Package 'seas'

July 2, 2014

Type Package

Version 0.4-3

Date 2014-02-26

Title Seasonal analysis and graphics, especially for climatology

Author Mike Toews

Maintainer Mike Toews <mwtoews@gmail.com>

Depends R(>= 2.10.0)

Suggests MASS

Description Capable of deriving seasonal statistics, such as ``normals", and
analysis of seasonal data, such as departures. This package also has
graphics capabilities for representing seasonal data, including boxplots for
seasonal parameters, and bars for summed normals. There are many specific
functions related to climatology, including precipitation normals,temperature normals, cumulative precipitation departures and precipitation
interarrivals. However, this package is designed to represent any
time-varying parameter with a discernible seasonal signal, such as found
in hydrology and ecology.

License GPL (>= 2)

URL http://github.com/mwtoews/seas

BugReports http://github.com/mwtoews/seas/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-02-26 10:20:40

R topics documented:

`seas-package` *Seasonal statistics: the 'seas' package for R*

Description

The `seas` package for the R programming environment is capable of conveying descriptive statistics and graphics for seasonal variables, as found in climatology, hydrology and ecology. Seasonal variables can be continuous (i.e., temperature) or discontinuous (i.e., precipitation). An annum can be partitioned into many arbitrary divisions, or seasonal components, such as *by month* or into other fixed intervals. Boxplots are used to describe the seasonal distributions of continuous variables. Discontinuous variables need to be summed over time to smooth the irregularities before the variable can evaluated and visualized. Statistics, such as precipitation normals, may be derived

from the summed variables, using the mean or median methods. Other tools and utilities provided in the package can calculate precipitation interarrivals, cumulative precipitation departures, find changes between two normals, and import data from archive formats.

To get started, try some examples, such as `seas.sum`, `seas.norm`. There will be more help added here someday! In the mean-time, please explore the various help pages and try the examples. Contact me for basic help and/or suggestions!

Author(s)

Mike Toews

References

Toews, M.W., Whitfield, P.H., and Allen, D.M., Seasonal statistics: The 'seas' package for R, Computers & Geosciences (2007), doi:10.1016/j.cageo.2006.11.011

Examples

```
# Show a list of changes to the package:
file.show(system.file("ChangeLog", package="seas"))
```

`A1128551.DLY` *MSC daily climate data file (DLY archive format) and instructions for Canadian Daily Climate Data CD-ROMs for analysis*

Description

Meteorological Service of Canada daily climate data (DLY archive format) from Vernon. This document also describes how to obtain data from the Canadian Daily Climate Data CD-ROMs for analysis in **seas**.

Format

MSC DLY archive format (4-digit year).

Details

The sample file name is '`A1128551.DLY`', which contains daily climate data from Vernon, British Columbia. Load this file using `read.msc`.

This file was created using the instructions below, with the addition of renaming the file extension from '`*.ALL`' to '`*.DLY`'.

How to obtain Canadian Daily Climate Data

Two CDCD CD-ROMs are currently available for free download, which have data from 11,216 locations throughout Canada.

This procedure shows how to extract the data using 'CDEX.EXE', which requires a DOS environment. There is, however, an alternative Python module, which can batch extract data from the CD-ROMs. If you are using a non-Microsoft platform, you could try 'DOSBox' to emulate the DOS environment (tested on Debian and Mac OS X; hint: mount the CD-ROM drive by using `-t cdrom` option).

To extract data from the CD-ROM:

1. Insert CD-ROM, and run 'CDEX.EXE' (or double-click it)
2. Select a 'district'; press 'enter'
3. Select a 'station'; press 'enter'
4. Select 'Elements to Convert', and select the desired fields using the 'space bar'; press 'enter'
5. Change 'Drive/directory of output files' to a convenient location, for example 'C:\TEMP'
6. Press 'F10' to extract the data (the name of the file is the 7-digit alphanumeric station number, followed by a '.ALL' extension)
7. Repeat these steps for each meteorological station desired (if there are more).

Multiple stations can be imported and combined before or after importing into R. Multiple files can be concatenated into one from the system shell (e.g. DOS: `COPY *.ALL new.dly`, or UNIX: `cat *.ALL > new.dly`). This cleans up the R workspace by only using one object to refer to several stations. Stations can be referred to functions in **seas** using their IDs.

To import the archive file into R:

1. Start R; type `library(seas)`
2. Import using `dat <- read.msc("/temp/C1161661.ALL")` (note that R uses forward slashes for directories, but you could alternatively type "C:\\TEMP\\C1161661.ALL" on a Microsoft-based platform to 'escape' the back slash characters)

To export the data from R in a more convenient format for other programs, use `write.csv(dat,"out.csv")`; MS Excel users may want to turn NA values into the format recognized by Excel, so modify the expression to `write.csv(dat,"out.csv",na="#N/A")`.

Author(s)

Mike Toews

Source

Data provided by the Meteorological Service of Canada (http://www.msc.ec.gc.ca/), with permission.

This data may only be reproduced for personal use; any other reproduction is permitted only with the written consent of Environment Canada (http://climate.weatheroffice.ec.gc.ca/contacts/).

References

http://climate.weatheroffice.gc.ca/prods_servs/documentation_index_e.html Technical Documentation - Documentation for the Digital Archive of Canadian Climatological Data (Surface) Identified By Element

http://climate.weatheroffice.gc.ca/prods_servs/index_e.html#cdcd CDCD CD-ROM download location

http://dosbox.sourceforge.net for emulating DOS on non-Microsoft platforms

http://www.intevation.de/~bernhard/archiv/uwm/canadian_climate_cdformat/ an alternative method of extracting data from the CDCD CD-ROMs using a Python module by Bernhard Reiter

See Also

read.msc

Examples

```
fname <- system.file("extdata", "A1128551.DLY", package="seas")
print(fname)
dat <- read.msc(fname)
head(dat)
str(dat)

seas.temp.plot(dat)
year.plot(dat)
```

---

change *Find seasonal and annual changes between two data sets*

Description

Find seasonal and annual changes between two data sets; relative and absolute changes are found between the central tendency and spread of each seasonal state.

Usage

```
change(x1, x2, var1, var2 = var1, width = "mon",
       cent = "mean", sprd = "sd", disc = FALSE, inter = FALSE,
       p.cut = 0.3, start.day = 1, calendar)
```

Arguments

x1	a data.frame of seasonal data
x2	a second data.frame of seasonal data
var1	a variable in x1
var2	a variable in x2

width	the width of the bins, see mkseas for more details; this will change the sample-sizes between x1 and x2, which can affect the changes detected
cent	a function to find a central tendency; usually this is mean, however median or other functions can be used too
sprd	a function to find a spread around a central tendency; usually this will be sd (standard-deviation), however mad or other functions can be used too
disc	if the data are discontinuous, the seas.sum objects are created for var1/var2 to determine the changes; this is ideal for precipitation, and other sparsely distributed variables
inter	interarrivals are calculated, and changes are found between *wet* and *dry* series
p.cut	cut-off for wet/dry; see interarrival
start.day	starting day
calendar	calendar; if not specified it will try to read this from the attributes, otherwise it is assumed to be a proleptic Gregorian calendar; see year.length

Details

This function is useful for finding changes between different states of seasonal data. Here, a *state* represents how seasonal data behave statistically at either a *time* or *place*. The stability of a state depends on both the variance throughout each portion of the season, as well as the number of years of observations.

For instance, seasonal and annual changes in climate can be detected in climate data series, by comparing the normals from two time periods.

Value

Returns a complex list of relative and absolute (if applicable) changes of var1/var2 between x1 and x2.

Seasonal and annual changes are identified independently of each other; where annual changes have a ann prefix.

Relative changes are not found if x$var has values less than 0, such as Temperature measured in degrees C or F.

Author(s)

Mike Toews

See Also

dathomog, lars

Examples

```
data(mscdata)
dat1 <- mksub(mscdata, id=1108447, start=1975, end=1984)
dat2 <- mksub(mscdata, id=1108447, start=1985, end=1995)

# A few plot functions to make thing easy
plot.ch <- function(x, main, h, col) {
  main <- paste(main, "between 1975-1984 and 1985-1994", sep="\n")
  barplot(x, main=main)
  abline(h=c(0, h), col=c(1, col), lty=c(1, 2))
}
plot.abs <- function(x, col="red", abs="abs", ann.abs="ann.abs") {
  main <- sprintf("Absolute change in %s", x$long.name[[1]])
  plot.ch(x[[abs]], main, x[[ann.abs]], col)
}
plot.rel <- function(x, col="orange", rel="rel", ann.rel="ann.rel") {
  main <- sprintf("Relative change in %s", x$long.name[[1]])
  plot.ch(x[[rel]], main, x[[ann.rel]], col)
}
plot.std <- function(x, col="purple") {
  main <- sprintf("Relative change in the\nstandard deviation of %s",
                  x$long.name[[1]])
  plot.ch(x$sprd.rel, main, x$ann.sprd.rel, col)
}

# Minimum temperature
ch <- change(dat1, dat2, "t_min")
str(ch)
plot.abs(ch)
plot.std(ch)
# Cannot do ch$rel ; since div/0!

# Precipitation
ch2 <- change(dat1, dat2, "precip", width="DJF", disc=TRUE)
plot.abs(ch2, "blue")
plot.rel(ch2, "purple")
plot.std(ch2)
```

`conv365toGregorian` *Converts a data.frame using a 365-day calendar to a Gregorian calendar*

Description

Converts a `data.frame` with a 365-day calendar to a proleptic Gregorian calendar, repeating data from December 30th on a leap year to the remaining and missing December 31st.

Usage

```
conv365toGregorian(x)
```

Arguments

x	a data.frame with a date column

Details

This function may be expanded in the future to be more flexible.

Value

Returns a data.frame with Gregorian calendar dates.

Author(s)

Mike Toews

---

dathomog *Homogenize daily data sets*

Description

Homogenizes daily data from two data sets into one data set; optionally show cross-plots to examine how well correlated that data sets are.

Usage

```
dathomog(x1, x2, by = "date", plot = FALSE)
```

Arguments

x1	a data.frame of seasonal data; 1st selection
x2	a data.frame or seasonal data; 2nd selection
by	name of common column, usually 'date', which is of class Date
plot	logical; produce cross-plots and correlation statistics of the variables between the two data sets

Details

Data from x1 has priority over x2. Where data from x1 is either NA or missing (outside of time range), data from x2 will be used (if available). Otherwise data form x1 will be used directly. Variables will be homogenized where their names are identical, found using names.

