Package: zoo (via r-universe)

October 12, 2024

Version 1.8-13

Date 2023-06-05

Title S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)

Description An S3 class with methods for totally ordered indexed observations. It is particularly aimed at irregular time series of numeric vectors/matrices and factors. zoo's key design goals are independence of a particular index/date/time class and consistency with ts and base R by providing methods to extend standard generics.

Depends R (>= 3.1.0), stats

Suggests AER, coda, chron, ggplot2 (>= 3.0.0), mondate, scales, stinepack, strucchange, timeDate, timeSeries, tis, tseries, xts

Imports utils, graphics, grDevices, lattice (>= 0.20-27)

License GPL-2 | GPL-3

URL https://zoo.R-Forge.R-project.org/

Repository https://r-forge.r-universe.dev

RemoteUrl https://github.com/r-forge/zoo

RemoteRef HEAD

RemoteSha 1e8071831a1308b678c3b59fb5bd68a81b74daf8

Contents

`aggregate.zoo` *Compute Summary Statistics of zoo Objects*

Description

Splits a `"zoo"` object into subsets along a coarser index grid, computes summary statistics for each, and returns the reduced `"zoo"` object.

Usage

```
## S3 method for class 'zoo'
aggregate(x, by, FUN = sum, ...,
  regular = NULL, frequency = NULL, coredata = TRUE)
```

Arguments

`x`	an object of class `"zoo"`.
`by`	index vector of the same length as `index(x)` which defines aggregation groups and the new index to be associated with each group. If `by` is a function, then it is applied to `index(x)` to obtain the aggregation groups.
`FUN`	a function to compute the summary statistics which can be applied to all subsets. Always needs to return a result of fixed length (typically scalar).

...	further arguments passed to FUN.
regular	logical. Should the aggregated series be coerced to class "zooreg" (if the series is regular)? The default is FALSE for "zoo" series and TRUE for "zooreg" series.
frequency	numeric indicating the frequency of the aggregated series (if a "zooreg" series should be returned. The default is to determine the frequency from the data if regular is TRUE. If frequency is specified, it sets regular to TRUE. See examples for illustration.
coredata	logical. Should only the coredata(x) be passed to every by group? If set to FALSE the full zoo series is used.

Value

An object of class "zoo" or "zooreg".

Note

The xts package functions endpoints, period.apply to.period, to.weekly, to.monthly, etc., can also directly input and output certain zoo objects and so can be used for aggregation tasks in some cases as well.

