

Let `ly = log(abs(y))`. Then with `la = lambda`,

$$w = \text{code}(\text{sign}(y) * \exp(\text{la} * \text{ly}) - 1) / \text{la}$$

$$= \text{sign}(y) * \text{ly} * (1 + (\text{la} * \text{ly} / 2) * (1 + (\text{la} * \text{ly} / 3) * (1 + (\text{la} * \text{ly} / 4) * (1 + 0 * (\text{la} * \text{ly})))))) + (\text{sign}(y) - 1) / \text{la}$$

For $y > 0$, the last term is zero. `boxcox` ignores cases with $y \leq 0$ and uses this formula (ignoring the final $O(\lambda^*y)$) whenever $\text{abs}(\lambda) \leq \text{eps} = 1/50$. That form is used here also.

For `invBoxCox` a complementary analysis is as follows:

$$\begin{aligned} \text{abs}(y + \text{lambda}[2]) &= \text{abs}(1 + \lambda * w)^{1/\lambda} \\ &= \exp(\log 1 p(\lambda * w) / \lambda) \text{ for } \text{abs}(\lambda * w) < 1 \\ &= w * (1 - \lambda * w^{(1/2)} - \lambda * w^{(1/3)} - \lambda * w^{(1/4)} - \dots) \end{aligned}$$

Value

`BoxCox` returns an object of class `BoxCox`, being a numeric vector of the same length as `y` with the following optional attributes:

- `lambda` the value of `lambda` used in the transformation
- `sign.y` `sign(y)` (or `sign(y - lambda[2])` if `lambda[2]` is provided) and if any of these quantities are negative. Otherwise, this is omitted and all are assumed to be positive.
- `rescale` logical: TRUE if `z(y, lambda)` is returned rescaled by $g^{(\lambda-1)}$ with g = the geometric mean of `y` and FALSE if `z(y, lambda)` is not so rescaled.
- `GeometricMean` If `rescale` is numeric, `attr(, 'GeometricMean') <- rescale`. Otherwise, `attr(, 'GeometricMean')` is the Geometric mean of $\text{abs}(y) = \exp(\text{mean}(\log(\text{abs}(y))))$ or of $\text{abs}(y + \text{lambda}[2])$ if $\text{length}(\text{lambda}) > 1$.

`invBoxCox` returns a numeric vector, reconstructing `y` from `BoxCox(y, ...)`.

Source

Bickel, Peter J., and Doksum, Kjell A. (1981) "An analysis of transformation revisited", *Journal of the American Statistical Association*, 76 (374): 296-311

Box, George E. P.; Cox, D. R. (1964). "An analysis of transformations", *Journal of the Royal Statistical Society, Series B* 26 (2): 211-252.

Box, George E. P.; Cox, D. R. (1982). "An analysis of transformations revisited, rebutted", *Journal of the American Statistical Association*, 77(377): 209-210.

References

[Wikipedia, "Power transform"](#)

See Also

[boxcox](#) in the MASS package

[quine](#) in the MASS package for data used in an example below.

[boxcox](#) and [boxcoxCensored](#) in the EnvStats package.

[boxcox.drc](#) in the drc package.

[boxCox](#) in the car package.

These other uses all wrap the Box-Cox transformation in something larger and do not give the transformation itself directly.

Examples

```

##
## 1. A simple example to check the two algorithms
##
Days <- 0:9
bc1 <- BoxCox(Days, c(0.01, 1))
# Taylor expansion used for obs 1:7; expm1 for 8:10

# check
GM <- exp(mean(log(abs(Days+1))))

bc0 <- (((Days+1)^0.01)-1)/0.01
bc1. <- (bc0 / (GM^(0.01-1)))
# log(Days+1) ranges from 0 to 4.4
# lambda = 0.01 will invoke both the obvious
# algorithm and the alternative assumed to be
# more accurate for (lambda(log(y)) < 0.02).
attr(bc1., 'lambda') <- c(0.01, 1)
attr(bc1., 'rescale') <- TRUE
attr(bc1., 'GeometricMean') <- GM
class(bc1.) <- 'BoxCox'

all.equal(bc1, bc1.)

##
## 2. another simple example with lambda=0
##
bc0.4 <- BoxCox(1:5, 0)
GM5 <- prod(1:5)^.2
bc0.4. <- log(1:5)*GM5
attr(bc0.4., 'lambda') <- 0
attr(bc0.4., 'rescale') <- TRUE
attr(bc0.4., 'GeometricMean') <- GM5
class(bc0.4.) <- 'BoxCox'

all.equal(bc0.4, bc0.4.)

##
## 3. The "boxcox" function in the MASS package
## computes a maximum likelihood estimate with
## BoxCox(Days+1, lambda=0.21)
## with a 95 percent confidence interval of
## approximately (0.08, 0.35)
##
bcDays1 <- BoxCox(MASS::quine$Days, c(0.21, 1))

# check
GeoMean <- exp(mean(log(abs(MASS::quine$Days+1))))

```

```
bcDays1. <- (((MASS::quine$Days+1)^0.21)-1) /
  (0.21*GeoMean^(0.21-1)))
# log(Days+1) ranges from 0 to 4.4
attr(bcDays1., 'lambda') <- c(0.21, 1)
attr(bcDays1., 'rescale') <- TRUE
attr(bcDays1., 'GeometricMean') <- GeoMean
class(bcDays1.) <- 'BoxCox'
```

```
all.equal(bcDays1, bcDays1.)
```

```
iDays <- invBoxCox(bcDays1)
```

```
all.equal(iDays, MASS::quine$Days)
```

```
##
## 4. Easily computed example
##
bc2 <- BoxCox(c(1, 4), 2)
```

```
# check
bc2. <- (c(1, 4)^2-1)/4
attr(bc2., 'lambda') <- 2
attr(bc2., 'rescale') <- TRUE
attr(bc2., 'GeometricMean') <- 2
class(bc2.) <- 'BoxCox'
```

```
all.equal(bc2, bc2.)
```

```
all.equal(invBoxCox(bc2), c(1, 4))
```

```
##
## 5. plot(BoxCox())
##
y0 <- seq(-2, 2, .1)
z2 <- BoxCox(y0, 2, rescale=FALSE)
plot(y0, z2)
```

```
# check
z2. <- (sign(y0)*y0^2-1)/2

attr(z2., 'lambda') <- 2
attr(z2., 'sign.y') <- sign(y0, 1)
attr(z2., 'rescale') <- FALSE
attr(z2., 'GeometricMean') <- 0
class(z2.) <- 'BoxCox'
```

```
all.equal(z2, z2.)
```

```
all.equal(invBoxCox(z2), y0)
```

camelParse	<i>Split a character string where a capital letter follows a lowercase letter</i>
------------	---

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') ) )
```

canbeNumeric

Can a variable reasonably be coerced to numeric?

Description

Can [seq](#) be reasonably applied to *x*? Returns TRUE if yes and FALSE otherwise.

We'd like to use this with, for example, date-time objects in [as.Date](#) and [as.POSIXct](#) formats. However, [as.numeric](#) of such objects is FALSE. Moreover, [as.numeric](#) of [factors](#) is TRUE.

The current algorithm (which may change in the future) returns TRUE if (`mode(x) == 'numeric'`) & (!('levels' %in% names(attributes(x)))).

Usage

```
canbeNumeric(x)
```

Arguments

x an R object

Value

A [logical](#) as described above.

Author(s)

Spencer Graves

See Also

[mode](#)

Examples

```
##
## Examples adapted from "mode"
##
cex4 <- c('letters[1:4]', "as.Date('2014-01-02')",
  'factor(letters[1:4])', "NULL", "1", "1:1", "1i",
  "list(1)", "data.frame(x = 1)", "pairlist(pi)",
  "c", "lm", "formals(lm)[[1]]", "formals(lm)[[2]]",
  "y ~ x", "expression((1))[[1]]", "(y ~ x)[[1]]",
  "expression(x <- pi)[[1]][[1]]")
lex4 <- sapply(cex4, function(x) eval(parse(text = x)))
mex4 <- t(sapply(lex4, function(x)
  c(typeof(x), storage.mode(x), mode(x), canbeNumeric(x))))
dimnames(mex4) <- list(cex4,
  c("typeof(.)", "storage.mode(.)", "mode(.)", 'canbeNumeric(x)'))
mex4

# check
mex. <- as.character(as.logical(c(0, 1, 0, 0, 1, 1, rep(0, 12))))
names(mex.) <- cex4

all.equal(mex4[,4], mex.)
```

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 <code>names(x)</code> 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 hierarchy, 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

<code>x, y</code>	objects whose lengths are to be compared
<code>name.x, name.y</code>	names of <code>x</code> and <code>y</code> to use in a message. Default = <code>deparse(substitute(.), width.cutoff, nlines=1)</code> .
<code>message0</code>	character string to be included with <code>name.x</code> and <code>name.y</code> in a message.
<code>compFun</code>	function to use in the comparison.
<code>action</code>	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)

```

confint.var

Confidence interval for sample variance or standard deviation

Description

Computes the standard normal (i.e., chi-square) confidence intervals for a sample variance or standard deviation.

Usage

```

## S3 method for class 'var'
confint(object, parm, level=0.95, ...)
## S3 method for class 'sd'
confint(object, parm, level=0.95, ...)

```

Arguments

object	a numeric vector possibly with a "df" or "df.residuals" attribute assumed to represent a sample variance, possibly computed as root mean square of residuals from a model.
parm	degrees of freedom in the estimated variance or standard deviation.
level	the confidence level required
...	optional arguments not used.

Details

1. If object is not numeric, throw an error.
2. If parm is missing, look for an attribute of object starting with "df". If present, use that for parm. If parm is absent or not numeric, throw an error.
3. replicate object, parm, and level to the same length. Issue a warning if the longest is not a multiple of the others.
4. $\text{alph2} <- (1 - \text{level})/2$
5. $\text{Qntls} <- \text{cbind}(\text{lower} = \text{qchisq}(\text{alph2}, \text{parm}, \text{lower} = \text{FALSE}), \text{upper} = \text{qchisq}(\text{alph2}, \text{parm}))$
6. $\text{CI} <- (\text{object} * \text{parm} / \text{Qntls})$
7. $\text{attr}(\text{CI}, 'level') <- \text{Level}$
7. $\text{return}(\text{CI})$

Value

a matrix with columns "lower" and "upper", nrow = the longest of the lengths of object, parm, and level, and an attribute "level".

Author(s)

Spencer Graves

References

[Wikipedia, "Standard deviation"](#), accessed 2016-07-06.