The cross-plots of the data are shown only for interest. They show useful correlation statistics, and a best-fit line using perpendicular offsets (which are preferred in this case over traditional linear regression). At some point, the equations for this line may be used to adjust the values from x2, however this can always be done externally to this function by pre-processing x2.

Value

Returns a `data.frame` of seasonal data required by most functions in **seas**. Variable names of the structure are found by a `union` of the names of `x1` and `x2`.

Warning

Weather stations should be sufficiently close enough to approximate the same weather. This distance depends on the spatial distance and local climatology.

Author(s)

Mike Toews

References

http://mathworld.wolfram.com/LeastSquaresFittingPerpendicularOffsets.html

Examples

```
data(mscdata)
dat1 <- mksub(mscdata, id=2100630)
dat2 <- mksub(mscdata, id=1108447)
year.plot(dat1)
year.plot(dat2)
newdata <- dathomog(dat1, dat2)
year.plot(newdata)
message(paste(c("This is a rather poor example, since the",
                "two stations are nowhere near each other"),
                collapse="\n"))
```

---

getstnname *Get station name*

Description

Retrieves the full name from `mscstn` using an ID

Usage

```
getstnname(id)
```

Arguments

`id`	`numeric` or `character` ID array

Details

This function simply converts the ID used in climate data frames into a meaningful name using `mscstn`. Presently it is useful only for Meteorological Service of Canada weather stations in BC, AB and YT, however `getstnname` can be overridden by another (similar) function and data object for other regions.

Value

Returns the station name(s). If the ID does not exist, returns `NULL`.

Author(s)

Mike Toews

See Also

`mscstn`, `mscdata`, `.seastitle`

Examples

```
data(mscdata)

mscdata$id[1]
getstnname(mscdata$id[1])

ids <- levels(mscdata$id)
data.frame(id=I(ids), name=getstnname(ids))
```

---

hidden *Return title and properties for seasonal graphs*

Description

Return title, x- and y-axis labels for seasonal graphs. Also, draw the month grid.

Usage

```
.seasxlab(width, start.day)
.seasylab(var, long.name = NULL, units = NULL)
.seastitle(main = NULL, id = NULL, name = NULL,
           orig = NULL, fun = NULL, range = NA)
.seasmonthgrid(width, days, start = 1, rep = 0, start.day = 1, month.label)
```

Arguments

orig	original object name, which is used if no other name can be found from id or name
var	original variable name in names(orig)
width	size of bin; see mkseas
start.day	the starting day of annum for the first bin; as either a Date or integer of day of the year
main	main title of plot; overrides any other title, but appends year range if show.range=TRUE
long.name	a long name for var, used for labelling y-axis
units	units of var, used for labelling y-axis
id	station ID, which is used to fetch a station name using getstnname
name	a name, which is used for labels
fun	function, if applicable
range	year range; c(start.year, end.year)
days	also known as bin.lengths, which represents the maximum number of days expected in each bin for a complete annum
start	the starting bin number
rep	the number of repeated bins
month.label	logical; put month name labels on grid

Details

These functions are intended for producing the graphics, and do not need to be used directly.

The month grid is drawn by .seasmonthgrid, and can be fine-tuned by setting some options in the R environment, found in seas.month.grid. This list, for instance, has len to adjust the length of each month label, and col for the colour of the lines. See SeasOpts for all available options, and instructions on how to change them.

The main and variable/unit label formating can also be customized by setting other options, documented in SeasOpts.

Value

.seasxlab, .seasylab and .seastitle return a character label intended for plots.

Author(s)

Mike Toews

See Also

getstnname, SeasOpts

Examples

```
setSeasOpts()

.seasxlab(11, 1)
.seasxlab("mon", 1)

# Not starting on January 1st
.seasxlab(11, 120)

# Labelled according to month (and possibly day)
getOption("seas.label")$month
.seasxlab("mon", as.Date("2000-08-01"))
getOption("seas.label")$monthday
.seasxlab(365/20, as.Date("2000-08-15"))
```

`image.seas.sum` *Show a seasonal sum data object*

Description

Graphically display a seasonal sum object, as well as the method of solution of the median/quantile "normal"

Usage

```
## S3 method for class 'seas.sum'
image(x, var, norm = "days", start = 1, rep = 0, zlim, alim,
    palette = colorRampPalette(c("white", "blue"))(64),
    year.filter, power, contour = TRUE, show.median, main, ...)
```

Arguments

`x`	a `seas.sum` object
`var`	the desired variable to show, otherwise will use the prime variable, defined in `x`
`norm`	variable to normalize by, usually `"days"`, to produce *unit/day*
`start`	starting bin number; e.g., for monthly sums, if `start=5`, the plot will start on "May" at the left-hand side; `show.median` cannot be produced if `start` is greater than one, since the annual sums (row-wise) would be a mix of different annums
`rep`	repetition of the bins (columns)
`zlim`	range of normalized values displayed; this can be either a single number for the maximum (minimum set to zero), or a `c(min, max)` range with a defined minimum
`alim`	if `show.median`, this is the range for the annual sums; this can either be a singe number for the maximum (minimum set to zero, or a `c(min, max)` range with a defined minimum
`palette`	colours for `image`; the use of `colorRampPalette` is recommended

year.filter	specifies the annual seasons to display
power	this transforms the normalized values for the colours to a power (^), such as 0.5 for square-root (sqrt), or others; this can help improve the contrast in the display of data, but the quantities displayed in the colour-bar and contours remain untransformed
contour	logical; show contours in lower left-hand plot
show.median	logical; show how the median calculation is achieved graphically (computationally it is done using a secant method); see seas.norm for more information on this method; this can only be shown if the annums (rows) are complete, so start must be 1, and rep must be 0 (otherwise the row-wise sums would not be the annual sums)
main	main title for plot, otherwise it will automatically be generated; NA suppresses a title, and automatically adjusts the device margins
...	ignored

Details

This is a graphical representation of a seas.sum object, and is far more informative than a traditional precipitation "normal" (i.e., precip.norm or precip.norm)

If norm = "days" and show.median = TRUE (default), the seasonal sums appear in right-hand frames. Horizontal and vertical lines indicate a 'normal' from the image, whereby the sum of the quantile is equal to the median of the annual amount. This numerical solution is found using seas.norm.

Author(s)

Mike Toews

See Also

seas.sum, seas.norm

See SeasOpts to modify other aspects of the plot

Examples

```
data(mscdata)
dat <- mksub(mscdata, id=1108447)

dat.ss <- seas.sum(dat, width="mon")
image(dat.ss)

image(dat.ss, contour=FALSE)

image(dat.ss, norm="active", start=6, rep=5)

# different start day (not Jan 1st)
dat2.ss <- seas.sum(dat, start.day=as.Date("2001-08-01"))
image(dat2.ss)
```

```
image(dat2.ss, power=2)
image(dat2.ss, palette=rainbow(64), main=NA)  # no title
image(dat2.ss, palette=colorRampPalette(c("white", "darkgreen"))(16))
image(dat2.ss, "snow")
image(dat2.ss, "snow", power=0.5)

# growing degree days for 10 degC
dat$gdd10 <- dat$t_mean - 10
dat$gdd10[dat$gdd10 < 0] <- 0
attr(dat$gdd10,"long.name") <- "growing degree days"
dat3.ss <- seas.sum(dat, var="gdd10")
image(dat3.ss, "gdd10", palette=colorRampPalette(c("white", "red"))(64))
```

`interarrival` *Calculate the interarrivals between and within precipitation events*

Description

Calculate the interarrivals (or spell periods), which are the number of days between precipitation events (dry days), and the number of days of continuous precipitation (wet days).

Usage

```
interarrival(x, var = "precip", p.cut = 0.3, inv = FALSE)
```

Arguments

`x`	a `data.frame` with `Date` and `var` columns of data; `x` can also have `id` or `name` attributes
`var`	a variable on to which the interarrivals are calculated; default is `"precip"`
`p.cut`	days with precipitation values greater than `p.cut` are considered to be *wet* days, and the complement are *dry* days; a trace amount of 0.3 mm is suggested
`inv`	`logical`; invert convention of the starting date such that the date is the first *wet* day if `inv=FALSE` (default), or the date is the first *dry* day if `inv=TRUE`

Details

The interarrival is the same as the *spell* period (i.e., dry spell), however this function simultaneously counts the number of *dry* and *wet* days relative to a single date. The date represents the first day of precipitation (if `inv=TRUE`, this convention is inverted to the first day of non-precipitation).

Missing or `NA` precipitation values voids the number of counted days between and within segments, which implies that days without precipitation need to explicitly have zeros.

Value

`interarrival` object (which inherits the `data.frame` class) with `date`, `wet`, `dry` columns.

The table has `id` and `name` attributes (if available from `x`).

Author(s)

Mike Toews

References

von Storch, H. and Zwiers, F.W., 1999, *Statistical analysis in climate research*, Cambridge: Cambridge University Press, 484 p.

See Also

plot.interarrival

Examples

```
data(mscdata)

van.int <- interarrival(mksub(mscdata, id=1108447))
summary(van.int)
van.int[which.max(van.int$dry),]
van.int[which.max(van.int$wet),]

plot(van.int, ylog=FALSE, maxy=30)
```

---

lars *Read and write data from LARS-WG file formats*

Description

Read and write data from the LARS-WG stochastic weather generator file formats; also convert to a format for HELP

Usage

```
# read synthetic or observed *.st file
read.lars(stfile, year.offset = 0)

# write observed climate data (*.st and/or *.sr)
write.lars(x, stfile, datfile, site, lat, lon, alt)

# experimental functions (may not work great; or at all!)
lars2help(infile, outfile, year.offset, site)
write.lars.scenario(file, x1, x2, name = "anomaly")
```

Arguments

stfile	file name with '*.st' extension; this is a 'site file' for LARS-WG which contains meta-data for the climate data, and has the location of the the climate data file; for write.lars, if this variable is NA or FALSE, this file will not be written (however, datfile must be defined)
datfile	file name with either '*.sr' or '*.dat' extension; contains climate data, as described by stfile; this does not need to be set if stfile is defined, as this datum is found in the 'st' file
file	file name with a '*.sce' extension; this is a 'scenario' file with absolute and relative changes of climate data
infile	input file
outfile	output file
x	data.frame of climate data
x1	same as x
x2	same as x
year.offset	offset of years between what is contained in the data files and what is needed in R to produce a reasonable 'Date'; this is required, for example, if synthetic data are produced that start from an arbitrary year '1' but represent climate from the year '2000'
site	same as '[SITE]' in 'st' file; if missing, this will try to read from attr(x$name); this is the same as a 'region' for HELP
name	scenario name
lat	same as 'LAT' in 'st' file; if missing, this will try to be read from attr(x$latitude)
lon	same as 'LON' in 'st' file; if missing, this will try to be read from attr(x$longitude)
alt	same as 'ALT' in 'st' file; if missing, this will try to be read from attr(x$elevation)

Details

These functions interface with the LARS-WG files (Version 4.0), which is a stochastic weather generator by Mikhail Semenov.