See Also

zoo

Examples

```
## averaging over values in a month:
# x.date is jan 1,3,5,7; feb 9,11,13; mar 15,17,19
x.date <- as.Date(paste(2004, rep(1:4, 4:1), seq(1,20,2), sep = "-")); x.date
x <- zoo(rnorm(12), x.date); x
# coarser dates - jan 1 (4 times), feb 1 (3 times), mar 1 (3 times)
x.date2 <- as.Date(paste(2004, rep(1:4, 4:1), 1, sep = "-")); x.date2
x2 <- aggregate(x, x.date2, mean); x2
# same - uses as.yearmon
x2a <- aggregate(x, as.Date(as.yearmon(time(x))), mean); x2a
# same - uses by function
x2b <- aggregate(x, function(tt) as.Date(as.yearmon(tt)), mean); x2b
# same - uses cut
x2c <- aggregate(x, as.Date(cut(time(x), "month")), mean); x2c
# almost same but times of x2d have yearmon class rather than Date class
x2d <- aggregate(x, as.yearmon, mean); x2d

# compare time series
plot(x)
lines(x2, col = 2)

## aggregate a daily time series to a quarterly series
# create zoo series
tt <- as.Date("2000-1-1") + 0:300
z.day <- zoo(0:300, tt)
```

```
# function which returns corresponding first "Date" of quarter
first.of.quarter <- function(tt) as.Date(as.yearqtr(tt))

# average z over quarters
# 1. via "yearqtr" index (regular)
# 2. via "Date" index (not regular)
z.qtr1 <- aggregate(z.day, as.yearqtr, mean)
z.qtr2 <- aggregate(z.day, first.of.quarter, mean)

# The last one used the first day of the quarter but suppose
# we want the first day of the quarter that exists in the series
# (and the series does not necessarily start on the first day
# of the quarter).
z.day[!duplicated(as.yearqtr(time(z.day)))]

# This is the same except it uses the last day of the quarter.
# It requires R 2.6.0 which introduced the fromLast= argument.
## Not run:
z.day[!duplicated(as.yearqtr(time(z.day)), fromLast = TRUE)]

## End(Not run)

# The aggregated series above are of class "zoo" (because z.day
# was "zoo"). To create a regular series of class "zooreg",
# the frequency can be automatically chosen
zr.qtr1 <- aggregate(z.day, as.yearqtr, mean, regular = TRUE)
# or specified explicitely
zr.qtr2 <- aggregate(z.day, as.yearqtr, mean, frequency = 4)


## aggregate on month and extend to monthly time series
if(require(chron)) {
y <- zoo(matrix(11:15, nrow = 5, ncol = 2), chron(c(15, 20, 80, 100, 110)))
colnames(y) <- c("A", "B")

# aggregate by month using first of month as times for coarser series
# using first day of month as repesentative time
y2 <- aggregate(y, as.Date(as.yearmon(time(y))), head, 1)

# fill in missing months by merging with an empty series containing
# a complete set of 1st of the months
yrt2 <- range(time(y2))
y0 <- zoo(,seq(from = yrt2[1], to = yrt2[2], by = "month"))
merge(y2, y0)
}

# given daily series keep only first point in each month at
# day 21 or more
z <- zoo(101:200, as.Date("2000-01-01") + seq(0, length = 100, by = 2))
zz <- z[as.numeric(format(time(z), "%d")) >= 21]
zz[!duplicated(as.yearmon(time(zz)))]

# same except times are of "yearmon" class
```

```
aggregate(zz, as.yearmon, head, 1)

# aggregate POSIXct seconds data every 10 minutes
Sys.setenv(TZ = "GMT")
tt <- seq(10, 2000, 10)
x <- zoo(tt, structure(tt, class = c("POSIXt", "POSIXct")))
aggregate(x, time(x) - as.numeric(time(x)) %% 600, mean)

# aggregate weekly series to a series with frequency of 52 per year
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
z <- zooreg(1:100 + rnorm(100), start = as.Date("2001-01-01"), deltat = 7)

# new.freq() converts dates to a grid of freq points per year
# yd is sequence of dates of firsts of years
# yy is years of the same sequence
# last line interpolates so dates, d, are transformed to year + frac of year
# so first week of 2001 is 2001.0, second week is 2001 + 1/52, third week
# is 2001 + 2/52, etc.
new.freq <- function(d, freq = 52) {
       y <- as.Date(cut(range(d), "years")) + c(0, 367)
       yd <- seq(y[1], y[2], "year")
       yy <- as.numeric(format(yd, "%Y"))
       floor(freq * approx(yd, yy, xout = d)$y) / freq
}

# take last point in each period
aggregate(z, new.freq, tail, 1)

# or, take mean of all points in each
aggregate(z, new.freq, mean)

# example of taking means in the presence of NAs
z.na <- zooreg(c(1:364, NA), start = as.Date("2001-01-01"))
aggregate(z.na, as.yearqtr, mean, na.rm = TRUE)

# Find the sd of all days that lie in any Jan, all days that lie in
# any Feb, ..., all days that lie in any Dec (i.e. output is vector with
# 12 components)
aggregate(z, format(time(z), "%m"), sd)
```

`as.zoo` *Coercion from and to zoo*

Description

Methods for coercing `"zoo"` objects to other classes and a generic function `as.zoo` for coercing objects to class `"zoo"`.

Usage

```
as.zoo(x, ...)
```

Arguments

x	an object,
...	further arguments passed to zoo when the return object is created.

Details

as.zoo currently has a default method and methods for ts, fts (currently archived on CRAN), irts, mcmc, tis, xts objects (and zoo objects themselves).

Methods for coercing objects of class "zoo" to other classes currently include: as.ts, as.matrix, as.vector, as.data.frame, as.list (the latter also being available for "ts" objects). Furthermore, fortify.zoo can transform "zoo" series to "data.frame" including the time index and optionally melting a wide series into a long data frame.

In the conversion between zoo and ts, the zooreg class is always used.

Value

as.zoo returns a zoo object.

See Also

zoo, fortify.zoo, zooreg, ts, irts, tis, mcmc, xts.

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## coercion to zoo:
## default method
as.zoo(rnorm(5))
## method for "ts" objects
as.zoo(ts(rnorm(5), start = 1981, freq = 12))

## coercion from zoo:
x.date <- as.POSIXct(paste("2003-", rep(1:4, 4:1), "-", sample(1:28, 10, replace = TRUE), sep = ""))
x <- zoo(matrix(rnorm(24), ncol = 2), x.date)
as.matrix(x)
as.vector(x)
as.data.frame(x)
as.list(x)
```

coredata *Extracting/Replacing the Core Data of Objects*

Description

Generic functions for extracting the core data contained in a (more complex) object and replacing it.

Usage

```
coredata(x, ...)
coredata(x) <- value
```

Arguments

x	an object.
...	further arguments passed to methods.
value	a suitable value object for use with x.

Value

In zoo, there are currently coredata methods for time series objects of class "zoo", "ts", "its", "irts", all of which strip off the index/time attributes and return only the observations. The are also corresponding replacement methods for these classes.

See Also

zoo

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
x <- zoo(matrix(rnorm(20), ncol = 2), x.date)

## the full time series
x
## and only matrix of observations
coredata(x)

## change the observations
coredata(x) <- matrix(1:20, ncol = 2)
x
```

frequency<- *Replacing the Index of Objects*

Description

Generic function for replacing the frequency of an object.