See Also

[cor.test](#), [VarCI](#)

Examples

```
##
## 1. simple examples
##
(CI.v <- confint.var(c(1,1,4), c(1, 9, 9)))
(CI.s <- confint.sd(c(1,1,2), c(1, 9, 9)))

# Compare with the examples on Wikipedia
```

```

all.equal(CI.s, sqrt(CI.v))

WikipEx <- t(matrix(c(0.45, 31.9, 0.69, 1.83, 1.38, 3.66),
                    nrow=2))
colnames(WikipEx) <- c('lower', 'upper')

(dCI <- (CI.s-WikipEx))
#Confirm within 2-digit roundoff

max(abs(dCI))<0.0102

##
## 2. test df attributes
##
v <- c(1,1,4)
attr(v, 'df.') <- c(1, 9, 9)
class(v) <- 'var'
vCI <- confint(v)

# check

all.equal(vCI, CI.v)

s <- sqrt(v)
class(s) <- 'sd'
sCI <- confint(s)

# check

all.equal(sCI, CI.s)

```

countByYear

Allocate a total by year

Description

Allocate total to countByYear for a constant count per day between start and end.

Usage

```
countByYear(start, end, total=1)
```


Arguments

start, end objects of class "Date" specifying the start, end, respectively, of the event
total A number to be allocated by year in proportion to the number of days in the event each year.

Value

a numeric vector whose `sum` is total with names for all the years between start and end

Author(s)

Spencer Graves

Examples

```
##  
## 1. All in one year  
##  
start73 <- as.Date('1973-01-22')  
tst1 <- countByYear(start73, start73+99, 123)  
  
# check  
tst1 <- 123  
names(tst1.) <- 1973  
  
all.equal(tst1, tst1.)  
  
##  
## 2. Two years  
##  
tst2 <- countByYear(start73, start73+365, 123)  
  
# check  
dur <- 366  
days1 <- (365-21)  
days2 <- 22  
tst2 <- 123 * c(days1, days2)/dur  
names(tst2.) <- 1973:1974  
  
all.equal(tst2, tst2.)  
  
##  
## 3. Ten years  
##  
tst10 <- countByYear(start73, start73+10*365.2, 123)  
  
# check  
days <- (c(rep(c(rep(365, 3), 366), length=10), 0)  
+ c(-21, rep(0, 9), 22) )
```

```
tst10. <- 123 * days/(10*365.2+1)
names(tst10.) <- 1973:1983
```

```
all.equal(tst10, tst10.)
```

countsByYear	<i>Allocate totals by year</i>
--------------	--------------------------------

Description

Allocate total to countByYear for a constant count per day between start and end for multiple events.

Usage

```
countsByYear(data, start="Start1", end='End1',
             total='BatDeath', event='WarName',
             endNA=max(data[, c(start,end)]))
```

Arguments

data	a data.frame with columns start, end, and total
start, end	columns of data of class Date with start <= end during which total is to be allocated
total	A quantity to be allocated by year giving a constant rate per day.
event	name of the event whose total is to be allocated.
endNA	Date to use if <code>is.na(data[, end])</code> .

Value

a numeric [matrix](#) whose [colSums](#) match total with names for all the years between start and end. The number of columns of the output matrix match the number of rows of data. The [colSums](#) match total.

Author(s)

Spencer Graves

Examples

```
##
## 1. data.frame(WarName, Start1, End1, BatDeath)
##
start73 <- as.Date('1973-01-22')
tstWars <- data.frame(WarName=c('short', '2yr', '10yr'),
  Start1=c(start73, start73+365, start73-365),
  End1=start73+c(99, 2*365, NA),
  BatDeath=c(100, 123, 456))
##
## 2. do
##
deathsByYr <- countsByYear(tstWars,
  endNA=start73+9*365.2)

# check
Counts <- matrix(0, 11, 3,
  dimnames=list(c(1972:1982), tstWars$WarName) )
Counts['1973', 1] <- 100
Counts[as.character(1974:1975), 2] <- with(tstWars,
  countByYear(Start1[2], End1[2], BatDeath[2]) )
Counts[as.character(1972:1982), 3] <- with(tstWars,
  countByYear(Start1[3], start73+9*365.2, BatDeath[3]) )

all.equal(deathsByYr, Counts)
```

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')
```



```

all.equal(lgc13, 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))

```

Date3to1

Convert three YMD vectors to a Date

Description

Given a [data.frame](#) with 3 columns, assume they represent Year, Month and Day and return a vector of class "Date".

Usage

```
Date3to1(data, default='Start')
```

Arguments

<code>data</code>	a data.frame with 3 columns assumed to represent Year, Month and Day.
<code>default</code>	A character string to indicate how missing months and days should be treated. If the first letter is "S" or "s", the default month will be 1 and the default day will be 1. Otherwise, "End" is assumed, for which the default month will be 12 and the default day will be the last day of the month. NOTE: Any number outside the range of 1 to the last day of the month is considered missing and its subscript is noted in the optional attribute "missing".

Details

The data sets from the [Correlates of War](#) project include dates coded in triples of columns with names like c("StartMonth1", "StartDay1", "StartYear1", "EndMonth1", ..., "EndYear2"). This function will accept one triple and translate it into a vector of class "Date".

Value

Returns an object of class "Date" with an optional attribute "missing" giving the indices of any elements with missing months or days, for which a default month or day was supplied.

Author(s)

Spencer Graves

See Also

[dateCols](#)

Examples

```
date.frame <- data.frame(Year=c(NA, -1, 1971:1979),
  Month=c(1:2, -1, NA, 13, 2, 12, 6:9),
  Day=c(0, 0:6, NA, -1, 32) )
```

```
DateVecS <- Date3to1(date.frame)
DateVecE <- Date3to1(date.frame, "End")
```

```
# check
na <- c(1:5, 9:11)
DateVs <- as.Date(c(NA, NA,
  '1971-01-01', '1972-01-01', '1973-01-01',
  '1974-02-04', '1975-12-05', '1976-06-06',
  '1977-07-01', '1978-08-01', '1979-09-01') )
DateVe <- as.Date(c(NA, NA,
  '1971-12-31', '1972-12-31', '1973-12-31',
  '1974-02-04', '1975-12-05', '1976-06-06',
  '1977-07-31', '1978-08-31', '1979-09-30') )
```

```
attr(DateVs, 'missing') <- na
attr(DateVe, 'missing') <- na
```

```
all.equal(DateVecS, DateVs)
```

```
all.equal(DateVecE, DateVe)
```

dateCols

Identify YMD names in a character vector

Description

`grep` for YMD (year, month, day) in `col.names`. Return a named list of integer vectors of length 3 for each triple found.

Usage

```
dateCols(col.names, YMD=c('Year', 'Month', 'Day'))
```

Arguments

col.names	either a character vector in which to search for names matching YMD or an object with non-null colnames
YMD	a character vector of patterns to use in grep to identify triples of columns coding YMD in col.names

Details

The data sets from the [Correlates of War](#) project include dates coded in triples of columns with names like c("StartMonth1", "StartDay1", "StartYear1", "EndMonth1", ..., "EndYear2"). This function will find all relevant date triples in a character vector of column names and return a list of integer vectors of length 3 with names like "Start1", "End1", ..., "End2" giving the positions in col.names of the desired date components.

Algorithm:

1. if(!is.null(colnames(YMD))) YMD <- colnames(YMD)
2. ymd <- [grep](#) for YMD (Year, Month, Day) in col.names.
3. groupNames <- [sub](#) pattern with " in ymd
4. Throw a [warning](#) for any groupNames character string that does not appear with all three of Year, Month, and Day.
5. Return a list of integer vectors of length 3 for each triple found.

Value

Returns a named list of integer vectors of length 3 identifying the positions in col.names of the desired date components.

Author(s)

Spencer Graves

See Also

[Date3to1](#)

Examples

```
##
## 1. character vector
##
colNames <- c('war', 'StartMonth1', 'StartDay1', 'StartYear1',
             'EndMonth1', 'EndMonth2', 'EndDay2', 'EndYear2', 'Initiator')

colNums <- dateCols(colNames)
# Should issue a warning:
```



```

# Warning message:
# In dateCols(colNames) :
#   number of matches for Year = 2 != number of matches for Month = 3

# check
colN <- list(Start1=c(Year=4, Month=2, Day=3),
            End2=c(Year=8, Month=6, Day=7) )

all.equal(colNums, colN)

##
## 2. array
##
A <- matrix(ncol=length(colNames),
           dimnames=list(NULL, colNames))

Anums <- dateCols(A)

# check

all.equal(Anums, colN)

```

Dates3to1

Convert 3-column dates in data to class Date

Description

Return a [data.frame](#) with columns of class "Date" replacing all 3-column dates.

Usage

```
Dates3to1(data, YMD=c('Year', 'Month', 'Day'))
```

Arguments

data	a data.frame assumed to include dates coded in three column sets with names matching YMD.
YMD	a character vector of length 3 of patterns to use in grep to identify triples of columns coding YMD in <code>col.names(data)</code> .

Details

The data sets from the [Correlates of War](#) project include dates coded in triples of columns with names like `c("StartMonth1", "StartDay1", "StartYear1", "EndMonth1", ..., "EndYear2")`. This function will accept a `data.frame` obtained via [read.csv](#) of such a file and replace each such triple with a single column of class 'Date' combining the triple appropriately.

Value

Return a `data.frame` containing the information in data reformatted as described above.

Author(s)

Spencer Graves

See Also

[dateCols Date3to1](#)

Examples

```
cow0 <- data.frame(rec=1:3, startMonth=4:6, startDay=7:9,
  startYear=1971:1973, endMonth1=10:12, endDay1=13:15,
  endYear1=1974:1976, txt=letters[1:3])

cow0. <- Dates3to1(cow0)

# check
cow0x <- data.frame(rec=1:3, txt=letters[1:3],
  start=as.Date(c('1971-04-07', '1972-05-08', '1973-06-09')),
  end1=as.Date(c('1974-10-13', '1975-11-14', '1976-12-15')) )

all.equal(cow0., cow0x)
```

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 <code>read.xls</code> 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 <code>financialCrisisFiles</code> 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) &&
  !fda::CRAN()){
  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. `EI <- if(In) object[[name]] else default`
4. `warn.NULL?`
5. `if(returnName) return(as.character(EI)) else return(eval(EI, 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)
}

```

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 identify 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