The climate data files used with LARS-WG have two parts: (1)~a 'site file' with a 'st' extension, containing the meta-data; and (2)~a data file with a '*.sr' or '*.dat' extension, containing all the data. The variable names are translated according to the following table:

seas	*LARS-WG*
year	'YEAR'
yday	'JDAY'
t_min	'MIN'
t_max	'MAX'
preicp	'RAIN'
solar	'RAD'
sun	'SUN'
pet	'PET'

To write climate data from R to a LARS-WG file, the `data.frame` names need to match those in the **seas**-side of the table.

Data exported from `write.lars` always has legal (according to the Gregorian calendar) and increasing sequence of days (even if there are gaps in `x$date`). Missing data values are written as `-99`.

Synthetically generated data from LARS-WG use a 365-day calendar, and may need to be converted to a Gregorian calendar, which can be done using `conv365toGregorian`.

`lars2help` and `write.lars.scenario` are experimental functions to translate data between LARS and HELP (see `write.help` for more info).

Author(s)

Mike Toews

References

LARS-WG was can be downloaded for academic and research uses from
http://www.rothamsted.bbsrc.ac.uk/mas-models/larswg.php

Semenov, M.A. and Barrow, E.M. (1997) "Use of a stochastic weather generator in the development of climate change scenarios". *Climate Change*: **35**, 397–414. doi: 10.1023/A:1005342632279

See Also

`write.help`, `read.sdsm`, `summerland` example synthetic data, `conv365toGregorian`

Examples

```
stfile <- system.file("extdata", "summerland.st", package="seas")
print(stfile)
summ <- read.lars(stfile, year.offset=1960)
head(summ)
str(summ)

# plot temperature
summ$t_mean <- rowMeans(summ[, c("t_min", "t_max")])
seas.temp.plot(summ)

# plot solar radiation
seas.var.plot(summ, "solar")

# plot precipitation
summ.ss <- seas.sum(summ)
image(summ.ss)
plot(seas.norm(summ.ss))
```

mkann *Make annum from a date*

Description

Discretizes a date into an annum, using a starting day to specify the start of a season, and ends in the next year.

Usage

```
mkann(x, start.day, calendar)
```

Arguments

x	A data.frame with a date column (of Date or POSIXct class) It may also be a vector of Date or POSIXct class
start.day	This is the starting day of the annum, and can be specified as either a Date, where year is ignored (e.g., as.Date("2000-08-01") for August 1st of any year); or it can be a day of the year, from 1–365
calendar	if unspecified, it will be attempted to be read from attr(x$date); otherwise it is assumed to be a normal proleptic Gregorian calendar; see year.length

Details

This date function finds the annual-breaks between seasons, using a start.day. Often, the start.day is 1, or January 1st, in which case simply the year is returned, since the season starts on January 1 and ends on December 31st. Otherwise, each annual break is set using start.day, and the annum is identified by the range of years, for example 1991_1992, identifying a season starting on start.day in 1991, and ending in the day before start.day in 1992.

The length of each year depends on the calendar; see year.length for details.

A choice of start.day can influence annual totals using seas.sum, such as annual precipitation. For instance, if a particular winter in the Northern hemisphere has snow before and after the new year, these would be divided counting annual sums based on the year, whereas if start.day were before the winter season, the annual sum would be calculated throughout the winter season.

Value

Returns factors for each date given in x, grouped by each annum.

Author(s)

Mike Toews

References

http://en.wikipedia.org/wiki/Gregorian_calendar

See Also

mkseas, seas.sum

Examples

```
data(mscdata)
dat <- mksub(mscdata, id=1108447)
dat$ann1 <- mkann(dat, start.day=1)
dat$ann2 <- mkann(dat, start.day=as.Date("2000-02-01"))
dat$ann3 <- mkann(dat, start.day=as.Date("2000-08-01"))
table(dat$ann1)
table(dat$ann2)
table(dat$ann3)
dat[26:36, c("date", paste("ann", 1:3, sep=""))]
```

---

mkseas *Make a date into a seasonal factor*

Description

Discretizes a date within a year into a bin (or factor) for analysis, such as 11-day groups or by month.

Usage

```
mkseas(x, width = 11, start.day = 1, calendar, year)
```

Arguments

x	A data.frame with a date column (of Date or POSIXct class) It can also be an integer specifying the Julian day (specify year to determine the leap year) If it is omitted, the full number of days will be calculated for the year, determined by either year or calendar
width	either numeric or other character value; if it is numeric, it specifies the number of days in each bin (default is 11 days); if character it specifies a common calendar usage, such as "mon" for months; see details below
start.day	this is the start of the season, specified as either a as a Date to specify a month and day (year is ignored; day of month is ignored if width relates to a month), or as a numeric day of year, between 1 and the number of days for the calendarter a leap day
calendar	used to determine the number of days per year and per bin; if not specified, a proleptic Gregorian calendar is assumed; see year.length
year	required if x is omitted, or if x is a Julian day integer and width is non-numeric; used to calculate leap year

Details

This useful date function groups *days* of a year into discrete bins (or into a `factor`). Statistical and plotting functions can be applied to a variable contained within each bin. An example of this would be to find the monthly temperature averages, where *month* is the bin.

If `width` is `integer`, the width of each bin (except for the last) will be exactly `width` days. Since the number of days in a year are not consistent, nor are always perfectly divisible by `width`, the numbers of days in the last bin will vary. `mkseas` determines that last bin must have at least 20% of the number of observations for a leap year, otherwise it is merged into the second to last bin (which will have extra numbers of days). If `width` is `numeric` (i.e. `366/12`), the width of each bin varies slightly. Using `width = 366/12` is slightly different than `width = "mon"`. Leap years only affect the *last* bin.

Other common classifications based on the Gregorian calendar can be used if `width` is given a `character` array. All of these systems are arbitrary: having different numbers of days in each bin, and leap years affecting the number of days in February. The most common, of course, is by *month* (`"mon"`). Meteorological quarterly seasons (`"DJF"`) are based on grouping three months, starting with December. This style of grouping is commonly used in climate literature, and is preferred over the season names 'winter', 'spring', 'summer', and 'autumn', which apply to only one hemisphere. The less common annual quarterly divisions (`"JFM"`) are similar, except that grouping begins with January. Zodiac divisions (`"zod"`) are included for demonstrative purposes, and are based on the Tropical birth dates (common in Western-culture horoscopes) starting with Aries (March 21).

Here are the complete list of options for the `width` argument:

- `numeric`: the width of each bin (or group) in days
- `366/n`: divide the year into `n` sections
- `"mon"`: month intervals (abbreviated month names)
- `"month"`: month intervals (full month names)
- `"DJF"`: meteorological quarterly divisions: DJF, MAM, JJA, SON
- `"JFM"`: annual quarterly divisions: JFM, AMJ, JAS, OND
- `"JF"`: annual six divisions: JF, MA, AJ, JA, SO, ND
- `"zod"`: zodiac intervals (abbreviated symbol names)
- `"zodiac"`: zodiac intervals (full zodiac names)

If a non-Gregorian calendar is used (see `year.length`), the number of days in a year can be set using `calendar` attribute in the `date` column (using `attr`). For example, `attr(x$date,"calendar") <- "365_day"` will set the dates using a 365-day per year calendar, where February is always 28-days in length. If this attribute is not set, it is assumed a normal Gregorian calendar is used. Calendars with 360-days per year (or 30-days per month) are incorrectly handled, since February cannot have 30 days, however this can be forced by including a duplicate February date in `x` for each year.

Value

Returns an array of `factor`s for each date given in `x`. The factor also has four attributes: `width`, `start.day`, `calendar` (assumed to be 366, unless from attribute set in `Date`), and an array `days` showing the maximum number of days in each bin.

See examples for its application.

Locale warning

Month names generated using "mon" or "months" are locale specific, and depend on your operating system and system language settings. Normally, abbreviated month names should have exactly three characters or less, with no trailing decimals. However, Microsoft-based operating systems have an inconsistent set of abbreviated month names between locales. For example, abbreviated month names in English locales have three letters with no period at the end, while French locales have 3–4 letters with a decimal at the end. If your OS is POSIX, you should have consistent month names in any locale. This can be fixed by setting `options("seas.month.len") <- 3`, which forces the length of the months to be three-characters in length.

To avoid any issues supporting locales, or to use English month names, simply revert to a C locale: `Sys.setlocale(loc="C")`.

Note

The phase of the Gregorian solar year (begins Julian day 1, or January 1st) is not in sync with the phase of "DJF" (begins Julian day 335/336) or "zod" (begins Julian day 80/81). If either of these systems are to be used, ensure that there are *several* years of data, or that the phase of the data is the same as the beginning Julian day.

For instance, if one years worth of data beginning on Julian day 1 is factored into "DJF" bins, the first bin will mix data from the first three months, and from the last month. The last three bins will have a continuous set of data. If the values are not perfectly periodic, the first bin will have higher variance, due to the mixing of data separated by nearly a year.