Usage

```
frequency(x) <- value
```

Arguments

x	an object.
value	a frequency.

Details

`frequency<-` is a generic function for replacing (or assigning) the frequency of an object. Currently, there is a `"zooreg"` and a `"zoo"` method. In both cases, the `value` is assigned to the `"frequency"` of the object if it complies with the `index(x)`.

See Also

zooreg, index

Examples

```
z <- zooreg(1:5)
z
as.ts(z)
frequency(z) <- 3
z
as.ts(z)
```

ggplot2.zoo *Convenience Functions for Plotting zoo Objects with ggplot2*

Description

`fortify.zoo` takes a zoo object and converts it into a data frame (intended for ggplot2). `autoplot.zoo` takes a zoo object and returns a ggplot2 object. It essentially uses the mapping `aes(x = Time, y = Value, group = Series)` and adds `colour = Series` in the case of a multivariate series with `facets = NULL`.

Usage

```
  ## S3 method for class 'zoo'
autoplot(object, geom = "line", facets, ...)
  ## S3 method for class 'zoo'
fortify(model, data,
 names = c("Index", "Series", "Value"),
 melt = FALSE, sep = NULL, ...)
  facet_free(facets = Series ~ ., margins = FALSE, scales = "free_y", ...)

  yearmon_trans(format = "%b %Y", n = 5)
  scale_x_yearmon(..., format = "%b %Y", n = 5)
  scale_y_yearmon(..., format = "%b %Y", n = 5)

  yearqtr_trans(format = "%Y-%q", n = 5)
  scale_x_yearqtr(..., format = "%Y-%q", n = 5)
  scale_y_yearqtr(..., format = "%Y-%q", n = 5)
```

Arguments

object	an object of class "zoo".
geom	character (e.g., "line") or function (e.g., geom_line) specifying which geom to use.
facets	specification of facets for facet_grid. The default in the autoplot method is to use facets = NULL for univariate series and facets = Series ~ . for multivariate series.
...	further arguments passed to aes for autoplot (e.g., linetype = Series and/or shape = Series). For fortify the arguments are not used. For the scale_*_* functions the arguments are passed on to scale_*_continuous.
model	an object of class "zoo" to be converted to a "data.frame".
data	not used (required by generic fortify method).
names	(list of) character vector(s). New names given to index/time column, series indicator (if melted), and value column (if melted). If only a subset of characters should be changed, either NAs can be used or a named vector.
sep	If specified then the Series column is split into multiple columns using sep as the split character.
melt	Should the resulting data frame be in long format (melt = TRUE) or wide format (melt = FALSE).
margins	As in facet_grid.
scales	As in facet_grid except it defaults to "free_y".
format	A format acceptable to format.yearmon or format.yearqtr.
n	Approximate number of axis ticks.

Details

Convenience interface for visualizing zoo objects with ggplot2. `autoplot.zoo` uses `fortify.zoo` (with `melt = TRUE`) to convert the zoo object into a data frame and then uses a suitable `aes()` mapping to visiualize the series.

Value

`fortify.zoo` returns a `data.frame` either in long format (`melt = TRUE`) or in wide format (`melt = FALSE`). The long format has three columns: the time `Index`, a factor indicating the `Series`, and the corresponding `Value`. The wide format simply has the time `Index` plus all columns of `coredata(model)`.

`autoplot.zoo` returns a `ggplot` object.

Author(s)