[stringi-package 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`1')
```

 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

<code>x, y</code>	two vectors of the same class or to be coerced to the same class.
<code>proportion</code>	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 <code>compareLengths</code> to improve the value of any diagnostic message in case lengths are not compatible.
message0	A character string to be passed with argnames to <code>compareLengths</code> to improve the value of any diagnostic message in case lengths are not compatible.
argsChk	a list as returned by <code>interpChkArgs</code>
...	optional arguments for <code>compareLengths</code>

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 algorithm 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" algorithm 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:

$x_{Out} \leftarrow x \cdot (1 - \text{proportion}) + y \cdot \text{proportion}$

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

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

- 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

<code>object</code>	A call , function , list or <code>data.frame</code> with names possibly matching <code>pairs[1:2]</code> . When names matching both of <code>pairs[1:2]</code> , they are converted to potentially common names using <code>sub(pairs[i], pairs[3], ...)</code> . When matches are found among the potentially common names, they are passed with <code>.proportion</code> to interpChar to compute an interpolation. The matches are removed and replaced with the interpolant, shortened by excluding any rows for which <code>.proportion</code> is outside <code>validProportion</code> . 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 <code>.proportion</code> outside <code>validProportion</code> . Thus, if <code>x.0</code> is found without <code>x.1</code> , <code>x.0</code> is removed and replaced by <code>x</code> .
<code>nFrames</code>	number of distinct plots to create.
<code>iFrame</code>	integer giving the index of the single frame to create. Default = <code>nFrames</code> .

	An error is thrown if both <code>iFrame</code> and <code>.proportion</code> are not NULL.
<code>endFrames</code>	Number of frames to hold constant at the end.
<code>.proportion</code>	a numeric vector assumed to lie between 0 and 1 specifying how far to go from <code>suffixes[1]</code> to <code>suffixes[2]</code> . For example, if <code>x.0</code> and <code>x.1</code> are found and are numeric, $x = x.0 + .proportion * (x.1 - x.0)$. Rows of <code>x</code> and any other element of object of the same length are dropped for any <code>.proportion</code> outside <code>validProportion</code> .
	An error is thrown if both <code>iFrame</code> and <code>.proportion</code> are not NULL.
<code>envir</code>	environment / list to use with <code>codeobject</code> , which can optionally provide other variables to compute what gets plotted; see the example below using this argument.
<code>pairs</code>	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 <code>sub</code> to convert each <code>pairs[1:2]</code> name found to the desired name of the interpolate. Common names found are then passed with <code>.proportion</code> to <code>interpChar</code> , which does the actual interpolation. (2,3) <code>interpPairs</code> also calls <code>checkNames(object, avoid = pairs[c(1, 3, 2, 5)])</code> . This confirms that object has <code>names</code> , and all such names are unique. If object does not have names or has some duplicate names, the <code>make.names</code> is called to fix that problem, and any new names that match <code>pairs[1:2]</code> are modified using <code>sub</code> to avoid creating a new match. If the modification still matches <code>pairs[1:2]</code> , it generates an error.
<code>validProportion</code>	Range of values of <code>.proportion</code> to retain, as noted with the discussion of the object argument.
<code>message0</code>	a character string passed to <code>interpChar</code> to improve the value of diagnostic messages
<code>...</code>	optional arguments for <code>sub</code>

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, the 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 $N = \text{length}(\text{.proportion})$. Find all vectors and `data.frames` 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))

```

logVarCor

Log-diagonal representation of a variance matrix

Description

Translate a square symmetric matrix with positive diagonal elements into a vector of the logarithms of the diagonal elements with the correlations as an attribute, and vice versa.

Usage

```
logVarCor(x, corr, ...)
```

Arguments

x	If a matrix, translate into a vector with a "corr" attribute. If a vector, translate into a matrix.
corr	optional vector of correlations for the <code>lower.tri</code> portion of a covariance matrix whose diagonal is <code>exp(x)</code> . Use a "corr" attribute of x only if this argument is <code>missing</code> .
...	(not currently used)

Value

if($\text{length}(\text{dim}(x))=2$) return $\text{log}(\text{diag}(x))$ with an attribute "corr" equal to the `lower.tri` of `cov2cor(x)`.

Otherwise, return a covariance matrix from x as described above.

Author(s)

Spencer Graves

See Also

`log diag cov2cor lower.tri pdLogChol` converts a k -dimensional covariance matrix into a vector of length `choose(k+1, 2)`. By contrast, `logVarCor` returns a vector of length k with a "corr" attribute of length `choose(k, 2)`.

Examples

```
##
## 1. Trivial 1 x 1 matrix
##
# 1.1. convert vector to "matrix"
mat1 <- logVarCor(1)
# check

all.equal(mat1, matrix(exp(1), 1))

# 1.2. Convert 1 x 1 matrix to vector
lVcD1 <- logVarCor(diag(1))
# check
lVcD1. <- 0
attr(lVcD1., 'corr') <- numeric(0)

all.equal(lVcD1, lVcD1.)

##
## 2. simple 2 x 2 matrix
##
# 2.1. convert 1:2 into a matrix
lVc2 <- logVarCor(1:2)
# check
lVc2. <- diag(exp(1:2))

all.equal(lVc2, lVc2.)

# 2.2. Convert a matrix into a vector
lVc2d <- logVarCor(diag(1:2))
# check
lVc2d. <- log(1:2)
```

```

attr(1VC2d., 'corr') <- 0

all.equal(1VC2d, 1VC2d.)

##
## 3. 3-d covariance matrix with nonzero correlations
##
# 3.1. Create matrix
(ex3 <- tcrossprod(matrix(c(rep(1,3), 0:2), 3)))
dimnames(ex3) <- list(letters[1:3], letters[1:3])

# 3.2. Convert to vector
(Ex3 <- logVarCor(ex3))

# check
Ex3. <- log(c(1, 2, 5))
names(Ex3.) <- letters[1:3]
attr(Ex3., 'corr') <- c(1/sqrt(2), 1/sqrt(5), 3/sqrt(10))

all.equal(Ex3, Ex3.)

# 3.3. Convert back to a matrix
Ex3.2 <- logVarCor(Ex3)
# check

all.equal(ex3, Ex3.2)

```

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

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

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 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))

```

matchName	<i>Match surname and givenName in a table</i>
-----------	---

Description

Use [parseName](#) to split a name into surname and givenName, the look for matches in table.

Usage

```

matchName(x, data, Names=1:2,
          nicknames=matrix(character(0), 0, 2),
          namesNotFound="attr.replacement", ...)
matchName1(x1, data, name=data[, 1],
           nicknames=matrix(character(0), 0, 2), ...)

```

Arguments

x	One of the following: <ul style="list-style-type: none"> A character matrix or data.frame with the same number of rows as data. The best partial match is sought in Names. The algorithm stops when a unique match is found; any remaining columns of x are then ignored. Any nicknames are ignored for the first column but not for subsequent columns. A character vector whose length matches the number of rows of data. This will be replaced by <code>parseName(x)</code>.
data	a character matrix or a data.frame . If surname and givenName are character vectors of names, their length must match the number of rows of data.
Names	One of the following in which matches for x will be sought: <ul style="list-style-type: none"> A character vector or matrix or a data.frame for which <code>NROW(Names) == nrow(data)</code>. Something to select columns of data to produce a character vector or matrix or data.frame via <code>data[, Names]</code>. In this case, accents will be stripped using subNonStandardNames.
nicknames	a character matrix with two columns, each row giving a pair of names like "Pete" and "Peter" that should be regarded as equivalent if no exact match(es) is(are) found.
...	optional arguments passed to subNonStandardNames

x1	a character vector of names to match name. NOTE: matchName calls subNonStandardNames, but matchName1 does not. Thus, x1 is assumed to NOT to contain characters not in standard English.
name	A character vector or matrix for which NROW(name) == nrow(data). NOTE: matchName calls subNonStandardNames, but matchName1 does not. Thus, name is assumed to NOT to contain characters not in standard English.
namesNotFound	character vector passed to subNonStandardNames and used to compute any "namesNotFound" attribute of the object returned by parseName.

Details

*** 1. matchName(x, data, Names, nicknames, ...):

1.1. if(length(dim(x)<2))x <- parseName(x, ...)

1.2. x1 <- matchName1(x[, 1], cata, Names[1], ...)

1.3. For any component i of x1 with multiple rows, let x1i <- matchName1(x[i, 2], x1[[i]], Name[-1], nicknames=nicknames, ...). If nrow(x1i)>0, x1[[i]] <- x1i; else leave unchanged.

1.4. return x1

=====

*** 2. matchName1(x1, data, name, nicknames, ...):

2.1. If name indicates a column of data, replace with data[, name].

2.2. xsplit <- strsplit(x1, ' ')

2.3. nx <- length(x1); xlist <- vector(nx, mode='list')

2.4. for(j in 1:nx):

2.5. xj <- xsplit[[j]]

2.6. let jd = the subset of names that match xj or subNonStandardNames(xj) or nicknames of xj; xlist[j] <- jd.

2.7. return xlist

Value

matchName returns a list of the same length as x, each of whose components is object obtained as a subset of rows of data or NULL if no acceptable matches are found. The list may have an attribute "namesNotFound" as determined per the argument of that name.

matchNames1 returns a list of vectors of integers for subsets of data matching x1.

Author(s)

Spencer Graves

See Also

[parseName](#) [subNonStandardNames](#)

Examples

```

##
## 1. Names to match exercising many possible combinations
##    of surname with 0, 1, >1 matches possibly after
##    replacing with subNonStandardNames
##    combined with possibly multiple givenName combinations
##    with 0, 1, >1 matches possibly requiring replacing with
##    subNonStandardNames or nicknames
##
# NOTE: "-" could also be "e" with an accent;
#    not included with this documentation, because
#    non-English characters generate warnings in standard tests.
Names2mtch <- c("Andr_ Bruce C_rdenas", "Dolores Ella Feinstein",
               "George Homer", "Inez Jane Kappa", "Luke Michael Noel",
               "Oscar Papa", "Quincy Ra_l Stevens",
               "Thomas U. Vel_zquez", "William X. Young",
               "Zebra")

##
## 2. Data = matrix(..., byrow=TRUE) to exercise the combinations
##    the combinations from 1
##
Data1 <- matrix(c("Feld", "Don", "789",
                 "C_rdenas", "Don", "456",
                 "C_rdenas", "Andre B.", "123",
                 "Smith", "George", "aaa",
                 "Young", "Bill", "369"),
               ncol=3, byrow=TRUE)
Data1. <- subNonStandardNames(Data1)
##
## 3. matchName1
##
parceNm1 <- parseName(Names2mtch)
match1.1 <- matchName1(parceNm1[, 'surname'], Data1.)