Author(s)

Mike Toews

References

http://en.wikipedia.org/wiki/Solar_calendar

See Also

mkann, seas.sum

Examples

```
# Demonstrate the number of days in each category
ylab <- "Number of days"

barplot(table(mkseas(width="mon", year=2005)),
        main="Number of days in each month",
        ylab=ylab)

barplot(table(mkseas(width="zod", year=2005)),
        main="Number of days in each zodiac sign",
        ylab=ylab)

barplot(table(mkseas(width="DJF", year=2005)),
```

```
        main="Number of days in each meteorological season",
        ylab=ylab)

barplot(table(mkseas(width=5, year=2004)),
        main="5-day categories", ylab=ylab)

barplot(table(mkseas(width=11, year=2005)),
        main="11-day categories", ylab=ylab)

barplot(table(mkseas(width=366 / 12, year=2005)),
        main="Number of days in 12-section year",
        sub="Note: not exactly the same as months")

# Application using synthetic data
dat <- data.frame(date=as.Date(paste(2005, 1:365), "%Y %j"),
  value=(-cos(1:365 * 2 * pi / 365) * 10 + rnorm(365) * 3 + 10))
attr(dat$date, "calendar") <- "365_day"

dat$d5 <- mkseas(dat, 5)
dat$d11 <- mkseas(dat, 11)
dat$month <- mkseas(dat, "mon")
dat$DJF <- mkseas(dat, "DJF")

plot(value ~ date, dat)
plot(value ~ d5, dat)
plot(value ~ d11, dat)
plot(value ~ month, dat)
plot(value ~ DJF, dat)

head(dat)

tapply(dat$value, dat$month, mean, na.rm=TRUE)
tapply(dat$value, dat$DJF, mean, na.rm=TRUE)

dat[which.max(dat$value),]
dat[which.min(dat$value),]

# start on a different day
st.day <- as.Date("2000-06-01")

dat$month <- mkseas(dat, "mon", start.day=st.day)
dat$d11 <- mkseas(dat, 11, start.day=st.day)
dat$DJF <- mkseas(dat, "DJF", start.day=st.day)

plot(value ~ d11, dat,
     main=.seasxlab(11, start.day=st.day))
plot(value ~ month, dat,
     main=.seasxlab("mon", start.day=st.day))
plot(value ~ DJF, dat,
     main=.seasxlab("DJF", start.day=st.day))
```

mksub *Make a subset of seasonal data*

Description

Creates a subset of a data.frame with temporal observations, using IDs and start and ending dates or years.

Usage

```
mksub(x, start, end, id)
```

Arguments

x	a data frame with temporal observations
start	either a starting Date or integer year; if omitted minimum will be used
end	either an ending Date or year; if omitted will use same year as start, and if start is omitted, will use maximum year
id	unique station identifier (if present), which is assumed to be a column of x as x$id; it is used to extract a subset of data from a single ID

Details

This utility function is useful for creating temporal subsets of seasonal data and for extracting a single station out of a data.frame with multiple stations or sets. The x object can have many columns, representing measured variables for each day, which will be returned with their original attributes.

If id is used, that station will be extracted from x. If id is not provided, but there are more than one unique IDs in x$id, the first unique ID will be extracted, with a warning.

Value

Returns a subset of a data.frame with the same columns and attributes as x, except id, which will be retained as an attribute (e.g., attr(x,"id")).

Author(s)

Mike Toews

See Also

read.msc, mscdata

Examples

```
data(mscdata)

# All available data from one station
summary(mksub(mscdata, id=1108447))

# One year
str(mksub(mscdata, id=1108447, start=1980))

# A range of years
str(mksub(mscdata, id=1108447, start=1980, end=1989))

# A range of dates
summary(mksub(mscdata, id=1108447,
              start=as.Date("1975-08-01"),
              end=as.Date("2000-07-31")))
```

---

mscdata *Meteorological Service of Canada sample climate data*

Description

Sample climate data from the Meteorological Service of Canada (MSC) climate stations in western Canada.

Usage

```
data(mscdata)
```

Format

A `data.frame` with 26358 daily observations on the following 10 variables (metric units of '°C' and 'mm' *per day*):

`id`: `factor` used to distinguish multiple stations within a single data frame

`year`: `integer` year

`yday`: `integer` day of year; 1–365 or 1–366

`date`: `Date` class

`t_max`: daily maximum temperature

`t_min`: daily minimum temperature

`t_mean`: daily mean temperature

`precip`: total daily precipitation

`rain`: total daily liquid-phase precipitation

`snow`: total daily solid-phase precipitation

The climate variables have attributes (`attr` of `units` and `long.name` to identify their units and long names for plotting labels.

There are three climate stations in this data frame from:

ID	Station Location	Province
1096450	Prince George	BC
1108447	Vancouver	BC
2100630	Haines Junction	YT

All data spans from 1975 to 2004 for each station. Missing values are present.

Details

The field `id` is optional, but very handy when handling multiple stations. Also, the day of year (`yday`) and `year` are optional, since these are stored in the `date`, using `dat$date <- as.Date(paste(dat$year,dat$yday),"%Y %j")`.

The `units` and `long.name` attributes stored in the climate variables are optional, but help annotate the graphics.

Author(s)

Mike Toews

Source

Data provided by the Meteorological Service of Canada, with permission. This data may only be reproduced for personal use; any other reproduction is permitted only with the written consent of Environment Canada.

http://www.msc.ec.gc.ca/
http://climate.weatheroffice.gc.ca/contacts/

See Also

`mscstn` has MSC station ID codes, locations and names; `mksub` produces subsets of data; `read.msc` reads MSC archive files, such as `A1128551.DLY`

Examples

```
data(mscstn)
data(mscdata)
par.orig <- par(no.readonly=TRUE)

# structure in R
str(mscdata)

# first few rows
head(mscdata)

# here are all the station IDs
stnids <- levels(mscdata$id)

# show all data
rng.p <- range(mscdata$precip, na.rm=TRUE)
```

```
rng.t <- range(mscdata$t_mean, na.rm=TRUE)
par(mfcol=c(2, 3), mgp=c(2, 1, 0), mar=c(3, 3, 3, 1), bty="l")
for (n in levels(mscdata$id)) {
  dat <- mscdata[mscdata$id == n,]
  plot(t_mean ~ date, dat, "l", col="red", ylim=rng.t)
  abline(h=0)
  plot(precip ~ date, dat, "l", col="blue", ylim=rng.p, main=n)
}
par(par.orig)

# show stations and station names available in this data frame
data.frame(stnids, name=getstnname(stnids))
dat <- mksub(mscdata, id=1108447)
dat$month <- mkseas(dat, "mon")
plot(t_mean ~ date, dat, "l")
plot(t_mean ~ date, dat, subset=(month == "Dec"))
seas.temp.plot(dat)
year.plot(dat)

# plot high-resolution statistics
dly.tmp <- tapply(dat$t_mean, dat$yday,
  quantile, c(5, 25, 50, 75, 95) / 100, na.rm=TRUE)
dly <- data.frame(yday=1:366,
  t(matrix(unlist(dly.tmp), nrow=5)))
names(dly) <- c("yday", "d5", "d25", "median", "d75", "d95")
plot(median ~ yday, dly, "n", ylim=c(-5, 25),
  ylab="mean temperature", xlab="day of year")
polygon(c(1:366, 366:1), c(dly$d5, rev(dly$d95)),
        border=FALSE, col="grey80")
polygon(c(1:366, 366:1), c(dly$d25, rev(dly$d75)),
        border=FALSE, col="grey50")
lines(median ~ yday, dly)
abline(h=0)
```

`mscstn` *Meteorological Service of Canada station information*

Description

Meteorological Service of Canada weather station data, including national ID, station ID, Province, latitude and longitude.

Format

A `data.frame` with 4493 climate stations with the following 6 columns:

`name`	Full station name
`nid`	National ID, alphanumeric key
`sid`	Station ID, also used for airport codes
`prov`	Canadian Province

lat	Decimal degrees latitude; NAD83
long	Decimal degrees longitude; NAD83

Details

This data object is used as a look-up table to convert a unique station identifier (nid) or ID into a station name, using getstnname.

Currently, this data only includes weather stations from Alberta, British Columbia and the Yukon.

Author(s)

Mike Toews

Source

Provided by the Meteorological Service of Canada (http://www.msc.ec.gc.ca/), with permission.

See Also

getstnname, mscdata, read.msc

Examples

```
str(seas::mscstn)

table(mscstn$prov)
plot(lat ~ long, seas::mscstn, pch=".")
```

---

plot.interarrival *Plot interarrivals for precipitation*

Description

Plots interarrivals for precipitation using boxplots, giving the typical number of continuous wet days and dry days (or spells) throughout the season. The mean value is also drawn as a single line.

Usage

```
## S3 method for class 'interarrival'
plot(x, width = 11, start = 1, rep = 0, start.day = 1,
          ylog = FALSE, maxy, main, ...)
```

Arguments

x	an interarrival object with numbers of dry/wet days
width	size of bin; see mkseas
start	starting bin number; e.g., if width="mon" and start=5, the plot will start on "May" at the left-hand side
rep	repetition of the bins in the boxplot
start.day	when width is numeric, this is the starting day of the year for the fist bin, or it can be a Date to specify a month and day (year is ignored)
ylog	logical; y-axis is logarithmic
maxy	maximum number of days for the y-axis; it can either be passed as c(wet, dry), or as a single value for both
main	main title for plot, otherwise other title will be automatically generated
...	ignored

Author(s)

Mike Toews

See Also

interarrival, seas.var.plot

Examples

```
data(mscdata)

dat.int <- interarrival(mksub(mscdata, id=1108447))
plot(dat.int, width="mon")

plot(dat.int, ylog=FALSE, maxy=35, rep=10)
```

---

plot.seas.norm *Plot seasonal normal of a variable, including precipitation normals*

Description

Plots a "normal" of a seasonal variable, including a precipitation normal (which shows rain and snow fractions, where available). Significant missing data values are also indicated.

Usage

```
## S3 method for class 'seas.norm'
plot(x, start = 1, rep = 0, ylim,
     varwidth = FALSE, normwidth = FALSE,
     leg, add.alt = FALSE, main, ylab, ...)
```

Arguments

x	a seas.norm object created by either seas.norm or precip.norm
start	starting bin
rep	repeat bins
ylim	range of y-axis; either as a single value, c(0, max), or as two values c(min, max)
varwidth	logical; varies the width of each bar directly proportional to the frequency of active days (defined by a threshold); the value is normalized according to the next argument
normwidth	normalizes the width of the bars to a fixed numeric value (in days), or the maximum value if given TRUE; the default FALSE value normalizes each bar to the number of potentially active days
leg	if TRUE shows a legend summary of the statistics in the upper left hand corner; it can also be a c(x, y) pair or "locator" to manually place the legend on the active graphics device
add.alt	logical; adds imperial units on the right-hand y-axis
main	title for plot; if it is missing, then it will automatically be generated
ylab	y-axis label; if it is missing, then it will automatically be generated
...	ignored

Details

The varwidth variable is useful for separating different precipitation patterns throughout the season. It changes the width of the bar proportional to the frequency of precipitation events within the bin. Ideally, the bars will be tall and narrow with intense storms that occur infrequently, such as convective storms. Conversely the bars will be broader with less-intense rainfall events occurring more frequently.