Trevor L. Davis <trevor.l.davis@gmail.com>, Achim Zeileis

See Also

autoplot, fortify, ggplot

Examples

```
if(require("ggplot2") && require("scales")) {
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## example data
x.Date <- as.Date(paste(2003, 02, c(1, 3, 7, 9, 14), sep = "-"))
x <- zoo(rnorm(5), x.Date)
xlow <- x - runif(5)
xhigh <- x + runif(5)
z <- cbind(x, xlow, xhigh)

## univariate plotting
autoplot(x)
## by hand
ggplot(aes(x = Index, y = Value), data = fortify(x, melt = TRUE)) +
  geom_line() + xlab("Index") + ylab("x")
## adding series one at a time
last_plot() + geom_line(aes(x = Index, y = xlow), colour = "red", data = fortify(xlow))
## add ribbon for high/low band
ggplot(aes(x = Index, y = x, ymin = xlow, ymax = xhigh), data = fortify(x)) +
  geom_ribbon(fill = "darkgray") + geom_line()

## multivariate plotting in multiple or single panels
autoplot(z)                      ## multiple without color/linetype
autoplot(z, facets = Series ~ .) ## multiple with series-dependent color/linetype
autoplot(z, facets = NULL)       ## single with series-dependent color/linetype
## by hand with color/linetype and with/without facets
ggz <- ggplot(aes(x = Index, y = Value, group = Series, colour = Series, linetype = Series),
```

```
  data = fortify(z, melt = TRUE)) + geom_line() + xlab("Index") + ylab("")
ggz
ggz + facet_grid(Series ~ .)
## variations
autoplot(z, geom = "point")
autoplot(z, facets = NULL) + geom_point()
autoplot(z, facets = NULL) + scale_colour_grey() + theme_bw()

## for "ts" series via coercion
autoplot(as.zoo(EuStockMarkets))
autoplot(as.zoo(EuStockMarkets), facets = NULL)

autoplot(z) +
 aes(colour = NULL, linetype = NULL) +
 facet_grid(Series ~ ., scales = "free_y")

autoplot(z) + aes(colour = NULL, linetype = NULL) + facet_free() # same

z.yq <- zooreg(rnorm(50), as.yearqtr("2000-1"), freq = 4)
autoplot(z.yq)

## mimic matplot
data <- cbind(A = c(6, 1, NA, NA), B = c(16, 4, 1, NA), C = c(25, 7, 2, 1))
autoplot(zoo(data), facet = NULL) + geom_point()
## with different line types
autoplot(zoo(data), facet = NULL) + geom_point() + aes(linetype = Series)

## illustrate just fortify() method
z <- zoo(data)
fortify(z)
fortify(z, melt = TRUE)
fortify(z, melt = TRUE, names = c("Time", NA, "Data"))
fortify(z, melt = TRUE, names = c(Index = "Time"))

## with/without splitting
z <- zoo(cbind(a.A = 1:2, a.B = 2:3, b.A = 3:4, c.B = 4:5))
fortify(z)
fortify(z, melt = TRUE, sep = ".", names = list(Series = c("Lower", "Upper")))

## scale_x_yearmon with custom discrete breaks
df <-  data.frame(dates = as.yearmon("2018-08") + 0:6/12, values = c(2:6, 0, 1))
ggdf <- ggplot(df, aes(x = dates, y = values)) +
  geom_bar(position = "dodge", stat = "identity") + theme_light() +
  xlab("Month") + ylab("Values")
ggdf ## with default scale_x_yearmon
ggdf + scale_x_yearmon(breaks = df$dates) ## with custom discrete breaks
}
```

---

index *Extracting/Replacing the Index of Objects*

Description

Generic functions for extracting the index of an object and replacing it.

Usage

```
index(x, ...)
index(x) <- value
```

Arguments

x	an object.
...	further arguments passed to methods.
value	an ordered vector of the same length as the "index" attribute of x.

Details

index is a generic function for extracting the index of objects, currently it has a default method and a method for zoo objects which is the same as the time method for zoo objects. Another pair of generic functions provides replacing the index or time attribute. Methods are available for "zoo" objects only, see examples below.

The start and end of the index/time can be queried by using the methods of start and end.

See Also

time, zoo

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

x.date <- as.Date(paste(2003, 2, c(1, 3, 7, 9, 14), sep = "-"))
x <- zoo(rnorm(5), x.date)

## query index/time of a zoo object
index(x)
time(x)

## change class of index from Date to POSIXct
## relative to current time zone
x
index(x) <- as.POSIXct(format(time(x)),tz="")
x

## replace index/time of a zoo object
index(x) <- 1:5
x
time(x) <- 6:10
x
```

```
## query start and end of a zoo object
start(x)
end(x)

## query index of a usual matrix
xm <- matrix(rnorm(10), ncol = 2)
index(xm)
```

`is.regular` *Check Regularity of a Series*

Description

`is.regular` is a regular function for checking whether a series of ordered observations has an underlying regularity or is even strictly regular.

Usage

```
is.regular(x, strict = FALSE)
```

Arguments

`x`	an object (representing a series of ordered observations).
`strict`	logical. Should strict regularity be checked? See details.

Details

A time series can either be irregular (unequally spaced), strictly regular (equally spaced) or have an underlying regularity, i.e., be created from a regular series by omitting some observations. Here, the latter property is called *regular*. Consequently, regularity follows from strict regularity but not vice versa.

`is.regular` is a generic function for checking regularity (default) or strict regularity. Currently, it has methods for `"ts"` objects (which are always strictly regular), `"zooreg"` objects (which are at least regular), `"zoo"` objects (which can be either irregular, regular or even strictly regular) and a default method. The latter coerces `x` to `"zoo"` before checking its regularity.

Value

A logical is returned indicating whether `x` is (strictly) regular.

See Also

`zooreg`, `zoo`

Examples

```
## checking of a strictly regular zoo series
z <- zoo(1:10, seq(2000, 2002.25, by = 0.25), frequency = 4)
z
class(z)
frequency(z) ## extraction of frequency attribute
is.regular(z)
is.regular(z, strict = TRUE)
## by omitting observations, the series is not strictly regular
is.regular(z[-3])
is.regular(z[-3], strict = TRUE)