# check
match1.1s <- vector('list', 10)
match1.1s[[1]] <- 2:3
match1.1s[[9]] <- 5
names(match1.1s) <- parceNm1[, 'surname']

all.equal(match1.1, match1.1s)

##
## 4. matchName1 with name = multiple columns
##
match1.2 <- matchName1(c('Cardenas', 'Don'), Data1.,
                      name=Data1.[, 1:2])

# check
match1.2a <- list(Cardenas=2:3, Don=1:2)

```

```

all.equal(match1.2, match1.2a)

##
## 5. matchName
##
nickNames <- matrix(c("William", "Bill"), 1, byrow=TRUE)

match1 <- matchName(Names2mtch, Data1, nicknames=nickNames)

# check
match1a <- list("Cardenas, Andre Bruce"=Data1[3,, drop=FALSE ],
               "Feinstein, Dolores Ella"=NULL,
               "Homer, George"=NULL, "Kappa, Inez Jane"=NULL,
               "Noel, Luke Michael"=NULL, "Papa, Oscar"=NULL,
               "Stevens, Quincy Raul"=NULL,
               "Velazquez, Thomas U."=NULL,
               "Young, William X."=Data1[5,, drop=FALSE],
               "Zebra"=NULL)

all.equal(match1, match1a)

##
## 6. namesNotFound
##
tstNotFound <- matchName('xx_x', Data1)

# check
tstNF <- list('xx_x'=NULL)
attr(tstNF, 'namesNotFound') <- 'xx_x'

all.equal(tstNotFound, tstNF)

##
## 7. matchName(NULL) to simplify use
##
mtchNULL <- matchName(NULL, Data1)

all.equal(mtchNULL, NULL)

```

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 character 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 concatenate 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 preceding 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 concatenated with a preceding line because the two lines contained unmatched Quote characters, and concatenating 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 concatenated 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	<i>Expand a dataset on some members of the US Congress to the entire membership</i>
---------------------	---

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 <code>regexpr(names(default)[i], tolower(names(x)))</code> .

Details

1. `keyx <- with(x, paste(houseSenate, state, District, sep=":"))`
2. `keyy <- with(UScongress(houseSenate, state, District, sep=":"))`
3. `notx <- !is.element(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 Chamber: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(Chamber=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=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('R', 'D', 'R', 'D', 'R', 'D'),
  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 )

if(!fda::CRAN()){
  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

<code>x</code>	a <code>data.frame</code> whose columns include <code>Office</code> , <code>surname</code> , and <code>givenName</code> .
<code>vote</code>	a <code>data.frame</code> with column names which when forced <code>tolower</code> would match <code>surname</code> , <code>givenname</code> , and <code>vote</code> . However, the <code>givenname</code> may not be complete, so use it only if the <code>surname</code> is not sufficient.
<code>Office</code>	Either "House" or "Senate"; ignored if <code>vote</code> includes a column <code>Office</code> .
<code>vote.x</code>	name of a column of <code>x</code> containing a vote to be updated with the <code>vote</code> column of the <code>vote data.frame</code> . If <code>missing</code> and <code>x</code> has a column with a name matching "vote", then <code>vote.x</code> is that column. If <code>missing</code> but <code>x</code> has no such column, then append a column to <code>x</code> with the name of the <code>vote</code> column of the <code>vote data.frame</code> .
<code>check.x</code>	logical: If TRUE, check for rows of <code>x[, vote.x]</code> that are NOT in <code>vote</code> 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`, `levv2`, 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 information.
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 $(\text{is.null}(x) \parallel (\text{length}(x) == 0) \parallel (\text{max}(\text{nchar}(x)) == 0))$.

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 $\text{max}(\text{nchar}(x)) == 0$. 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)
```

Newdata

Create a new data.frame for predict

Description

Generate a new [data.frame](#) or [matrix](#) from another with column(s) selected by `x` adopting `n` values in `range(data[,x])` and all other columns constant.

If `canbeNumeric(x)` is TRUE, the output with has `x` adopting `n` values in the `range(x)` and all other numeric variables at their [median](#) and other variables at their most common values.

If `canbeNumeric(x)` is FALSE, the output with has `x` adopting all possible values of `x` with all other variables at the same constant values as when `canbeNumeric(x)` is TRUE (and `n` is ignored). If `x` has a `levels` attribute, the possible values are defined by that `levels` attribute. Otherwise, it is defined by `unique(x)`.

This is designed to create a new [data.frame](#) to be used as newdata for `predict`.

Usage

```
Newdata(data, x, n, na.rm=TRUE)
```

Arguments

<code>data</code>	a data.frame or matrix.
<code>x</code>	name of a column of data. If NA or NULL, select all columns of data.
<code>n</code>	an integer vector indicating the number of levels of <code>data[, x]</code> if <code>canbeNumeric(data[, x])</code> . If <code>canbeNumeric(data[, x])</code> is FALSE, take at most <code>n</code> of the most popular levels. Default is 2 if <code>length(x) > 1</code> or if <code>x</code> is either NA or NULL. If <code>n = 1</code> , use the median for <code>canbeNumeric</code> and the most popular level otherwise. If <code>n < 1</code> , drop that variable.
<code>na.rm</code>	logical passed to <code>range(x)</code>

Details

1. Check data, `x`.
2. If `canbeNumeric(x)` is TRUE, let `xNew` be `n` values spanning `range(x)`. Else, let `xNew <- levels(x)`.
3. If `is.null(xNew)`, set it to `sort(unique(x))`.
4. let `newDat <- data[rep(1, n),]`, and replace `x` by `xNew`.
5. `otherVars <- colnames(data) != x`
6. for(`x2` in `otherVars`)replace `newDat[, x2]`: If `canbeNumeric(x2)` is TRUE, use `median(x2)`. Otherwise, use its (first) most common value.

Value

A `data.frame` with `n` rows and columns matching those of `data`, as described above.

Author(s)

Spencer Graves

See Also

`predict.lm`

Examples

```
##
## 1. A reasonable test with numerics, dates,
##    an ordered factor and character variables
##
xDate <- as.Date('2001-02-03')+1:4
tstDF <- data.frame(x1=1:4, xDate=xDate,
  xD2=as.POSIXct(xDate),
  sex=ordered(c('M', 'F', 'M', 'F'))),
  huh=letters[c(1:3, 3)], stringsAsFactors=FALSE)

newDat <- Newdata(tstDF, 'xDate', n=5)

# check
newD <- data.frame(x1=2.5,
  xDate=xDate[1]+seq(0, 3, length=5),
  xD2=as.POSIXct(xDate[2]+0.5),
  sex=ordered(c('M', 'F', 'M', 'F'))[2],
  huh=letters[3], stringsAsFactors=FALSE)
attr(newD, 'out.attrs') <- attr(newDat, 'out.attrs')

all.equal(newDat, newD)

##
## 2. Test with only one column
```

```
##
newDat1 <- Newdata(tstDF[, 2, drop=FALSE], 'xDate', n=5)

# check
newDat1. <- newD[, 2, drop=FALSE]
attr(newDat1., 'out.attrs') <- attr(newDat1, 'out.attrs')

all.equal(newDat1, newDat1.)

##
## 3. Test with a factor
##
newSex <- Newdata(tstDF, 'sex')

# check
newS <- with(tstDF, data.frame(
  x1=2.5, xDate=xDate[1]+1.5,
  xD2=as.POSIXct(xDate[1]+1.5),
  sex=ordered(c('M', 'F'))[2:1],
  huh=letters[3], stringsAsFactors=FALSE) )
attr(newS, 'out.attrs') <- attr(newSex, 'out.attrs')

all.equal(newSex, newS)

##
## 4. Test with an integer column number
##
newDat2 <- Newdata(tstDF, 2, n=5)

# check

all.equal(newDat2, newD)

##
## 5. Test with all
##
NewAll <- Newdata(tstDF)

# check
tstLvls <- as.list(tstDF[c(1, 4), ])
tstLvls$sex <- tstDF$sex[2:1]
tstLvls$huh <- letters[c(3, 1)]
tstLvls$stringsAsFactors <- FALSE

NewA. <- do.call(expand.grid, tstLvls)
attr(NewA., 'out.attrs') <- attr(NewAll, 'out.attrs')

all.equal(NewAll, NewA.)
```

parseCommas	<i>Convert character string with Dollar signs and commas to numerics</i>
-------------	--

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 converted 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". If only one name is provided (without punctuation), it is assumed to be the givenName; see Wikipedia, "[Given name](#)" and "[Surname](#)".

Usage

```
parseName(x, surnameFirst=(median(regexpr(',', x))>0),
          suffix=c('Jr.', 'I', 'II', 'III', 'IV', 'Sr.', 'Dr.',
                  'Jr', 'Sr'),
          fixNonStandard=subNonStandardNames,
          removeSecondLine=TRUE,
          namesNotFound="attr.replacement", ...)
```

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.
removeSecondLine	logical: If TRUE, delete anything following "\n" and return it as an attribute "secondLine".
namesNotFound	character vector passed to subNonStandardNames and used to compute any "namesNotFound" attribute of the object returned by parseName.
...	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., "Anastasio Somoza Debayle", "Somoza Debayle" give the (first) surnames of Anastasio'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.

This matrix also has a "namesNotFound" attribute if one is returned by subNonStandardNames.

Author(s)

Spencer Graves

See Also

[strsplit identity subNonStandardNames](#)

Examples

```
##
## 1. Parse standard first-last name format
##
tstParse <- 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', 'Sting', 'Colette, ')

parsed <- parseName(tstParse)

tstParse2 <- 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', '', 'Sting', 'Colette', ''),
                  ncol=2, byrow=TRUE)
# NOTE: The 'Anastasio Somoza Debayle' 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(tstParse2) <- c("surname", 'givenName')

all.equal(parsed, tstParse2)

##
## 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', '', 'Sting', 'Colette,')
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', '', 'Sting', 'Colette', ''),
              ncol=2, byrow=TRUE)
colnames(tst5) <- c("surname", 'givenName')

all.equal(tst4, tst5)
```

```

##
## 3. secondLine
##
L2 <- parseName(c('Adam\n2nd line', 'Ed \n --Vacancy', 'Frank'))

# check
L2. <- matrix(c('', 'Adam', '', 'Ed', '', 'Frank'),
              ncol=2, byrow=TRUE)
colnames(L2.) <- c('surname', 'givenName')
attr(L2., 'secondLine') <- c('2nd line', ' --Vacancy', NA)

all.equal(L2, L2.)

##
## 4. Force surnameFirst when in a minority
##
snf <- c('Sting', 'Madonna', 'Smith, Al')
SNF <- parseName(snf, surnameFirst=TRUE)

# check
SNF2 <- matrix(c('', 'Sting', '', 'Madonna', 'Smith', 'Al'),
               ncol=2, byrow=TRUE)
colnames(SNF2) <- c('surname', 'givenName')

all.equal(SNF, SNF2)

##
## 5. nameNotFound
##
noSub <- parseName('xx_x')

# check
noSub. <- matrix(c('', 'xx_x'), 1)
colnames(noSub.) <- c('surname', 'givenName')
attr(noSub., 'namesNotFound') <- 'xx_x'

all.equal(noSub, noSub.)

```

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

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

Details

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

Value

list with the following components:

<code>rawResults</code>	character vector of the raw results from the ping command
<code>rawNumbers</code>	numeric vector of the times measured
<code>counts</code>	numeric vector of numbers of packets sent, received, and lost
<code>p.lost</code>	proportion lost = lost / sent
<code>stats</code>	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)
```

pmatch2

Value matching or partial matching

Description

pmatch2 returns a list of the positions of matches or partial matches of x in table.

This does sloppy matching to find "Peter" to match "Pete" only if "Pete" is not in table, and we want "John Peter" if neither "Pete" nor "Peter" are in table.

Usage

```
pmatch2(x, table)
```

Arguments

x	the values to be matched
table	the values to be matched against

Details

1. nx <- length(x); out <- vector(nx, "list"); names(out) <- x
2. for(ix in seq(length=nx)):
 3. xi <- which(x[ix] %in% table)
 4. if(length(xi)<1) xi <- grep(paste0('^', x[ix]), table).
 5. if(length(xi)<1)xi <- grep(x[ix], table).
 6. out[[ix]] <- xi

Value

A list of integer vectors indicating the positions in table matching each element of x

Author(s)

Spencer Graves

See Also

[match](#) [pmatch](#) [grep](#) [matchName](#)

Examples

```
##
## 1. common examples
##
x2match <- c('Pete', 'Peter', 'Ma', 'Mo', 'Paul',
            'Cardenas')

tbl <- c('Peter', 'Mary', 'Martha', 'John Paul', 'Peter',
        'Cardenas', 'Cardenas')

x2mtchd <- pmatch2(x2match, tbl)

# answer
x2mtchd. <- list(Pete=c(1, 5), Peter=c(1, 5), Ma=2:3,
               Mo=integer(0), Paul=4, Cardenas=6:7)

all.equal(x2mtchd, x2mtchd.)