Author(s)

Mike Toews

See Also

seas.norm, precip.norm, seas.sum

Examples

```
data(mscdata)
dat <- mksub(mscdata, id=1108447)
d.ss <- seas.sum(dat)
plot(seas.norm(d.ss))
plot(precip.norm(d.ss, fun=median))
plot(precip.norm(d.ss, fun=mean))
plot(precip.norm(d.ss, fun=mean, norm="active"))
plot(precip.norm(d.ss, fun=median, norm="active"))
plot(precip.norm(d.ss), start=15, rep=12)
```

```
mar <- par("mar")
plot(precip.norm(d.ss), add.alt=TRUE)

par(mar=mar)
d2.ss <- seas.sum(dat, start.day=as.Date("2000-08-01"))
plot(precip.norm(d2.ss, fun="mean"))
```

`plot.seas.sum` *Plot boxplots of normalized seasonal sums*

Description

Plots normalized seasonal sums using boxplots.

Usage

```
## S3 method for class 'seas.sum'
plot(x, var, norm = "days", year.filter, ylim,
    start = 1, rep = 0, col = "lightgrey", main, ylab, ...)
```

Arguments

`x`	a `seas.sum` object created by `seas.sum`
`var`	name of seasonal variable in `x`
`norm`	a variable to normalize by, either `"days"` (to produce *unit/day*) or `"active"` (*unit/day*, when active); it may also be a `matrix` with the same dimensions as `x$days`
`year.filter`	use only these years for analysis
`ylim`	either a single value for `c(0, ylim)`, or a range of `c(min, max)` for the y-axis limits
`start`	starting bin at left-hand side of plot
`rep`	repeat bins on right-hand side of plot
`col`	colour for boxplot, default is `"lightgrey"`
`main`	title for plot; if it is missing, then it will automatically be generated
`ylab`	y-axis label; if it is missing, then it will automatically be generated
`...`	ignored

Details

This function is a boxplot interpretation of a `seas.sum` object. This is not the same as treating `var` as a continuous variable and using `seas.var.plot`, since a `seas.sum` object has been smoothed. Daily extreme values are not well represented here as a result.

Warning

The appearance of the boxplots are sensitive to the `width` parameter specified in the `seas.sum` function on strongly discontinuous variables. Small bin widths capture the discontinuities better than wider bins, and changes the skew of the distribution.

For instance, the median will appear to decrease as `width` decreases.

Author(s)

Mike Toews

See Also

`seas.sum`, `image.seas.sum`, `seas.norm`

Examples

```
data(mscdata)
par.orig <- par(no.readonly=TRUE)
on.exit(par.orig)

dat <- mksub(mscdata, id=1108447)
dat.ss <- seas.sum(dat)

# Normalized by the number of days in each bin
plot(dat.ss)

# Normalized by the number of active days in each bin
plot(dat.ss, norm="active")

# Snow, using a different start day, and a better y-axis:
dat2.ss <- seas.sum(dat, var="snow", width="mon",
                    start.day=as.Date("2000-08-01"))
par(yaxs="i")
plot(dat2.ss, var="snow")
plot(dat2.ss, var="snow", norm="active")
```

---

precip.dep *Cumulative precipitation departure*

Description

Calculate the cumulative precipitation departure (CPD) for a station with a given precipitation normal.

Usage

```
precip.dep(x, norm, var = "precip")
```

Arguments

x	a seasonal data.frame of climate data
norm	a precip.norm object containing the precipitation normal for the same station as x
var	a common seasonal variable found in x and norm

Details

This function is useful for looking at the behaviour of a precipitation time-series in relation to its precipitation normal over an extended period of time. This is especially useful for identifying changes in precipitation, and is useful for relating to groundwater recharge patterns (e.g., http://www.env.gov.bc.ca/wsd/data_searches/obswell/obsw002.html).

Value

Returns a data.frame similar to x, but contains the departures in the dep column.

Note

The selection of fun in precip.norm, such as using mean or median, will affect the result of this function; width has only a minor effect.

Periods with missing (NA) values in var of x will have a flat departure, neither increasing nor decreasing.

Author(s)

Mike Toews

See Also

precip.norm

Examples

```
data(mscstn)
data(mscdata)

dat <- mksub(mscdata, id=1108447)
dat.ss <- seas.sum(dat)
dat.dep <- precip.dep(dat,precip.norm(dat.ss, fun="mean"))
plot(dep ~ date, dat.dep, type="l", main="CPD from mean normals")

dat.dep <- precip.dep(dat, precip.norm(dat.ss, fun="median"))
plot(dep ~ date, dat.dep, type="l", main="CPD from median normals")
```

read.msc *Read a MSC archive file into a data.frame*

Description

Reads a Meteorological Service of Canada (MSC) digital archive files (HLY and DLY formats) into a data.frame.

Usage

```
read.msc(file, flags = FALSE, add.elem, format, verbose = TRUE)
```

Arguments

file	file name (with path, if not in getwd); it can also be a connection, such as bzfile
flags	logical return the flags with the data.frame
add.elem	either a data.frame or a list with additional elements not found in this function
format	force this function to read a format (not recommended, since this is automatically done)
verbose	logical verbose output, such as number of stations, elements, records and years in the archive file

Details

This function currently reads in HLY (hourly) and DLY (daily) archive formats. This is automatically detected. The other formats, FIF (fifteen-minute) and MLY (monthly), are not currently supported.

The input file can include multiple stations and multiple elements (measured variables). The multiple stations are deciphered through the id column, and the multiple variables appear as columns to the output data frame.

This function currently only reads a limited number of elements, however additional elements can be used by editing two lines in the R source for this function.

Value

Returns a data.frame object with the following minimum fields:

id: factor used to distinguish multiple stations within a single data frame

year: integer year

yday: integer day of year; 1–365 or 1–366

date: Date, useful for plotting a continuous time-series

datetime: POSIXct, includes date and time info, only included if file is in HLY archive format

`element`: numeric, with attributes set for `units` and `long.name`; these can be changed using attr on `dat$varname`

`flag`: `factor`; included if `flags=TRUE`

The are as many `element` columns for each element found in the archive file, such as:

alias	name	long.name	units
1	t_max	daily maximum temperature	°C
2	t_min	daily minimum temperature	°C
3	t_mean	daily mean temperature	°C
10	rain	total rainfall	mm
11	snow	total snowfall	mm
12	precip	total precipitation	mm
13	snow_d	snow on the ground	cm
...	...	*other elements*	*optional*

Additional elements (or variables) can be added by specifying the `element` variable, and their units can be set using, for example, `attr(dat$var, "units") <- "cm"`.

Units are in common metric units: 'mm' for precipitation-related measurements, 'cm' for snow depth, and '°C' for temperature. The `flag` columns are a single character factor, described in the MSC Archive documentation. Units are added to each column using, for example `attr(dat$precip, "units") <- "mm"`.

Author(s)

Mike Toews

Source

Climate data can be requested from MSC, or can be obtained directly from the Canadian Daily Climate Data (CDCD) CD-ROMs, which are available for a free download (procedure described in A1128551.DLY).

References

http://climate.weatheroffice.gc.ca/prods_servs/documentation_index_e.html Technical Documentation - Documentation for the Digital Archive of Canadian Climatological Data (Surface) Identified By Element

http://climate.weatheroffice.gc.ca/prods_servs/index_e.html#cdcd CDCD CD-ROM download location

See Also

mscstn, mksub, mkseas, A1128551.DLY

Examples

```
fname <- system.file("extdata", "A1128551.DLY", package="seas")
print(fname)
dat <- read.msc(fname)
```

```
print(head(dat))

seas.temp.plot(dat)
year.plot(dat)

# Show how to convert from daily to monthly data
dat$yearmonth <- factor(paste(format(dat$date, "%Y-%m"), 15, sep="-"))
mlydat <- data.frame(date=as.Date(levels(dat$yearmonth)))
mlydat$year <- factor(format(mlydat$date, "%Y"))
mlydat$month <- mkseas(mlydat, "mon")

# means for temperature data
mlydat$t_max <- as.numeric(
    tapply(dat$t_max, dat$yearmonth, mean, na.rm=TRUE))
mlydat$t_min <- as.numeric(
    tapply(dat$t_min, dat$yearmonth, mean, na.rm=TRUE))
mlydat$t_mean <- as.numeric(
    tapply(dat$t_mean, dat$yearmonth, mean, na.rm=TRUE))

# sums for precipitation-related data
mlydat$rain <- as.numeric(
    tapply(dat$rain, dat$yearmonth, sum, na.rm=TRUE))
mlydat$snow <- as.numeric(
    tapply(dat$snow, dat$yearmonth, sum, na.rm=TRUE))
mlydat$precip <- as.numeric(
    tapply(dat$precip, dat$yearmonth, sum, na.rm=TRUE))
print(head(mlydat), 12)

# Show how to convert from a HLY file into daily summaries
## Not run:
hlydat <- read.msc(bzfile("HLY11_L1127800.bz2"), flags=TRUE)
hlydat$date <- factor(hlydat$date)

# sum the solar radiation for each day to find the 'total daily'
sumdat <- tapply(hlydat$solar, hlydat$date, sum, na.rm=TRUE)
dlydat <- data.frame(date=as.Date(names(sumdat)),
                     solar=as.numeric(sumdat))

# sum the number of hours without measurements
sumdat <- tapply(hlydat$solar, hlydat$date,
                 function(v)(24 - sum(!is.na(v))))
dlydat$na <- as.integer(sumdat)

# quality control to remove days with less than 4 hours missing
Summerland <- dlydat[dlydat$na < 4,]

attr(Summerland$solar, "units") <- "W/(m^2*day)"
attr(Summerland$solar, "long.name") <- "Daily total global solar radiation"
seas.var.plot(Summerland, var="solar", col="yellow", width=5)

## End(Not run)
```

sdsm *Read and write from SDSM*

Description

Reads and writes the data format used in SDSM's 'DAT' and 'OUT' extensions.