## checking of a plain zoo series without frequency attribute
## which is in fact regular
z <- zoo(1:10, seq(2000, 2002.25, by = 0.25))
z
class(z)
frequency(z) ## data driven computation of frequency
is.regular(z)
is.regular(z, strict = TRUE)
## by omitting observations, the series is not strictly regular
is.regular(z[-3])
is.regular(z[-3], strict = TRUE)

suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## checking of an irregular zoo series
z <- zoo(1:10, rnorm(10))
z
class(z)
frequency(z) ## attempt of data-driven frequency computation
is.regular(z)
is.regular(z, strict = TRUE)
```

`lag.zoo` *Lags and Differences of zoo Objects*

Description

Methods for computing lags and differences of "zoo" objects.

Usage

```
## S3 method for class 'zoo'
lag(x, k = 1, na.pad = FALSE, ...)
## S3 method for class 'zoo'
diff(x, lag = 1, differences = 1, arithmetic = TRUE, na.pad = FALSE, log = FALSE, ...)
```

Arguments

x	a "zoo" object.
k, lag	For lag the number of lags (in units of observations). Note the sign of k behaves as in lag. For diff it is the number of backward lags used (or if negative the number of forward lags.
differences	an integer indicating the order of the difference.
arithmetic	logical. Should arithmetic (or geometric) differences be computed?
na.pad	logical. If TRUE it adds any times that would not otherwise have been in the result with a value of NA. If FALSE those times are dropped.
log	logical. Should the differences of the log series be computed?
...	currently not used.

Details

These methods for "zoo" objects behave analogously to the default methods. The only additional arguments are arithmetic and log in diff and na.pad in both lag.zoo and diff.zoo. Also, "k" can be a vector of lags in which case the names of "k", if any, are used in naming the result.

Value

The lagged or differenced "zoo" object.

Note

Note the sign of k: a series lagged by a positive k is shifted *earlier* in time.

lag.zoo and lag.zooreg can give different results. For a lag of 1 lag.zoo moves points to the adjacent time point whereas lag.zooreg moves the time by deltat. This implies that a point in a zoo series cannot be lagged to a time point that is not already in the series whereas this is possible for a zooreg series.

See Also

zoo, lag, diff

Examples

```
x <- zoo(11:21)

lag(x, k = 1)
lag(x, k = -1)
# this pairs each value of x with the next or future value
merge(x, lag1 = lag(x, k=1))
diff(x^3)
diff(x^3, -1)
diff(x^3, na.pad = TRUE)
```

lagts *Lags and Leads for Time Series Objects*

Description

lagts and leadts are new generic functions that do intuitive computations of lags and leads. The main purpose is to overcome the confusion about the sign of the lag order in the base lag function that is employed differently by different classes.

Usage

```
lagts(x, ...)
leadts(x, ...)

## Default S3 method:
lagts(x, k = 1, na.pad = TRUE, ...)
## S3 method for class 'ts'
lagts(x, k = 1, na.pad = TRUE, ...)
## S3 method for class 'zoo'
lagts(x, k = 1, na.pad = TRUE, ...)
```

Arguments

x	an object. The default method works for vectors and matrices.
k	For lag the number of lags (in units of observations). Note the sign of k behaves as employed in most textbooks and *not* as in lag.
na.pad	logical. If TRUE it adds any times that would not otherwise have been in the result with a value of NA. If FALSE those times are dropped.
...	currently not used.

Details

lagts by default computes real lags for positive k while lag would in fact compute leads. The latter lead to a lot of confusion among users of lag, especially because some time series classes followed the base behaviour while others decided for the intuitive (but inconsistent) behaviour. lagts aims to overcome this confusions

This functionality is still under development.

Value

The lags or leads of the original object.

See Also

lag

Examples

```
x <- zoo(11:21)
lag(x, k = 1)
lag(x, k = -1)
lagts(x, k = 1)
lagts(x, k = -1) ## leadts not written yet
```

`listindex` *An Index Class for Testing the Generality of zoo*

Description

`"listindex"` is a deliberately awkward class, set up only to test **zoo**, especially in conversion to C. It is not intended for users.

Usage

```
listindex(object, ...)
```

Arguments

`object`	any object that could be used as an index for a `"zoo"` object.
`...`	currently not used.

Details

The `"listindex"` class does nothing else than wrapping its `object` into a list with a single element called `index`. This assures that none of the usual methods required by **zoo** work sensibly out of the box.

Then, the required methods `c()`, `[`, `length`, `ORDER`, and `MATCH` are set up. These simply work by emulating a dispatch on `listindex$index`.

Additionally, methods for `as.numeric`, `as.vector`, and `as.character` are required as well. These are sort of expected by **zoo** in some places.

Finally, adding `print` and `xtfrm` methods helps in other places as well.

Value

Returns `structure(list(index = object), class = "listindex")`, i.e., a list of class `"listindex"` with a single element `index` containing the unchanged input `object`.

See Also

`zoo`

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## two listindex objects wrapping Date vectors
x <- listindex(as.Date(0) + 0:2)
y <- listindex(as.Date(0) + 3:5)

## define zoo objects based on them
z1 <- zoo(rnorm(4), c(x, y[1]))
z2 <- zoo(rnorm(4), c(x[3], y))

## check some simple operations
z1 + z2
lag(z1, -1)
merge(z1, z2)
c(z1[1:2], z2[4])
```

---

make.par.list *Make a List from a Parameter Specification*

Description

Process parameters so that a list of parameter specifications is returned (used by `plot.zoo` and `xyplot.zoo`).