##
```

```
## 2. strange cases that caused errors and are now warnings
##
huh <- pmatch2("(7", tbl)

# answer
huh. <- list("(7"=integer(0))

all.equal(huh, huh.)
```

qqnorm2

Normal Probability Plot with Multiple Symbols

Description

Create a normal probability plot with one line and different symbols for the values of another variable, *z*.

qqnorm2 produces an object of class qqnorm2, whose plot method produces the plot.

To create a normal normal probability plots with multiple lines, see [qqnorm2t](#) or [qqnorm2s:x'](#)

- [qqnorm2s](#) produces a plot with multiple lines specified either by different names in a character vector *y* or by different [data.frames](#) in a list *data.*, with different points labeled according to the different levels of *z*.
- [qqnorm2t](#) produces a plot with multiple lines with *y* split on different levels of *x*, optionally with different points labeled according to different levels of *z*.

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

- | | |
|----------|---|
| <i>y</i> | For qqnorm2, <i>y</i> is a numeric vector for which a normal probability plot is desired. For plot.qqnorm2, <i>y</i> is ignored; it is included, because the generic plot function requires it. |
| <i>z</i> | A variable to indicate different plotting symbols.
NOTE: is.logical(<i>z</i>) is replaced by <i>z</i> <- as.character(<i>z</i>).
Otherwise, pch[<i>z</i>] would delete symbols in pch for which <i>z</i> is FALSE and would recycle the remaining symbols. That would rarely be what we want. |

plot.it	logical: Should the result be plotted?
datax	The datax argument of <code>qqnorm</code> : 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 <code>qqnorm</code> .)
x	an object of class <code>qqnorm2</code> .
pch	a named vector of the plotting symbols to be used with names corresponding to the levels of z. If pch is provided, it must either have names corresponding to levels of z, or z must be integers between 1 and <code>length(pch)</code> . Otherwise, if z takes levels FALSE and TRUE (or 0 and 1), <code>pch=c(4, 1)</code> to plot an "x" for FALSE and "o" for TRUE. Or if z assumes integer values between 0 and 255, by default, the symbols are chosen as described with <code>points</code> . NOTE: *** <code>points.qqnorm2</code> may not work properly for z being integer between 0 and 255. <code>lines.qqnorm2</code> is more likely to work in such cases. *** No time to fix this as of 2018-01-20. Otherwise, by default, z is coerced to <code>character</code> , and the result is plotted.
...	Optional arguments. For <code>plot.qqnorm2</code> , they are passed to <code>plot</code> . For <code>qqnorm2</code> , they are passed to <code>qqnorm</code> and to <code>plot.qqnorm2</code> .

Details

For `qqnorm2`:

```
qq1. q2 <- qqnorm(y, datax=datax, ...)
```

```
qq2. q2[["z"]] <- z
```

```
qq3. q2[["pch"]] gets whatever pch decodes to.
```

```
qq4. Silently return(list(x, y, z, pch, ...)), where "x" and "y" are as returned by qqnorm in step 1 above. If pch is not provided and z is not logical or positive integers, then z itself will be plotted and "pch" will not be in the returned list.
```

For `plot.qqnorm2`:

```
plot1. plot(x$x, x$y, type="n", ...) with "..." taking precedence over x, where the same plot argument appears in both.
```

```
plot2. if(type %in% c('l', 'b', 'c', 'o')) lines(x$x, x$y, ...)
```

```
plot3. if(type %in% c('p', 'b', 'o')): if(is.null(x$z))points(x$x, x$y, ...) else if(is.logical(x$z))points(x$x, x$y, pch=x$pch[x$z], ...) else if(is.numeric(x$z) && (min(z0 <- round(x$z))>0) && (max(abs(x$z-z0))<10*.Machine$double.eps)) points(x$x, x$y, pch=x$pch[x$z], ...) else text(x$x, x$y, x$z, ...)
```

```
For lines.qqnorm2 lines1. if(type != 'p')lines(x$x, x$y, ...);
```

```
lines2. if(type %in% c('p', 'b', 'o')) if(is.null(pch))text(x$x, x$y, x$z, ...) else if(is.character(pch))text(x$x, x$y, x$pch[x$z], ...) else points(x$x, x$y, pch=x$pch[x$z], ...)
```

```
For points.qqnorm2 points1. if(type %in% c('p', 'b', 'o')) if(is.null(pch))text(x$x, x$y, x$z, ...) else if(is.character(pch))text(x$x, x$y, x$pch[x$z], ...) else points(x$x, x$y, pch=x$pch[x$z], ...)
```

```
points2. if(!(type %in% c('p', 'n')) lines(x$x, x$y, ...)
```

Value

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

Author(s)

Spencer Graves

See Also

[qqnorm](#), [qqnorm2s](#), [qqnorm2t](#) [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)
```

```
##
## Test lines(qn4) without z
##
# just as a test, so this code can be used
# in other contexts
qn4. <- qn4
qn4.$z <- NULL
plot(qn4.)
```

qqnorm2s

Normal Probability Plot with Multiple Lines and Multiple Symbols

Description

Create a normal probability plot with one line for each y variable or each `data.frame` in a list `data.` with different plotting symbols for the values of `z`.

To create a normal probability plot with one y variable split on a `link{factor}` or `character` variable `x`, see [qqnorm2t](#).

`qqnorm2s` produces an object of class `qqnorm2s`, whose `plot` method produces the plot.

Usage

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

Arguments

- | | |
|----------------|--|
| <code>y</code> | a <code>character</code> vector of names of columns of <code>data.</code> for which normal probability plots are desired. <code>data.</code> is either a <code>data.frame</code> or a list of <code>data.frames</code> of the same length as <code>y</code> , with <code>y[i]</code> being the name of a column of the <code>data.frame</code> <code>data.[[i]]</code> . <code>z</code> is a similar character vector of names of columns of <code>data.</code> , which identify symbols for plotting different points in a normal probability plot. The lengths of <code>y</code> , and <code>z</code> must match the number of <code>data.frames</code> in <code>data.</code> ; if not, the lengths of the shorter are replicated to the length of the longest before computations begin.
For <code>plot.qqnorm2s</code> , <code>y</code> is ignored; it is included, because the generic <code>plot</code> function requires it. |
| <code>z</code> | A character vector giving the names of columns of <code>data.</code> to indicate different plotting symbols. <code>z</code> should be the same length as <code>y</code> and must equal the number of <code>data.frames</code> in the list <code>data.</code> of <code>data.frames</code> . If not, the shorter are replicated to the length of the longer. |

<code>data.</code>	a data.frame or a list of data.frames with columns named in <code>y</code> and <code>z</code> .
<code>plot.it</code>	logical: Should the result be plotted?
<code>datax</code>	The <code>datax</code> argument of qqnorm : If TRUE, the data are displayed on the horizontal rather than the vertical axis. (The default value for <code>datax</code> is the opposite of that for qqnorm .)
<code>outnames</code>	Names for the components of the <code>qqnorm2s</code> object returned by the <code>qqnorm2s</code> function.
<code>pch</code>	a named vector of the plotting symbols to be used with names corresponding to the levels of <code>z</code> . By default, if <code>z</code> takes levels FALSE and TRUE (or 0 and 1), <code>pch=c(4, 1)</code> to plot a "x" for FALSE and "o" for TRUE. If <code>z</code> assumes integer values between 0 and 255, by default, the symbols are chosen as described with points . Otherwise, by default, <code>z</code> is coerced to character , and the result is plotted. If <code>pch</code> is provided, it must either have names corresponding to levels of <code>z</code> , or <code>z</code> must be integers between 1 and <code>length(pch)</code> .
<code>col</code>	A vector indicating the colors corresponding to each element of <code>y</code> . Defaults to <code>rep(c(1:4, 6), length=length(y))</code> , with 1:4 and 6 being black, red, green, blue, and pink.
<code>x</code>	an object of class <code>qqnorm2</code> .
<code>legend.</code>	A list with components <code>pch</code> and <code>col</code> providing information for legend to identify the plotting symbols (<code>pch</code>) and colors (<code>col</code>). By default, <code>pch = list(x='right', legend=names(qq2s[[1]][['pch']]), pch=qq2s[[1]][[1]])</code> , where <code>qq2s</code> is described below in details. Similarly, by default, <code>lines = list(x='bottomright', legend=y, lty=1, pch=NA, col=qq2s[[1]][[1]])</code> .
<code>...</code>	Optional arguments. For <code>plot.qqnorm2s</code> , they are passed to <code>plot</code> . For <code>qqnorm2s</code> , they are passed to qqnorm2 and to <code>plot.qqnorm2s</code> .