Usage

```
# reading
read.sdsm(file, start = 1961, end = 2000, calendar)

# writing
write.sdsm(dat, var, start, end, file = "")
```

Arguments

file	name of 'DAT' or 'OUT' file
dat	data.frame of variables to be written
start	starting year
end	ending year
var	name of variable to be written from dat
calendar	calender used for data; if unspecified, this is assumed to be proleptic Gregorian (normal); however, for CCCma models this should be "365_day", and for Hadley models this should be "360_day"; see year.length

Details

This function readings and writes climate data with the Statistical Downscaling Model, or SDSM. The model uses 'DAT' extensions for input data, such as daily observations of mean temperature, and 'OUT' extensions for modeled output.

Value

read.sdsm returns a data.frame of the measured variables. The variables are named V1...Vn, for *n* ensembles.

If a calendar is specified, this is stored as an attribute in the date data frame column.

Author(s)

Mike Toews

References

Wilby, R.L., Dawson, C.W. and Barrow, E.M. 2002, 'SDSM - a decision support tool for the assessment of regional climate change impacts', *Environmental Modelling Software*, **77**, 473–471.

SDSM can be downloaded free-of-charge for Windows platforms from
http://co-public.lboro.ac.uk/cocwd/SDSM/

CGCM1 and HADCM3 model data for SDSM can be downloaded from the Canadian Climate Impacts and Scenarios website:
http://www.cics.uvic.ca/scenarios/sdsm/select.cgi

See Also

read.msc, change

Examples

```
## Not run:
# reading
fname <- system.file("extdata", "GF_2050s_precip.OUT", package="seas")
gf50 <- read.sdsm(fname)
gf50.ss <- seas.sum(gf50, var=paste("V", 1:20, sep=""), name="Grand Forks")

# analysis
image(gf50.ss, var="V1")
image(gf50.ss, var="V2")
image(gf50.ss, var="V3")

# writing
data(mscdata)
hj <- mksub(mscdata, id=2100630)
fname <- paste(tempdir(), "HJ_Obs_prcp.DAT", sep="/")
write.sdsm(hj, "precip", 1961, 2000, fname)

## End(Not run)
```

---

seas.check *Check the suitability of a data.frame, or seas.sum for seas*

Description

Check the suitability of a data.frame or seas.sum object for **seas**.

Usage

```
seas.df.check(x, orig, var)
seas.sum.check(x, orig, var, norm, year.filter, ann.only)
```

Arguments

x	a data frame with temporal observations
orig	the original name of the data frame, for error messages
var	one or more variables in x, which are tested; ignored if NULL or missing
norm	something to normalize var; it can either be the name of an array in x, a matrix (bin vs years), or a 3-dim array (used to normalize multiple var); this is tested to see if norm exists, and that the dimension are consistent with x
year.filter	a subset of x$years, which filters all the related arrays in the returned value
ann.only	x$seas arrays are ignored

Details

This utility function simply checks the suitability of a data.frame or seas.sum objects for use with **seas**.

If x is data.frame (using seas.df.check that is really required, is a 'date' column, named x$date with a class of either link{POSIXct} or link{Date}, and one or more variables in the var columns of x.

There must be at least one finite observation in each of var, if supplied.

These function is used within other functions, and is not intended to be called directly.

Value

seas.df.check returns a few helpful items from x in a list using invisible:

- id:station ID from one of attr(x,"id") or x$id[1]
- name:name of seasonal data, such as a place
- year.range:integers of start, and ending years
- calendar:an attribute from x$date; otherwise this will be NULL for a normal proleptic Gregorian calendar
- main:main title, from .seastitle
- units:units for var[1]
- long.name:long name for var[1]
- ylab:y-axis label for var[1]

seas.sum.check returns x with modifications, depending on norm and year.filter.

Author(s)

Mike Toews

See Also

hidden functions for **seas**

Examples

```
data(mscdata)
dat <- mksub(mscdata, id=1108447)
str(seas.df.check(dat))

dat.ss <- seas.sum(dat)
str(seas.sum.check(dat.ss, norm="days"))
```

`seas.norm` *Calculate annual and seasonal 'normal' statistics, including precipitation normals*

Description

Calculates annual and seasonal 'normal' statistics on a `seas.sum` object, including precipitation normals for rain, snow and total precipitation.

Usage

```
seas.norm(x, var, fun = "median", norm = "days", year.filter,
          ann.only = FALSE, precip.norm = FALSE)

precip.norm(x, fun = "median", norm = "days", year.filter)
```

Arguments

`x`	`seas.sum` object
`var`	variable name for the 'normal'; if omitted will use `x$prime` (the prime variable of the `seas.sum` object), or if `precip.norm=TRUE` will be `"precip"`
`norm`	variable for normalization of the sum, usually the number of "days" in each bin, but it can also be "active" to estimate the precipitation normal for days of active precipitation
`year.filter`	filter specific years for analysis
`fun`	`character` of an existing `function` object, or a `function` to operate across the number of years of observations, usually `"mean"` or `"median"` (default); details described below
`ann.only`	only annual statistics returned (saves time from other calculations)
`precip.norm`	`logical`; computes precipitation normal statistics, which is done slightly differently since it involves rain, snow and total precipitation; if TRUE, `x$var` must include "rain", "snow" and "precip" summed variables

Details

This function calculates the statistics of precipitation data on an *annual* and *seasonal* scope from a `seas.sum` object.

The seasonal input data are normalized by the number of days in each bin, to produce a precipitation rate in 'mm/day'. This is because the number of days in each bin is not equal. The function `fun` is then applied to the normalized precipitation, and operates along each bin, across multiple years of data. The supplied function is usually `"median"` or `"mean"`, but it can also be a built in R function, such as `"var"` for variance, or a composite such as:

`function(i, na.rm)(quantile(i, 0.2, na.rm=na.rm, names=F))`	the 20% quantile
`function(i, na.rm)(mean(i, na.rm=na.rm)/(sd(i, na.rm=na.rm)^3))`	skewness

If `fun = "mean"`, then the statistics are straightforward (using `apply`), however if `fun = "median"` and there are more than 2 years of data, a different approach is taken. The *median* is a special case of the *quantile* function, where the probability is 50% of the population. The `median` and `quantile` functions are more resistant to outliers than `mean`, and can have advantages on precipitation data. Precipitation occurring at a given time of year does not have a normal distribution since it is a value that is not always occurring. It often has a left-skewed distribution, consisting of many zero measurements, and few extreme precipitation events.

In this function, if `fun = "median"` (default) the `median` function is only used to calculate the median annual precipitation. The `quantile` function is used to calculate the seasonal statistics, since the sum of medians applied in each bin are less than the median annual precipitation. This is because there are usually many measurements of no rain, which skew the distribution to the left. The percentile for the quantile function is found using a secant method (Cheny and Kincaid, 1999) such that the sum of the quantiles from each bin are equal to the median of the annual precipitation.

Snow and rain (which are the two components of precipitation) are calculated similarly (if `fun = "median"`). The annual total rain and snow amounts are determined by finding the percentile of a quantile function where the sum is equal to the median of the annual precipitation. The seasonal snow and rain amounts are independently found using the same method to find the seasonal precipitation. The fraction of the snow in each bin, $snow.frac.b = snow.b/(snow.b + rain.b)$ is multiplied by the seasonal precipitation to determine the seasonal rain and snow amounts. This is because the sum of rain and snow in each bin does not equal the seasonal precipitation. This way, a figure with `precip.only = TRUE` and `= FALSE` will have identical daily precipitation rates in each bin.

The pitfalls of calculating precipitation 'normals' is that it assumes that precipitation occurs *every* day at a constant rate within each bin. This is not realistic, as the precipitation rates are much higher when it is actually occurring.

Value

Returns a `precip.norm` object, which is a `list` with the following elements:

`seas`	An `array` of seasonal precipitation statistics: `precip`, `rain` and `snow` (if `precip.only = FALSE`) are in 'mm/day'; `freq` and `na` are the fraction of a day in which precipitation is occurring and that data is missing.
`ann`	Annual precipitation statistics. `precip`, `rain` and `snow` (if `precip.only = FALSE`) are in 'mm/year'; `active` and `na` are the number of days per year which are active (for example, days with precipitation) and that data are missing.

width	from x
bins	from x
bin.lengths	maximum number of days in each bin
year.range	from x
start.day	from x
var	same as input parameter
units	units for var, usnig attr
long.name	long name for var, using attr
ann.only	ann.only same as input parameter
precip.only	from same as input parameter
a.cut	from x
fun	function used in analysis
id	from x
name	from x

Note

Seasonal data are explicitly normalized to a rate *per day* (i.e., mm/day), and not *per month* (i.e., mm/month). This is because a time-derivative *per month* has unequal intervals of time, ranging between 28 to 31 days. This directly creates up to 10% error in the analysis between months.

Units for annual normals, however, remain *per year*, since a year is a suitable time derivative.

Author(s)

Mike Toews

References

Cheny, E. W. and Kincaid, D. 1999, *Numerical Mathematics and Computing*, Pacific Grove: Brooks/Cole Pub., 671 p.

Guttman, N.B. 1989, 'Statistical descriptors of climate', *American Meteorological Society*, **70**, 602–607.

See Also

plot.seas.norm, seas.var.plot, precip.dep

Examples

```
data(mscdata)

# calculate precipitation normal
dat <- mksub(mscdata, id=1108447)
dat.ss <- seas.sum(dat)
dat.nm <- precip.norm(dat.ss, fun="mean")
```

```
# plot precipitation normal
plot(dat.nm) # this is the same as plot.precip.norm(dat.nm)

# use precipitation normal
dat.dep <- precip.dep(dat, dat.nm)
plot(dep ~ date, dat.dep, type="l",
     main="CPD from mean normals")
```

`seas.sum` *Seasonal sum data object*

Description

Create a seasonal sum object used for analysis of precipitation data (among other things, such as recharge rates); this object has sums in each 'bin' of a season, as well as for each annum (or year).