Usage

```
make.par.list(nams, x, n, m, def, recycle = sum(unnamed) > 0)
```

Arguments

`nams`	character vector with names of variables.
`x`	list or vector of parameter specifications, see details.
`n`	numeric, number of rows.
`m`	numeric, number of columns. (Only determines whether `m` is 1 or greater than 1.
`def`	default parameter value.
`recycle`	logical. If `TRUE` recycle columns to provide unspecified ones. If `FALSE` use `def` to provide unspecified ones. This only applies to entire columns. Within columns recycling is always done regardless of how `recycle` is set. Defaults to `TRUE` if there is at least one unnamed variable and defaults to `FALSE` if there are only named variables in `x`.

Details

This function is currently intended for internal use. It is currently used by `plot.zoo` and `xyplot.zoo` but might also be used in the future to create additional new plotting routines. It creates a new list which uses the named variables from `x` and then assigns the unnamed in order. For the remaining variables assign them the default value if `!recycle` or recycle the unnamed variables if `recycle`.

Value

A list of parameters, see details.

Examples

```
make.par.list(letters[1:5], 1:5, 3, 5)
suppressWarnings( make.par.list(letters[1:5], 1:4, 3, 5, 99) )
make.par.list(letters[1:5], c(d=3), 3, 5, 99)
make.par.list(letters[1:5], list(d=1:2, 99), 3, 5)
make.par.list(letters[1:5], list(d=1:2, 99, 100), 3, 5)
```

MATCH *Value Matching*

Description

MATCH is a generic function for value matching.

Usage

```
MATCH(x, table, nomatch = NA, ...)
## S3 method for class 'times'
MATCH(x, table, nomatch = NA, units = "sec", eps = 1e-10, ...)
```

Arguments

x	an object.
table	the values to be matched against.
nomatch	the value to be returned in the case when no match is found. Note that it is coerced to integer.
units	See trunc.times.
eps	See trunc.times.
...	further arguments to be passed to methods.

Details

MATCH is a new generic function which aims at providing the functionality of the non-generic base function match for arbitrary objects. Currently, there is a default method which simply calls match and various methods for time/date objects.

The MATCH method for Date objects coerces the table to Date as well (if necessary) and then uses match(unclass(x), unclass(table), ...). Similarly, the MATCH methods for POSIXct, POSIXlt, and timeDate coerce both x and table to POSIXct and then match the unclassed objects.

MATCH.times is used for chron objects. x will match any time in table less than units away.

See Also

match

Examples

```
MATCH(1:5, 2:3)
```

---

merge.zoo *Merge Two or More zoo Objects*

Description

Merge two zoo objects by common indexes (times), or do other versions of database *join* operations.

Usage

```
## S3 method for class 'zoo'
merge(..., all = TRUE, fill = NA, suffixes = NULL,
  check.names = FALSE, retclass = c("zoo", "list", "data.frame"),
  drop = TRUE, sep = ".")
```

Arguments

...	two or more objects, usually of class "zoo".
all	logical vector having the same length as the number of "zoo" objects to be merged (otherwise expanded).
fill	an element for filling gaps in merged "zoo" objects (if any).
suffixes	character vector of the same length as the number of "zoo" objects specifying the suffixes to be used for making the merged column names unique.
check.names	See read.table.
retclass	character that specifies the class of the returned result. It can be "zoo" (the default), "list" or NULL. For details see below.
drop	logical. If a "zoo" object without observations is merged with a one-dimensional "zoo" object (vector or 1-column matrix), should the result be a vector (drop = TRUE) or a 1-column matrix (drop = FALSE)? The former is the default in the Merge method, the latter in the cbind method.
sep	character. Separator character that should be used when pasting suffixes to column names for making them unique.

Details

The merge method for "zoo" objects combines the columns of several objects along the union of the dates for all = TRUE, the default, or the intersection of their dates for all = FALSE filling up the created gaps (if any) with the fill pattern.

The first argument must be a zoo object. If any of the remaining arguments are plain vectors or matrices with the same length or number of rows as the first argument then such arguments are coerced to "zoo" using as.zoo. If they are plain but have length 1 then they are merged after all non-scalars such that their column is filled with the value of the scalar.

all can be a vector of the same length as the number of "zoo" objects to merged (if not, it is expanded): All indexes (times) of the objects corresponding to TRUE are included, for those corresponding to FALSE only the indexes present in all objects are included. This allows intersection, union and left and right joins to be expressed.

If retclass is "zoo" (the default) a single merged "zoo" object is returned. If it is set to "list" a list of "zoo" objects is returned. If retclass = NULL then instead of returning a value it updates each argument (if it is a variable rather than an expression) in place so as to extend or reduce it to use the common index vector.

The indexes of different "zoo" objects can be of different classes and are coerced to one class in the resulting object (with a warning).