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' )
##
## Split one data.frame
##
tstDF2. <- rbind(cbind(tstDF2, z1=1),
  cbind(tstDF2b, z1=2) )
```

```

Qn2. <- qqnorm2s('y', 'z1', tstDF2.)
# Plot has only one line, because only 1 y variable.
##
## Two data.frames without z
##
Qn2.0 <- qqnorm2s('y',
  data.=list(tstDF2, tstDF2b),
  outnames=c('ok', 'oops'), log='x' )

```

qqnorm2t

Normal Probability Plot with Multiple Lines and Multiple Symbols

Description

Create a normal probability plot of y with one line for each level of a [factor](#) or [character](#) variable x and (optionally) different symbols for the different levels of a variable z.

To create a normal probability plot with one line for each of multiple y variables, see [qqnorm2s](#).

To create a normal probability plot with one line and different symbols for each level of a variable z, see [qqnorm2](#).

Usage

```

qqnorm2t(y, x, z=NULL, data., plot.it=TRUE,
  datax=TRUE, outnames=NULL, pch=NULL,
  col=c(1:4, 6), legend.=NULL, ...)

```

Arguments

y	a character vector of length 1 with the name of a column of data. for which normal probability plots are desired, with one line for each level of x.
x	a factor or character vector indicating how to split y for plotting.
z	A character vector giving the name of a column of data. to indicate different plotting symbols.
data.	a data.frame with columns named in y, x, 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. Equal to the levels of x by default.
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 <code>z</code> assumes integer values between 0 and 255, by default, the symbols are chosen as described with points . Otherwise, by default, <code>z</code> is coerced to character , and the result is plotted. If <code>pch</code> is provided, it must either have names corresponding to levels of <code>z</code> , or <code>z</code> must be integers between 1 and <code>length(pch)</code> .
<code>col</code>	A vector indicating the colors corresponding to each element of <code>x</code> . Defaults to <code>rep(c(1:4, 6), length=length(x))</code> , with 1:4 and 6 being black, red, green, blue, and pink.
<code>legend.</code>	A list with components <code>pch</code> and <code>col</code> providing information for legend to identify the plotting symbols (<code>pch</code>) and colors (<code>col</code>). By default, <code>pch = list(x='right', legend=names(qq2s[[1]][['pch']]), pch=qq2s[[1]][[1]])</code> , where <code>qq2s</code> is described below in details. Similarly, by default, <code>lines = list(x='bottomright', legend=y, lty=1, pch=NA, col=qq2s[[1]][[2]])</code> .
<code>...</code>	Optional arguments. For <code>plot.qqnorm2s</code> , they are passed to <code>plot</code> . For <code>qqnorm2s</code> , they are passed to qqnorm2 and to <code>plot.qqnorm2s</code> .

Details

`data.` is split by `x` and the result is passed to `qqnorm2s`

Value

Returns an object of class [qqnorm2s](#).

Author(s)

Spencer Graves

See Also

[qqnorm2](#), [qqnorm2s](#), [plot](#)

Examples

```
##
## One data.frame
##
tstDF2 <- data.frame(y=1:6, x=c('a','b'),
  z2=c(TRUE, TRUE, FALSE),
  z3=c('tell', 'me', 'why') )
# produce the object and plot it
Qnt <- qqnorm2t('y', 'x', 'z2', tstDF2)

# plot the object previously created
plot(Qnt)

Qnt0 <- qqnorm2t('y', 'x', data.=tstDF2)
# without z
qqnorm2t('y', 'x', data.=tstDF2)
```

rasterImageAdj	<i>rasterImage adjusting to zero distortion</i>
----------------	---

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

<code>image</code>	a raster object, or an object that can be coerced to one by as.raster .
<code>xleft</code>	a vector (or scalar) of left x positions.
<code>ybottom</code>	a vector (or scalar) of bottom y positions.
<code>xright</code>	a vector (or scalar) of right x positions.
<code>ytop</code>	a vector (or scalar) of top y positions.
<code>angle</code>	angle of rotation in degrees, anti-clockwise about the centroid of image. NOTE: <code>rasterImage</code> rotates around <code>(xleft, ybottom)</code> . <code>rasterImage</code> rotates around the center $((xleft+xright)/2, (ybottom+ytop)/2)$. See the examples.
<code>interpolate</code>	a logical vector (or scalar) indicating whether to apply linear interpolation to the image when drawing.
<code>xsub, ysub</code>	subscripts to subset image
<code>...</code>	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. $(dX, dY) = \text{imageUnitsAdj}/2 =$ 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 character.
na.strings	character string(s) that translate into NA
...	optional arguments for strsplit

Details

1. `txt <- readLines(file)`
2. Split into fields.
3. Identify 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.=
"https://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
PVIInum	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
PVIInum	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 party 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.()
}
```

readDates3to1

read.csv with Dates in 3 columns

Description

[read.csv](#), converting 3-column dates into vectors of class 'Date'.

Usage

```
readDates3to1(file, YMD=c('Year', 'Month', 'Day'), ...)
```

Arguments

file	the name of a file from which the data are to be read.
YMD	Character vector of length 3 passed to dateCols
...	optional arguments for read.csv

Details

Some files (e.g., from the [Correlates of War](#) project) have dates specified in three separate columns with names like "startMonth1", "startDay1", "startYear1", "endMonth1", ..., "endYear2". This function looks for such triples and replaces each found with a single column with a name like, "start1", "end1", ..., "end2".

ALGORITHM

1. `dat <- read.csv(file, ...)`
2. `Dates3to1(dat, YMD)`

Value

a [data.frame](#) with 3-column dates replace by single-column vectors of class "Date"

Author(s)

Spencer Graves

See Also

[read.csv Dates3to1 dateCols](#)

Examples

```
##
## 1. Write a file to be read
##
cow0 <- data.frame(rec=1:3, startMonth=4:6, startDay=7:9,
  startYear=1971:1973, endMonth1=10:12, endDay1=13:15,
  endYear1=1974:1976, txt=letters[1:3])

write.csv(cow0, "cow0.csv", row.names=FALSE)
##
## 2. Read it
##
cow0. <- readDates3to1("cow0.csv")

# check
cow0x <- data.frame(rec=1:3, txt=letters[1:3],
  start=as.Date(c('1971-04-07', '1972-05-08', '1973-06-09')),
  end1=as.Date(c('1974-10-13', '1975-11-14', '1976-12-15')) )

all.equal(cow0., cow0x)
```

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 <code>financialCrisisFiles</code> .
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 sovereign 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 <code>read.xls</code> 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 <code>financialCrisisFiles</code> 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)

# optional tests if not CRAN
  if(!fda::CRAN()){
    bankingCrises.tst <- readFinancialCrisisFiles(tst)
    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')])

}
}

```

Description

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

Usage

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

Arguments

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

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	<i>Read the list of representatives in the United States House of Representatives</i>
-------------	---

Description

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

Usage

```
readUSHouse(url="http://www.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 row names = District (State + district, like "New York 3rd") and the following columns:

State	A factor identifying the state or territory the person represents
state	A factor giving the 2-letter US Postal Service abbreviation for the state or non-state governmental entity.
district	An integer identifying the district each person represents, with 0 for entities with only one representative, designated "At Large" for states, "Resident Commissioner" for Puerto Rico, and "Delegate" for the territories.
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	a character vector identifying the room number of the office
Phone	a character vector giving the phone number
Committees	a character vector giving the committee assignments of each representative
surname	a character vector giving the surname of each representative
givenName	a character vector with the 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	<i>Read the list of elected officials in the United States Senate</i>
--------------	---

Description

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

Usage

```
readUSsenate(url.=
  "https://www.senate.gov/general/contact_information/senators_cfm.xml",
  stateAbbreviations=Ecdat::USstateAbbreviations,
  fixNonStandard=subNonStandardNames, ...)
```

Arguments

`url.` Universal resource locator to be read and processed to obtain the desired list. (The current default was suggested by Julia Silge.)

`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 corresponding 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:

<code>State</code>	A factor identifying the state the person represents
<code>state</code>	A factor giving the 2-letter USPS code for the state represented
<code>Class</code>	an integer = 1, 2, or 3 for election in the 6-year cycle including 2008, 2010, or 2012, respectively.
<code>Name</code>	A character vector giving the name of each representative (in surname, given name format)
<code>party</code>	a factor identifying the party affiliation of each representative ("D", "R", "I" or ...)

address	character vector of physical addresses of offices in Washington, DC.
phone	character vector the phone number of the offices in Washington, DC.
email	a character vector giving the URL of their office contact form.
website	a character vector giving the URL of their web site in "senate.gov".
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.")
leadership_position	a character vector giving the names of the official leaders or blank.

Author(s)

Spencer Graves with help from Julia Silge

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.=
  "https://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 <code>strsplit(x, "\\[")</code>
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

`readUSstateAbbreviations` 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 <- unique(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)
```

 rgrep

Reverse grep

Description

Find which pattern matches `x`.

Usage

```
rgrep(pattern, x, ignore.case = FALSE, perl = FALSE,
  value = FALSE, fixed = FALSE, useBytes = FALSE,
  invert = FALSE)
```

Arguments

`pattern` a [character](#) vector of regular expressions to be matched to `x`
`x` a [character](#) string or vector for which a matching regular expression is desired.
`ignore.case`, `perl`, `value`, `fixed`, `useBytes`, `invert`
as for [grep](#)

Details

1. `np <- length(pattern)`
2. `g. <- rep(NA, np)`
3. `for(i in seq(length=np)) g.[i] <- (length(grep(pattern[i], x))>0)`
4. `return(which(g.))`

Value

an [integer](#) vector of indices of elements of `pattern` with a match in `x`.

Author(s)

Spencer Graves

See Also

[grep](#), [pmatch](#)

Examples

```
##  
## 1. return index  
##  
dd <- data.frame(a = gl(3,4), b = gl(4,1,12)) # balanced 2-way  
mm <- model.matrix(~ a + b, dd)  
  
b. <- rgrep(names(dd), colnames(mm)[5])  
# check  
  
all.equal(b., 2)  
  
##  
## 2. return value  
##  
bv <- rgrep(names(dd), colnames(mm)[5], value=TRUE)  
# check  
  
all.equal(bv, 'b')
```

sign *Sign function with zero option*

Description

sign returns a vector with the signs of the corresponding elements of *x*, being 1, zero, or -1 if the number is positive, zero or negative, respectively.

This generalizes the [sign](#) function in the base package to allow something other than 0 as the the "sign" of 0.

Usage

```
sign(x, zero=0L)
```

Arguments

x a numeric vector for which signs are desired
zero an [integer](#) value to be assigned for *x*==0.

Value

an [integer](#) vector of the same length as *x* assuming values 1, zero and -1, as discussed above.

See Also

[sign](#)

Examples

```
##  
## 1. default  
##  
sx <- sign((-2):2)  
  
# check  
  
all.equal(sx, base::sign((-2):2))  
  
##  
## 2. with zero = 1  
##  
s1 <- sign((-2):2, 1)  
  
# check  
  
all.equal(s1, rep(c(-1, 1), c(2,3)))
```

strsplit1	<i>Split the first field</i>
-----------	------------------------------

Description

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

Usage

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

Arguments

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

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. `split <- regexpr(split, x, ...)`
2. `Quote <- regexpr(Quote, x, ...)`
3. For any `(Quote < split)`, look for `Quote <- regexpr(Quote, substring(x, Quote+1))`, then look for `split <- regexpr(split, substring(x, Quote+Quote+1))`
4. `out <- list(substr(x, 1, split-1), substr(x, split+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](#)

replacement	a character to replace the substring 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. `x <- stringi::stri_trans_general(x, "Latin-ASCII")`
3. `nx <- length(x)`
4. `x. <- strsplit(x, "", ...)`
5. `for(ix in 1:nx)` find the first and last standardCharacters in `x.[ix]` and substitute replacement for everything in between.

NOTES:

** To find the elements of `x` that have changed, use either `subNonStandardCharacters(x) != x` or `grep(replacement, subNonStandardCharacters(x))`, where `replacement` is the replacement argument = "_" by default.

** On 13 May 2013 Jeff Newmiller at the University of California, Davis, wrote, 'I think it is a fools errand to think that you can automatically "normalize" arbitrary Unicode characters to an ASCII form that everyone will agree on.' (This was a reply on r-help@r-project.org, subject: "[R] Matching names with non- English characters".)

** On 2014-12-15 Ista Zahn suggested [stri_trans_general](#). (This was a reply on r-help@r-project.org, subject: "[R] Comparing Latin characters with and without accents?".)

Value

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

Author(s)

Spencer Graves with thanks to Jeff Newmiller, who described this as a "fool's errand", Milan Bouchet-Valat, who directed me to [iconv](#), and Ista Zahn, who suggested [stri_trans_general](#).

See Also

[sub](#), [strsplit](#), [grepNonStandardCharacters](#), [subNonStandardNames](#) [subNonStandardNames](#) [iconv](#) in the base package does some conversion, but is not consistent across platforms, at least using R 3.1.2 on 2015-01.25. [stri_trans_general](#) seems better.

Examples

```

##
## 1. 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 might be
## converted to something like Rub_n, _vila, and Jos_, with
## different software possibly mangling the names differently.
## (The standard checks for R packages in an English locale
## complains about non-ASCII characters, because they are
## not portable.)
##
nonstdNames <- c('Ra`l', 'Ra`', '`l', 'Torres, Raul',
                "Robert C. \\Bobby\\\\" , NA, '', ' ',
                '$12', '12%')

# confusion in character sets can create
# names like Names[2]
Name2 <- subNonStandardCharacters(nonstdNames)
str(Name2)

# check
Name2. <- c('Ra_l', 'Ra_', '_l', nonstdNames[4],
            'Robert C. "Bobby"', NA, '', ' ',
            '$12', '12%')
str(Name2.)

all.equal(Name2, Name2.)