Usage

```
seas.sum(x, var, width = 11, start.day = 1, prime,
         a.cut = 0.3, na.cut = 0.2)
```

Arguments

`x`	a `data.frame` with daily variables to be summed, such as precipitation
`var`	the names of one or more variables in `x`, such as `c("rain","snow","precip")`
`width`	a number specifying the width of the bin (factor) in days, or `"mon"` for months (see `mkseas` for others)
`start.day`	the first day of the season, specified as either a `Date` or as an integer day of the year; annual sums start on this day, and end a day before `start.day` in the following year
`prime`	a single variable from `var` which is the prime variable of interest, such as `"precip"`; this is the variable used for comparison with `a.cut` and `na.cut` in the resulting `active` and `na` dimensions
`a.cut`	cut-off value for the day to be considered an *active* or 'wet day' (based on the `prime` variable); a trace amount of 0.3 mm is suggested; if `a.cut` is `NA` or zero, the active variable and analysis will be ignored
`na.cut`	cut-off fraction of missing values; can be single value or a vector for `c(annual,seasonal)`; details given below

Details

This function is used to discretize and sum time-varying data in a `data.frame` for analysis in *seasonal* and *annual* parts. This is particularly useful for calculating normals of rates, such as precipitation and recharge. This function simply sums up each variable in each bin for each annum (or year), and provides the results in several arrays.

Sums are *not* normalized, and represent a sum for the number of days in the bin (seasonal data) or annum (for annual data). Seasonal data can be normalized by the number of days (for a rate per day) or by the number of active days where `prime > a.cut`.

For annual sums, annums with *many* missing values are ignored (receiving a value of `NA`) since it has insufficient data for a complete sum. The amount of allowable `NA` values per annum is controlled by `na.cut[1]`, which is a fraction of `NA` values for the whole annum (default is 0.2).

The seasonal sums are calculated independently from the annual sums. Individual bins from each year with *many* missing values are ignored, where the amount of allowable `NA` values is controlled by `na.cut[2]` (or `na.cut[1]`, if the `length` of `na.cut` is 1). The default fraction of `NA`s in each bin of each annum is 0.2.

Value

Returns a `seas.sum` object, which is a `list` with the following elements:

`ann`**:** A `data.frame` of annual data; the columns are:

- `year`**:** year, or annum
- `active`**:** the number of 'active' days in the year where the prime variable is above `a.cut` (if used)
- `days`**:** number of days in each year
- `na`**:** number of missing days in the year
- ***var(s)***: annual sum of one or more variable; if the original units were mm/day, they are now mm/year

`seas`**:** An `array`: of seasonal data; the dimensions are:

- `[[1]]`**:** year, or annum
- `[[2]]`**:** bins, or seasonal factors generated by `mkseas`
- `[[3]]`**:** sums of variables for each bin of each year; if the original unit was mm/day, it is now mm per number of days, which is held in the `days` item

`active`**:** the number of 'active' days in the bin where the prime variable is above `a.cut` (if used)

`days`**:** an array of the number of days in each bin; this array is useful for normalizing the numbers in `seas` to comparable units of mm/day

`na`**:** number of missing days in each bin

`start.day`**:** same as input

`years`**:** years (same as `ann[[1]]` and `seas[[1]]`); if `start.day` is not 1, this represents the starting and ending years (i.e., `1991_1992`) of each annum; see `mkann`

`var`**:** variable(s) which the sums represent (part of `ann[[2]]` and `seas[[3]]`)

`units`**:** a `list` of units for each `var`, such as "mm/day"; these are obtained from the `units` attribute (using `attr`) found in `x$var`

`long.name`**:** a `list` of long names for each `var`; these are obtained from `long.name` in `x$var`; set to be `var` if `NULL`

`prime`**:** a `prime` variable, such as `"precip"`

`width`**:** `width` argument passed to `mkseas`

`bins`**:** names of bins returned by `mkseas` (same as `seas[[2]]`)

`bin.lengths`: the maximum length in days for each `bin`

`year.range`: range of years from `x`

`precip.only`: value used in argument (modified if insufficient data found in `x`)

`na.cut`: value used in argument

`a.cut`: value used in argument; if it is zero or `NA`, this will be `FALSE`

`id`: from `attr(x,"id")` (`NULL` if not set)

`name`: from `attr(x,"name")` (`NULL` if not set)

Author(s)

Mike Toews

See Also

To view the result try `image.seas.sum`, or alternatively, `plot.seas.sum`

To calculate and view a "normal", use `seas.norm` and `plot.seas.norm`, or for precipitation use `precip.norm` and `plot.precip.norm`

Examples

```
data(mscdata)
dat <- mksub(mscdata, id=1108447)
dat.ss <- seas.sum(dat, width="mon")

# Structure in R
str(dat.ss)

# Annual data
dat.ss$ann

# Demonstrate how to slice through a cubic array
dat.ss$seas["1990",,]
dat.ss$seas[,2,] # or "Feb", if using English locale
dat.ss$seas[,,"precip"]

# Simple calculation on an array
(monthly.mean <- apply(dat.ss$seas[,,"precip"], 2, mean,na.rm=TRUE))
barplot(monthly.mean, ylab="Mean monthly total (mm/month)",
main="Un-normalized mean precipitation in Vancouver, BC")
text(6.5, 150, paste("Un-normalized rates given 'per month' should be",
"avoided since ~3-9% error is introduced",
"to the analysis between months", sep="\n"))

# Normalized precip
norm.monthly <- dat.ss$seas[,,"precip"] / dat.ss$days
norm.monthly.mean <- apply(norm.monthly, 2, mean,na.rm=TRUE)
print(round(norm.monthly, 2))
print(round(norm.monthly.mean, 2))
barplot(norm.monthly.mean,
```

```
    ylab="Normalized mean monthly total (mm/day)",
main="Normalized mean precipitation in Vancouver, BC")

# Better graphics of data
dat.ss <- seas.sum(dat, width=11)
image(dat.ss)
```

`seas.temp.plot` *Plot seasonal temperature normals*

Description

Plot seasonal temperature normals using boxplots, and also plot seasonal diurnal variability between minimum and maximum temperature.

Usage

```
seas.temp.plot(x, width = 11, start = 1, rep = 0, start.day = 1,
           var = c("t_min", "t_max", "t_mean"),
           add.alt = FALSE, ylim, main, ylab, ...)
```

Arguments

`x`	a `data.frame` with `Date`, `t_min`, `t_max`, and (optionally) `t_mean` columns; `x` can also have `id` or `name` attributes to help give a title for the plot
`width`	size of bin; see `mkseas`
`start`	starting bin number; e.g., if `width="mon"` and `start=5`, the plot will start on "May" at the left-hand side
`rep`	repetition of the bins in the boxplots
`start.day`	if `width` is numeric, this is the day of year which is considered to be the start of the first bin
`var`	array specifying the names of the columns in `x` which relate to the *minimum*, *maximum* and *mean* temperatures; the `units` attribute for the y-axis label are taken from the `minimum`, if available, otherwise it is assumed it is in °C
`add.alt`	`logical`; add an alternative scale: if the units are in °C, the alternative is °F; if units are °F, the alternative is °C; and if units are K, the alternative is °C
`ylim`	`c(min, max)` range for temperature, or y-axis
`main`	title for plot; if it is missing, then it will automatically be generated
`ylab`	y-axis label; if it is missing, then it will automatically be generated
`...`	ignored

Details

Plots boxplots for seasonal temperature normals from mean daily temperature, and diurnal variability with the mean difference of daily minimum and maximum temperatures (red vertical lines). If the mean is not supplied, it is calculated from the mean of daily maximum and minimum temperatures.

Value

Returns values from boxplot statistics on mean temperature.

Note

This function was formerly named plot.seas.temp, but required renaming as it is not an S3 method.

Author(s)

Mike Toews

See Also

seas.var.plot, plot.seas.norm, year.plot

Use mksub to make a subset of x.

Examples

```
data(mscdata)

dat <- mksub(mscdata, id=1108447)
seas.temp.plot(dat)
seas.temp.plot(dat, width="mon", add.alt=TRUE)

# starting and ending elsewhere
seas.temp.plot(dat, start=18, rep=3)
```

---

seas.var.plot *Plot seasonal normals of a given variable*

Description

Plot seasonal normals of a variable using boxplots.

Usage

```
seas.var.plot(x, var, width = 11, start = 1, rep = 0, start.day = 1,
              col, ylim, add.alt, alt.ylab, main, ylab, ylog, ...)
```

Arguments

x	a data.frame with Date and var columns of data; x can also have id or name attributes to help give a title for the plot
var	a variable; a column name in x; this can also have attributes of units and long.name to help give a title for the y-axis
width	size of bin; see mkseas

start	starting bin number; e.g., if width="mon" and start=5, the plot will start on "May" at the left-hand side
rep	repetition of the bins in the boxplot
start.day	when width is numeric, this is the starting day of the year for the fist bin, or it can be a Date to specify a month and day (year is ignored)
col	colour for the boxplots; the default is "lightgrey"
ylim	c(min, max) range for y-axis
add.alt	this adds an alternative axis, and is specified by c(slope,inter); for example, if the primary measure is in °C, a secondary scale in K would be c(1,273.15), or in °F would be c(5/9,32); if ylog=TRUE, then this can also be TRUE to display the $\log_{10}$ transformed values of var on the alternative axis
alt.ylab	label for the alternate y-axis (the primary y-axis label is set through attributes for var in x)
main	title for plot; if it is missing, then it will automatically be generated
ylab	y-axis label; if it is missing, then it will automatically be generated
ylog	used to $\log_{10}$ transform values of var for the boxplots; this has a similar but different affect than specifying par(ylog=TRUE) before this function
...	ignored

Details

Shows normals of a seasonal variable using boxplots.

Value

Returns values from boxplot statistics on the variable.

Note

This function was formerly named plot.seas.var, but required renaming as it is not an S3 method.

Author(s)

Mike Toews

See Also

seas.var.plot, plot.seas.norm, year.plot.

Use mksub to make a subset of x.

Examples

```
opar <- par(no.readonly=FALSE)
on.exit(par(opar))
data(mscdata)
dat <- mksub(mscdata, id=1108447)
```

```
seas.var.plot(dat, var="t_max", col="tomato",
  add.alt=c(5/9, 32), alt.ylab="F")
abline(h=0)

par(opar) # reset graphics parameters

seas.var.plot(dat, var="t_min",
  start=18, rep=16)

pdat <- dat[dat$precip > 0,]
attr(pdat$precip, "long.name") <- "precipitation intensity"
attr(pdat$precip, "units") <- "mm/day"

par(ylog=TRUE)
seas.var.plot(pdat, var="precip", col="azure")
title(sub="These boxplots are simply plotted on a log-y scale")

par(opar)

seas.var.plot(pdat, var="precip", col="azure", ylog=TRUE)
title(sub="These boxplots are based on log-transformed values")

seas.var.plot(pdat, var="precip", col="azure", ylog=TRUE, add.alt=TRUE)
title(sub="The actual axis for graph is on the right-side")
```

`SeasOpts` *Options for seas*

Description

Set default options for **seas**.