The default cbind method is essentially the default merge method, but does not support the retclass argument. The rbind method combines the dates of the "zoo" objects (duplicate dates are not allowed) and combines the rows of the objects. Furthermore, the c method is identical to the rbind method.

Value

An object of class "zoo" if retclass="zoo", an object of class "list" if retclass="list" or modified arguments as explained above if retclass=NULL. If the result is an object of class "zoo" then its frequency is the common frequency of its zoo arguments, if they have a common frequency.

See Also

zoo

Examples

```
## simple merging
x.date <- as.Date(paste(2003, 02, c(1, 3, 7, 9, 14), sep = "-"))
x <- zoo(rnorm(5), x.date)

y1 <- zoo(matrix(1:10, ncol = 2), 1:5)
y2 <- zoo(matrix(rnorm(10), ncol = 2), 3:7)

## using arguments `fill' and `suffixes'
merge(y1, y2, all = FALSE)
merge(y1, y2, all = FALSE, suffixes = c("a", "b"))
merge(y1, y2, all = TRUE)
merge(y1, y2, all = TRUE, fill = 0)
```

```
## if different index classes are merged, as in
## the next merge example then ## a warning is issued and
### the indexes are coerced.
## It is up to the user to ensure that the result makes sense.
merge(x, y1, y2, all = TRUE)

## extend an irregular series to a regular one:
# create a constant series
z <- zoo(1, seq(4)[-2])
# create a 0 dimensional zoo series
z0 <- zoo(, 1:4)
# do the extension
merge(z, z0)
# same but with zero fill
merge(z, z0, fill = 0)

merge(z, coredata(z),  1)


## merge multiple series represented in a long form data frame
## into a multivariate zoo series and plot, one series for each site.
## Additional examples can be found here:
## https://stat.ethz.ch/pipermail/r-help/2009-February/187094.html
## https://stat.ethz.ch/pipermail/r-help/2009-February/187096.html
##
m <- 5 # no of years
n <- 6 # no of sites
sites <- LETTERS[1:n]
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
DF <- data.frame(site = sites, year = 2000 + 1:m, data = rnorm(m*n))
tozoo <- function(x) zoo(x$data, x$year)
Data <- do.call(merge, lapply(split(DF, DF$site), tozoo))
plot(Data, screen = 1, col = 1:n, pch = 1:n, type = "o", xlab = "")
legend("bottomleft", legend = sites, lty = 1, pch = 1:n, col = 1:n)

## for each index value in x merge it with the closest index value in y
## but retaining x's times.
x<-zoo(1:3,as.Date(c("1992-12-13", "1997-05-12", "1997-07-13")))
y<-zoo(1:5,as.Date(c("1992-12-15", "1992-12-16", "1997-05-10","1997-05-19", "1997-07-13")))
f <- function(u) which.min(abs(as.numeric(index(y)) - as.numeric(u)))
ix <- sapply(index(x), f)
cbind(x, y = coredata(y)[ix])

## this merges each element of x with the closest time point in y at or
## after x's time point (whereas in previous example it could be before
## or after)
window(na.locf(merge(x, y), fromLast = TRUE), index(x))


## c() can combine several zoo series, e.g., zoo series with Date index
z <- zoo(1:5, as.Date("2000-01-01") + 0:4)
z2 <- zoo(6:7, time(z)[length(z)] + 1:2)
```

```
## c() combines these in a single series
c(z, z2)

## the order does not matter
c(z2, z)

## note, however, that combining a zoo series with an unclassed vector
## of observations would try to coerce the indexes first
## which might either give an unexpected result or an error in R >= 4.1.0
## c(z, 6:7)
```

`na.aggregate` *Replace NA by Aggregation*

Description

Generic function for replacing each NA with aggregated values. This allows imputing by the overall mean, by monthly means, etc.

Usage

```
na.aggregate(object, ...)
## Default S3 method:
na.aggregate(object, by = 1, ..., FUN = mean,
             na.rm = FALSE, maxgap = Inf)
```

Arguments

object	an object.
by	a grouping variable corresponding to object, or a function to be applied to time(object) to generate the groups.
...	further arguments passed to by if by is a function.
FUN	function to apply to the non-missing values in each group defined by by.
na.rm	logical. Should any remaining NAs be removed?
maxgap	maximum number of consecutive NAs to fill. Any longer gaps will be left unchanged.

Value

An object in which each NA in the input object is replaced by the mean (or other function) of its group, defined by by. This is done for each series in a multi-column object. Common choices for the aggregation group are a year, a month, all calendar months, etc.

If a group has no non-missing values, the default aggregation function mean will return NaN. Specify na.rm = TRUE to omit such remaining missing values.

See Also

zoo

Examples

```
z <- zoo(c(1, NA, 3:9),
         c(as.Date("2010-01-01") + 0:2,
           as.Date("2010-02-01") + 0:2,
           as.Date("2011-01-01") + 0:2))
## overall mean
na.aggregate(z)
## group by months
na.aggregate(z, as.yearmon)
## group by calendar months
na.aggregate(z, months)
## group by years
na.aggregate(z, format, "%Y")
```

---

na.approx *Replace NA by Interpolation*

Description

Generic functions for replacing each NA with interpolated values.