##
## 2. Example from iconv
##
icx <- c("Ekstr\\xf8m", "J\\xf6reskog",
        "bi\\xdfchen Z\\xfcrcher")
icx2 <- subNonStandardCharacters(icx)

# check
icx. <- c('Ekstrom', 'Joreskog', 'bisschen Zurcher')

all.equal(icx2, icx.)

```

Description

```
sub(nonStandardNames[, 1], nonStandardNames[, 2], x)
```

Accented characters common in non-English languages often get mangled in different ways by different software. For example, the "e" in "Andre" may carry an accent that gets replaced by other characters by different software. This function first converts "Andr*" with "Andr_" for and character "*" not in standardCharacters. It then looks for "Andr_" in nonStandardNames. By default, it will find that and replace it with "Andre".

Usage

```
subNonStandardNames(x,
  standardCharacters=c(letters, LETTERS, ' ', '.', '?', '!',
    ',', 0:9, '/', '*', '$', '%', '\\', "\\", '-', '+', '&', '_', ';',
    '(', ')', '[', ']', '\\n'),
  replacement='_',
  gsubList=list(list(pattern='\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\',
    replacement='\\')),
  removeSecondLine=TRUE,
  nonStandardNames=Ecdat::nonEnglishNames,
  namesNotFound="attr.replacement", ...)
```

Arguments

- x** character vector or matrix or a data.frame of character vectors 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 element of nonStandardNames[, 2]
- namesNotFound** character vector describing how to treat substitutions not found in nonStandardNames[, 1]:
- "attr.replacement": Return an attribute "namesNotFound" with `grep(replacement, subNonStandardCharacters(x, ...))` if any.
 - "attr.notFound": Return an attribute "namesNotFound" with `x[!subNonStandardCharacters(x, ...)]` if any.
 - "print": Print the elements of x "notFound" per either "attr.replacement" or "attr.notFound", as requested.
 - "": Do not report any "notFound" elements of x.

NOTE: x = "_" will be identified by "attr.replacement" but not by "attr.notfound" assuming the default value for replacement.

Details

1. removeSecondLine
2. `x. <- subNonStandardCharacters(x, standardCharacters, replacement, ...)`
3. Loop over all rows of `nonStandardNames` substituting anything matching `nonStandardNames[i, 1]` with `nonStandardNames[i, 2]`.
4. Eliminate leading and trailing blanks.
5. `if(is.matrix(x))` return a matrix; `if(is.data.frame(x))` return a `data.frame(..., stringsAsFactors=FALSE)`

NOTE: On 13 May 2013 Jeff Newmiller at the University of California, Davis, wrote, 'I think it is a fools errand to think that you can automatically "normalize" arbitrary Unicode characters to an ASCII form that everyone will agree on.' (This was a reply on `r-help@r-project.org`, subject: "[R] Matching names with non- English characters".) Doubtless someone has software to do a better job of this than what this function does, but I've so far been unable to find it in R. If you know of a better solution to this problem, I'd be pleased to hear from you. Spencer Graves

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. If a `secondLine` is found on any elements, it is returned as a "secondLine" attribute. If any names with `nonStandardCharacters` are not found in `nonStandardNames[, 1]`, they are identified in an optional attribute per the `namesNotFound` argument.

Author(s)

Spencer Graves

See Also

[sub nonEnglishNames](#) [subNonStandardCharacters](#) [stripBlanks](#)

Examples

```
##
## 1. Example
##
tstSNSN <- c('Raul', 'Ra`l', 'Torres,Raul', 'Torres, Ra`l',
            "Robert C. \\Bobby\\\\" , 'Ed \n --Vacancy', '', ' ')

# confusion in character sets can create
# names like Names[2]
##
## 2. subNonStandardNames(vector)
##

SNS2 <- subNonStandardNames(tstSNSN)
SNS2

# check
SNS2. <- c('Raul', 'Raul', 'Torres,Raul', 'Torres, Raul',
```

```
      'Robert C. "Bobby"', 'Ed', '', '')
attr(SNS2., 'secondLine') <- c(rep(NA, 5), ' --Vacancy', NA, NA)

all.equal(SNS2, SNS2.)

##
## 3. subNonStandardNames(matrix)
##
tstmat <- parseName(tstSNSN, surnameFirst=TRUE)
submat <- subNonStandardNames(tstmat)

# check
SNSmat <- parseName(SNS2., surnameFirst=TRUE)

all.equal(submat, SNSmat)

##
## 4. subNonStandardNames(data.frame)
##
tstdf <- as.data.frame(tstmat)
subdf <- subNonStandardNames(tstdf)

# check
SNSdf <- as.data.frame(SNSmat, stringsAsFactors=FALSE)

all.equal(subdf, SNSdf)

##
## 5. namesNotFound
##
noSub <- subNonStandardNames('xx_x')

# check
noSub. <- 'xx_x'
attr(noSub., 'namesNotFound') <- 'xx_x'

all.equal(noSub, noSub.)
```

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`), precede each test with "ping url[i]". NOTE: Some Internet Service Providers seem to block some attempts to use "ping" or return fraudulent 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, tzzone='GMT', ping=FALSE, ...)
```