Usage

```
setSeasOpts()
```

Details

`setSeasOpts` sets all the default values for options in **seas**, and at some point it may support arguments for styles, such as 'black and white'. However, after the initial setting of options, users may change the options to modify the look of graphics produced in **seas**.

Other details of the graphics can be modified using `par`. This includes the font sizes, back-ground colour, font family, and many others. For example, setting `par(cex=0.75)` will reduce the font size in the active device by 75% of the original size; while `par(font.main=2)` will change only the font for the main titles.

Value

`setSeasOpts()` only sets the `options` in the current environment, and returns nothing.

This is automatically done when **seas** is loaded (using `.onLoad`).

Options used in seas

Here are all the supported options for **seas**, with the default values shown for each option. Options are stored in lists, which make them easy to 'get', but difficult to 'set', and is shown in the *Examples* section at the bottom.

`seas.main`: formating style for main title:

- `fmt`: format for `name` and `id` (if available) as the first `"%s"`, followed by a range of years as the second `"%s"`; these are formated by `sprintf`; `"%s\n%s"`
- `rngsep`: separation between ranges of years; `"-"`, other alternatives could be `" to "`
- `show.id`: show id (if available) in main title; `TRUE`
- `show.fun`: show function (where applicable) in main title; `TRUE`

`seas.label`: label formating for variables:

- `fmt`: label for name and units (if available); `"%s (%s)"`, other alternatives could be `"%s, %s"`
- `monthday`: format for month and day (see `strftime` for format codes); this can be either `"%b %-d"` (for most Unix-like systems), `"%b %#d"` (for Windows systems), or `"%b %d"` (for other systems); this should produce a string, such as 'Aug 1' for *August 1st*
- `month` similar as previous, but when starting exactly on month-breaks; `"%B"`
- `ann` a label for `image.seas.sum`; default is 'annual'

`seas.month.grid`: setting for the display of the month grid (see `.seasmonthgrid`), which is common to many plots that use a `numeric` `width` in `mkseas`:

- `abb`: abbreviate month names for grid; `TRUE`
- `len`: trim month name lengths to a number, for instance to get J|F|M|A|M|J|J|A|S|O|N|D, use `1`; `NULL`
- `force`: force the display of each month label using `mtext`, otherwise labels can be automatically placed and adjusted for device using `axis`; `TRUE`
- `label`: show a month label on the grid; `TRUE`
- `col`: colour for month grid; `"lightgrey"`
- `lwd`: width for month grid lines, multiplied by `par("lwd")`; `1`
- `lty`: style for month grid lines; `1`

`seas.bxp`: attributes which affect the display of boxplots, used by various functions:

- `boxcol`: default box-fill colour; `"lightgrey"`
- `outcex`: outlier symbol size, multiplied by `par("cex")`; `1`

`seas.temp`: attributes which affect the display of `seas.temp.plot` (among other functions):

- `col`: colours for boxplot fill and diurnal variability lines; `c("lightgrey","red")`
- `lwd`: width of diurnal variability lines in `seas.temp.plot`, multiplied by `par("lwd")`; `3`

`seas.precip`: attributes which affect the display of precipitation:

- `col`: colour; `"grey"`
- `density`: pattern density; `NULL`
- `angle`: pattern angel; `45`
- `lwd`: thickness of box line, multiplied by `par("lwd")`; `1`

`seas.rain`: attributes which affect the display of rain:

col: colour; "lightblue"

density: pattern density; NULL

angle: pattern angel; 45

lwd: thickness of box line, multiplied by par("lwd"); 1

seas.snow: attributes which affect the display of snow:

col: colour; "lightgrey"

density: pattern density; NULL

angle: pattern angel; -45

lwd: thickness of box line, multiplied by par("lwd"); 1

seas.interarrival: attributes which affect the display of wet- and dry-spells in plot.interarrival; organized as c(wet,dry):

col: colour; c("lightblue","orange")

seas.median: attributes which affect the display of the median lines in image.seas.sum:

col: colour; "red"

lwd: width of line, multiplied by par("lwd"); 1

lty: style of line; 1

seas.mean: attributes which affect the display of the mean lines in image.seas.sum:

col: colour; "red"

lwd: width of line, multiplied by par("lwd"); 1

lty: style of line; 1

seas.na: attributes which affect the display of NA or missing values in various plots:

col: colour; "red"

pch: character symbol; "x"

Author(s)

Mike Toews

See Also

hidden

Examples

```
if(is.null(getOption("seas.main")))
  setSeasOpts()

# Modify an option
getOption("seas.main")$show.id
cp <- orig <- getOption("seas.main")
cp$show.id <- FALSE
options(seas.main=cp)
getOption("seas.main")$show.id

options(seas.main=orig)
```

summerland *Example LARS-WG data file of synthetic data from Summerland, BC*

Description

Example LARS-WG data file of synthetic data from Summerland, BC.

Format

Both files are ASCII-based, and can be viewed in any text editor

- 'summerland.sr' is the 'site file', which contains the meta-data
- 'summerland.dat' is the data file

Details of these file formats can be found in the LARS-WG manual and help documentation.

Details

The sample file name was generated in LARS-WG from calibration of data from Summerland (MSC ID: 1127800). Thirty-years were generated, each synthetic year has 365-days.

Author(s)

Mike Toews

See Also

read.lars, which contains an example using these files

write.help *Write climate data in the format used by the HELP model*

Description

Write climate data in the format used by the Hydrological Evaluation of Landfill Performance (HELP) model. This exports the data using two slightly different variants of HELP: the DOS versions (3.07 to 3.80D) and for Visual HELP.

Usage

```
write.help(file, dat, var = "", name = "", region, lat,
           visual.help = FALSE, metric = TRUE)
```

Arguments

file	name of output file; [DOS] HELP uses extensions '*.D4', '*.D7', and '*.D13' for daily precipitation, temperature and solar radiation, respectively; Visual HELP uses the file names '_weather1.dat', '_weather2.dat' and '_weather3.dat' for the same series of variables
dat	data.frame of climate data
var	variable to be exported; must be one of "precip", "t_mean" or "solar"
name	character; location name
region	character; region
lat	numeric; location latitude in decimal degrees
visual.help	logical formats output for Visual HELP; else formated for the DOS HELP versions (default)
metric	logical if using metric units (this only sets a flag, please ensure the data are in either °C, mm/day and MJ/(m^2 · day) or °F, in./day and langleys/day)

Details

This utility function is experimental and has not been extensively tested; please report any errors to me.

HELP requires continuous data; no missing values are allowed.

Data imported from SDSM use a 365-day calendar, and can be approximated using conv365toGregorian.

Author(s)

Mike Toews

References

HELP 3.07 - Original version for the US EPA; free download
http://el.erdc.usace.army.mil/products.cfm?Topic=model&Type=landfill

HELP 3.80D - Developed by Dr. Klaus Berger, University of Hamburg
http://www.geowiss.uni-hamburg.de/i-boden/fsimhelp_e.htm

Visual HELP - Uses a similar underlying code as HELP 3.07, but features a Windows GUI
http://www.swstechnology.com/groundwater-software/unsaturated-zone-modeling/visual-help

See Also

read.msc, read.sdsm, read.lars, conv365toGregorian

`year.length` *Calculate the number of days in a year*

Description

Determines the number of days per year using a given calendar.

Usage

```
year.length(x, calendar)
```

Arguments

`x`	year or a `Date`
`calendar`	calendar, see details

Details

The number of days per year depends on the choice of calendar. Calendar names used in the function are the same defined for the CF conventions, used for netCDF files. If a calendar is not specified (or `NULL`), then it is assumed to be a proleptic Gregorian calendar (which extends before 1582-10-15). Other accepted calendars are:

- `"360"`: always 360-days per year
- `"365_day"` or `"noleap"`: always 365-days per year
- `"366"` or `"all_leap"`: always 366-days per year
- `"julian"`: 366 days on years divisible by 4, otherwise 365 days

Value

Returns a vector the same length as `x` with the numbers of days corresponding to each year.

Author(s)

Mike Toews

References

http://www.cgd.ucar.edu/cmw/eaton/cf-metadata/CF-current.html#cal

See Also

`mkseas`, `mkann`

Examples

```
cal <- data.frame(year=c(1899, 1900, 1904, 2000, 2080, 2100))
cal[["Gregorian"]] <- year.length(cal$year)
cal[["Julian"]] <- year.length(cal$year, "julian")
cal[["360_day"]] <- year.length(cal$year, "360_day")
cal[["365_day"]] <- year.length(cal$year, "365_day")
cal[["366_day"]] <- year.length(cal$year, "366_day")

cal
```

`year.ploy` *Plot annual temperature and precipitation statistics*

Description

Plots a continuous set of annual temperature and precipitation statistics for a single climate station.

Usage

```
year.plot(x, start.day = 1, precip.only = FALSE, precip.ylim,
          temp.ylim, na.cut = 10, ...)
```

Arguments

`x`	a `data.frame` of climate data
`start.day`	starting day of annum; either a `Date` or an integer day of year; this influences the statistics for each year or annum; such as annual precipitation sums
`precip.only`	only precipitation data is used; rain and snow ignored
`precip.ylim`	range for precipitation graph
`temp.ylim`	range for temperature graph
`na.cut`	minimum number of missing data points in a year to make it void; temperature and precipitation are treated independently
`...`	ignored

Details

This simply shows temperature using (boxplots) and annual precipitation totals. The red bars are directly proportional to the fraction of missing (or `NA`) values for the year; statistics not shown if there are more than `na.cut` `NA` values in a given year.

Note

This function was formerly named `plot.year`, but required renaming as it is not an S3 method.

Author(s)

Mike Toews

See Also

mscdata, seas.temp.plot, plot.seas.norm (can be used for precipitation normals), calculate statistics with tapply

Examples

```
data(mscdata)

year.plot(mksub(mscdata, id=1108447))

year.plot(mksub(mscdata, id=1108447,
                start=as.Date("1975-08-01"),
                end=as.Date("2004-07-31")),
          start.day=as.Date("2000-08-01"))
```

Index