Arguments

<code>urls</code>	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.
<code>file</code> .	Name of a CSV file to which to write the results. If the file already exists, new results are appended to it.
<code>n</code>	number of times to repeat the cycle testing each member of <code>urls</code> .
<code>maxFail</code>	max tests for a continually failing URL. This is designed to make it relatively easy to 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.
<code>warn</code>	warn argument to pass to Ping .
<code>tzzone</code>	Time zone for Time. Defaults to GMT (UTC). <code>tzzone=NULL</code> will use the current locale.
<code>ping</code>	logical: TRUE to include Ping , FALSE otherwise.
<code>...</code>	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'))
```

<code>trimImage</code>	<i>Trim zero rows or columns from an object of class Image.</i>
------------------------	---

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

<code>x</code>	a numeric matrix or 3-dimensional array or an object with subscripting defined so it acts like such.
<code>max2trim</code>	a single number indicating the max absolute numeric value to trim.
<code>na.rm</code>	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.
<code>returnIndices2Keep</code>	if TRUE, return a list with 2 integer vectors giving row and column indices to use in selecting the desired subset of <code>x</code> . This allows an array <code>y</code> to be trimmed to match <code>x</code> . If FALSE, return the desired trimmed version of <code>x</code> . If this is a list with two two integer vectors, use them to trim <code>x</code> .
<code>...</code>	Optional arguments; not currently used.

Details

1. Check arguments: $2 \leq \text{length}(\text{dim}(x)) \leq 3$? $\text{is.logical}(\text{na.rm})$? $\text{returnIndices2Keep} = \text{logical}$ or list of 2 integer vectors, all the same sign, not exceeding $\text{dim}(x)$?
2. $\text{if}(\text{is.list}(\text{returnIndices2Keep}))$ check that $\text{returnIndices2Keep}$ is a list with 2 integer vectors, all the same sign, not exceeding $\text{dim}(x)$. If yes, return x appropriately subsetted.
3. $\text{if}(\text{!is.logical}(\text{returnIndices2Keep}))$ throw an error message.
4. Compute indices2Keep .
5. $\text{If}(\text{returnIndices2Keep})$ return (indices2Keep) else return x appropriately subsetted.

Value

$\text{if}(\text{returnIndices2Keep} == \text{TRUE})$ 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 $x \leq \text{truncmin}$, 1 for $\text{truncmax} < x$, and in between is as follows:

$$(\text{pdist}(x, \dots) - \text{pdist}(\text{truncmin}, \dots)) / (\text{pdist}(\text{truncmax}, \dots) - \text{pdist}(\text{truncmin}, \dots))$$

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 $\text{length}(n) > 1$, 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 ptruncdist is pnorm, etc.
truncmin, truncmax	lower and upper truncation points, respectively.

Details

NOTE: Truncation 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.

By contrast, with a truncated distribution, events below truncmin and above truncmax may exist but are not observed. Thus, it's not known how many events occur outside the given range, truncmin to truncmax, if any. Given data believed to come from a truncated distribution, estimating the

parameters provide a means of estimating the number of unobserved events, assuming a particular form for their distribution.

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))

# 1.5. Poisson without 0.
dPois0.9 <-dtruncdist(0:9, lambda=1, dist='pois', truncmin=0)

# check
dP0.9 <- c(0, dpois(1:9, lambda=1)/ppois(0, lambda=1, lower.tail=FALSE))

all.equal(dPois0.9, dP0.9)

##
## 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
qlx101 <- qtruncdist(log(seq(0, 1, .2)), log(100), 2,
                    log.p=TRUE, dist='lnorm', truncmin=500)

# check

all.equal(qlx10, qlx101)

##
## 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

NOTE: THIS FUNCTION IS CURRENTLY UNAVAILABLE: A CALL TO "RCurl::getURL" THAT RETURNED INFO ON ALL MEMBERS OF THE US HOUSE IN 2013 AND 2016 ONLY RETURNS 222 CHARACTERS AS OF 2018-01-11.

If and when a fix for this is found, this function may again:

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:

Chamber	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., 'D', 'R', 'I'.
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(FALSE){
#NOTE: THIS FUNCTION IS CURRENTLY UNAVAILABLE: A CALL TO
#"RCurl::getURL" THAT RETURNED INFO ON ALL MEMBERS OF THE
#US HOUSE IN 2013 AND 2016 ONLY RETURNS 222 CHARACTERS
#AS OF 2018-01-11.

# If and when a fix for this is found, this function may again
# work as it did 2013-2016.

if(!fda::CRAN()){
house <- readUSHouse()

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

USsenateClass

Election Class given state and surname of a US Senator

Description

For all individuals in `x` with `houseSenate == "Senate"`, look up their state and surname in the reference table `senate` and return their Class. For individuals 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 2018, 2020, and 2022), 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. In particular, 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 would 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 <i>x</i> in which the members of the US Senate can be identified by <code>grep(senatePattern, x[, Office])</code> .
state	Standard 2-letter abbreviation for the state of the US
surname	the name of a column of <i>x</i> containing the surname
district	name of a column of <i>x</i> containing the number of the district in the US House. For states with only one representative, this may be 0.
senatePattern	a regular expression for identifying the senators from <code>x[, Office]</code> .

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 <code>Office == 'house'</code> . If <code>Office == 'senate'</code> , then TRUE if <code>state:surname</code> found in <code>senate</code> and FALSE otherwise.
District	a character vector containing the desired Class for all US Senators found in <code>senate</code> or a guess at the Class for non-incumbents. For members of the House, this returned the previous content of <code>x[[District]]</code> .

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

*Topic **IO**

- financialCrisisFiles, 34
- Ping, 80
- read.testURLs, 95
- read.transpose, 96
- readCookPVI, 98
- readDates3to1, 100
- readFinancialCrisisFiles, 101
- readNIPA, 103
- readUSHouse, 105
- readUSSenate, 107
- readUSstateAbbreviations, 108
- testURLs, 119
- USHouse.senate, 130

*Topic **aplot**

- Arrows, 3
- canbeNumeric, 15
- rgrep, 110

*Topic **datasets**

- readFinancialCrisisFiles, 101
- readUSstateAbbreviations, 108
- USHouse.senate, 130

*Topic **distribution**

- truncdist, 125

*Topic **hplot**

- rasterImageAdj, 93

*Topic **htest**

- confint.var, 22

*Topic **manip**

- as.Date1970, 4
- asNumericDF, 5
- BoxCox, 8
- camelParse, 14
- checkNames, 16
- classIndex, 18
- compareLengths, 20
- countByYear, 24
- countsByYear, 26
- createMessage, 27

- createX2matchY, 29
- Date3to1, 30
- dateCols, 31
- Dates3to1, 33
- getElement2, 36
- grepNonStandardCharacters, 38
- Interp, 40
- interpChar, 45
- interpPairs, 49
- match.data.frame, 59
- matchName, 61
- matchQuote, 64
- mergeUSHouse.senate, 66
- mergeVote, 68
- missing0, 70
- nchar0, 71
- Newdata, 72
- parseCommas, 75
- parseDollars, 76
- parseName, 77
- pmatch2, 82
- recode2, 109
- sign, 112
- strsplit1, 113
- subNonStandardCharacters, 114
- subNonStandardNames, 116
- trimImage, 122
- USSenateClass, 131

*Topic **multivariate**

- logVarCor, 57

*Topic **plot**

- qqnorm2, 84
- qqnorm2s, 88
- qqnorm2t, 91
- whichAeqB, 133

- agrep, 60
- approx, 43
- arrow, 3
- Arrows, 3

- arrows, [3](#)
- as.character, [37](#)
- as.Date, [4](#), [6](#), [7](#), [15](#)
- as.Date1970, [4](#)
- as.numeric, [7](#), [15](#), [75](#), [77](#)
- as.POSIXct, [6](#), [7](#), [15](#)
- as.POSIXct1970, [4](#)
- as.raster, [93](#)
- asNumericChar (asNumericDF), [5](#)
- asNumericDF, [5](#)
- attributes, [9](#)

- BoxCox, [8](#)
- boxCox, [11](#)
- boxcox, [11](#)
- boxcox.drc, [11](#)
- boxcoxCensored, [11](#)

- call, [50](#)
- camelParse, [14](#), [107](#), [108](#)
- canbeNumeric, [15](#), [72](#), [73](#)
- character, [6](#), [43](#), [85](#), [88](#), [89](#), [91](#), [92](#), [106](#), [111](#)
- checkNames, [16](#)
- classify, [60](#)
- classIndex, [18](#), [43](#), [47](#)
- colSums, [26](#)
- compareLengths, [20](#), [41](#), [46](#), [52](#)
- complex, [43](#)
- confint.sd (confint.var), [22](#)
- confint.var, [22](#)
- cor.test, [23](#)
- coredata, [40](#), [41](#)
- countByYear, [24](#)
- countsByYear, [26](#)
- cov2cor, [58](#)
- createMessage, [27](#)
- createX2matchY, [29](#)

- data.frame, [5](#), [7](#), [26](#), [30](#), [33](#), [52](#), [61](#), [66–69](#), [72](#), [73](#), [75](#), [84](#), [88](#), [89](#), [91](#), [96](#), [98–100](#), [107](#), [130–132](#)
- Date3to1, [30](#), [32](#), [34](#)
- dateCols, [31](#), [31](#), [34](#), [100](#), [101](#)
- Dates3to1, [33](#), [101](#)
- delimMatch, [65](#)
- deparse, [21](#), [27](#)
- diag, [58](#)
- dim, [110](#)
- Distributions, [126](#)

- dtruncdist (truncdist), [125](#)

- enquote, [52](#)
- eval, [37](#), [52](#)

- factor, [6](#), [7](#), [15](#), [91](#), [106](#)
- financialCrisisFiles, [34](#), [101](#), [102](#)
- function, [50](#)

- getElement, [37](#)
- getElement2, [36](#)
- getURL, [106](#), [108](#), [109](#), [120](#), [121](#)
- grep, [6](#), [31–33](#), [39](#), [51](#), [52](#), [60](#), [83](#), [111](#), [113](#)
- grepNonStandardCharacters, [38](#), [115](#)
- gsub, [6](#), [7](#), [65](#), [75–77](#)

- iconv, [115](#)
- identity, [78](#), [79](#), [105](#)
- index2class (classIndex), [18](#)
- integer, [43](#), [72](#), [106](#), [111](#), [112](#)
- Interp, [40](#)
- InterpChar (Interp), [40](#)
- interpChar, [19](#), [21](#), [45](#), [49–52](#)
- InterpChkArgs (Interp), [40](#)
- InterpNum (Interp), [40](#)
- interpPairs, [29](#), [43](#), [47](#), [49](#), [134](#)
- invBoxCox (BoxCox), [8](#)
- is.na, [60](#)
- is.null, [73](#)

- join, [60](#)

- legend, [89](#), [92](#)
- length, [41](#), [42](#)
- levels, [72](#), [73](#)
- lines, [86](#)
- lines.qqnorm2 (qqnorm2), [84](#)
- log, [58](#)
- logical, [15](#), [43](#), [70](#), [72](#)
- logVarCor, [57](#)
- lower.tri, [57](#), [58](#)

- make.names, [16](#), [17](#), [51](#), [109](#)
- match, [60](#), [83](#)
- match.data.frame, [59](#)
- match_df, [60](#)
- matchName, [61](#), [83](#)
- matchName1 (matchName), [61](#)
- matchQuote, [64](#)
- matrix, [26](#), [72](#), [104](#)

- median, [72, 73](#)
- merge, [67](#)
- mergeUShouse.senate, [66, 69](#)
- mergeVote, [68](#)
- missing, [41, 57, 59, 68, 70](#)
- missing0, [70](#)
- mode, [15](#)
- name, [37](#)
- names, [16, 17, 51](#)
- nchar, [28, 40, 42, 71](#)
- nchar0, [71](#)
- Newdata, [72](#)
- nonEnglishNames, [118](#)
- Normal, [126](#)
- numeric, [43](#)
- options, [16, 17, 81](#)
- par, [93](#)
- parseCommas, [75, 76, 77](#)
- parseDollars, [75, 76](#)
- parseName, [61, 62, 77, 106, 108](#)
- paste, [27, 28, 60, 65](#)
- pdLogChol, [58](#)
- Ping, [80, 120, 121](#)
- plot, [84, 86, 88, 90, 92](#)
- plot.qqnorm2 (qqnorm2), [84](#)
- plot.qqnorm2s (qqnorm2s), [88](#)
- pmatch, [83, 111](#)
- pmatch2, [82](#)
- points, [85, 86, 89, 92](#)
- points.qqnorm2 (qqnorm2), [84](#)
- predict, [72](#)
- predict.lm, [73](#)
- ptruncdist (truncdist), [125](#)
- qqnorm, [85, 86, 89, 91](#)
- qqnorm2, [84, 89–92](#)
- qqnorm2s, [84, 86, 88, 91, 92](#)
- qqnorm2t, [84, 86, 88, 91](#)
- qtruncdist (truncdist), [125](#)
- quine, [11](#)
- Quotes, [7](#)
- range, [72, 73](#)
- rasterImage, [93, 94](#)
- rasterImageAdj, [52, 93](#)
- raw, [43](#)
- rbind, [106](#)
- read.csv, [33, 95, 96, 100, 101](#)
- read.table, [97, 105](#)
- read.testURLs, [95](#)
- read.transpose, [96, 104](#)
- read.xls, [6, 35, 36, 102](#)
- read.xlsx, [6, 7](#)
- readCookPVI, [98, 106](#)
- readDates3to1, [100](#)
- readFinancialCrisisFiles, [101](#)
- readHTMLTable, [106, 108, 109](#)
- readLines, [97, 105](#)
- readNIPA, [103](#)
- readUShouse, [98, 99, 105, 108, 130, 131](#)
- readUSsenate, [98, 99, 106, 107, 130–132](#)
- readUSstateAbbreviations, [106, 108](#)
- recode2, [109](#)
- regexpr, [17, 39](#)
- rep, [42, 46](#)
- rgrep, [110](#)
- row.match, [60](#)
- rownames, [110](#)
- rtruncdist (truncdist), [125](#)
- scan, [7](#)
- seq, [15](#)
- showNonASCII, [39](#)
- sign, [9, 112, 112](#)
- sort, [73](#)
- stri_trans_general, [115](#)
- stripBlanks, [7, 118](#)
- strsplit, [14, 60, 79, 97, 105, 113, 115](#)
- strsplit1, [65, 113](#)
- sub, [6, 17, 32, 51, 115, 118](#)
- subNonStandardCharacters, [39, 108, 114, 117, 118](#)
- subNonStandardNames, [61, 62, 79, 106, 108, 115, 116](#)
- substr, [28](#)
- substring, [46, 113](#)
- sum, [25](#)
- system, [81](#)
- testURLs, [96, 119](#)
- tolower, [68](#)
- trim, [123](#)
- trimImage, [122](#)
- truncdist, [125](#)
- try, [121](#)

unique, [72](#), [73](#)

USFinanceIndustry, [104](#)

USHouse.senate, [67](#), [106](#), [108](#), [130](#)

USsenateClass, [131](#)

USstateAbbreviations, [109](#)

VarCI, [23](#)

warning, [32](#)

whichAeqB, [133](#)

xyinch, [94](#)

zoo, [40](#), [41](#)