Usage

```
na.approx(object, ...)
## S3 method for class 'zoo'
na.approx(object, x = index(object), xout, ..., na.rm = TRUE, maxgap = Inf, along)
## S3 method for class 'zooreg'
na.approx(object, ...)
## S3 method for class 'ts'
na.approx(object, ...)
## Default S3 method:
na.approx(object, x = index(object), xout, ..., na.rm = TRUE, maxgap = Inf, along)

na.spline(object, ...)
## S3 method for class 'zoo'
na.spline(object, x = index(object), xout, ..., na.rm = TRUE, maxgap = Inf, along)
## S3 method for class 'zooreg'
na.spline(object, ...)
## S3 method for class 'ts'
na.spline(object, ...)
## Default S3 method:
na.spline(object, x = index(object), xout, ..., na.rm = TRUE, maxgap = Inf, along)
```

Arguments

object	object in which NAs are to be replaced
x, xout	Variables to be used for interpolation as in approx.
na.rm	logical. If the result of the (spline) interpolation still results in leading and/or trailing NAs, should these be removed (using na.trim)?
maxgap	maximum number of consecutive NAs to fill. Any longer gaps will be left unchanged. Note that all methods listed above can accept maxgap as it is ultimately passed to the default method. In na.spline the maxgap argument cannot be combined with xout, though.
along	deprecated.
...	further arguments passed to methods. The n argument of approx is currently not supported.

Details

Missing values (NAs) are replaced by linear interpolation via approx or cubic spline interpolation via spline, respectively.

It can also be used for series disaggregation by specifying xout.

By default the index associated with object is used for interpolation. Note, that if this calls index.default this gives an equidistant spacing 1:NROW(object). If object is a matrix or data.frame, the interpolation is done separately for each column.

If obj is a plain vector then na.approx(obj, x, y, xout, ...) returns approx(x = x[!na], y = coredata(obj)[!na], xout = xout, ...) (where na indicates observations with NA) such that xout defaults to x. Note that if there are less than two non-NAs then approx() cannot be applied and thus no NAs can be replaced.

If obj is a zoo, zooreg or ts object its coredata value is processed as described and its time index is xout if specified and index(obj) otherwise. If obj is two dimensional then the above is applied to each column separately. For examples, see below.

If obj has more than one column, the above strategy is applied to each column.

Value

An object of similar structure as object with NAs replaced by interpolation. For na.approx only the internal NAs are replaced and leading or trailing NAs are omitted if na.rm = TRUE or not replaced if na.rm = FALSE.

See Also

zoo, approx, na.contiguous, na.locf, na.omit, na.trim, spline, stinterp

Examples

```
z <- zoo(c(2, NA, 1, 4, 5, 2), c(1, 3, 4, 6, 7, 8))

## use underlying time scale for interpolation
na.approx(z)
```

```
## use equidistant spacing
na.approx(z, 1:6)

# with and without na.rm = FALSE
zz <- c(NA, 9, 3, NA, 3, 2)
na.approx(zz, na.rm = FALSE)
na.approx(zz)

d0 <- as.Date("2000-01-01")
z <- zoo(c(11, NA, 13, NA, 15, NA), d0 + 1:6)

# NA fill, drop or keep leading/trailing NAs
na.approx(z)
na.approx(z, na.rm = FALSE)

# extrapolate to point outside of range of time points
# (a) drop NA, (b) keep NA, (c) extrapolate using rule = 2 from approx()
na.approx(z, xout = d0 + 7)
na.approx(z, xout = d0 + 7, na.rm = FALSE)
na.approx(z, xout = d0 + 7, rule = 2)

# use splines - extrapolation handled differently
z <- zoo(c(11, NA, 13, NA, 15, NA), d0 + 1:6)
na.spline(z)
na.spline(z, na.rm = FALSE)
na.spline(z, xout = d0 + 1:6)
na.spline(z, xout = d0 + 2:5)
na.spline(z, xout = d0 + 7)
na.spline(z, xout = d0 + 7, na.rm = FALSE)

## using na.approx for disaggregation
zy <- zoo(1:3,  2000:2001)

# yearly to monthly series
zmo <- na.approx(zy, xout = as.yearmon(2000+0:13/12))
zmo

# monthly to daily series
sq <- seq(as.Date(start(zmo)), as.Date(end(zmo), frac = 1), by = "day")
zd <- na.approx(zmo, x = as.Date, xout = sq)
head(zd)

# weekly to daily series
zww <- zoo(1:3, as.Date("2001-01-01") + seq(0, length = 3, by = 7))
zww
zdd <- na.approx(zww, xout = seq(start(zww), end(zww), by = "day"))
zdd

# The lines do not show up because of the NAs
plot(cbind(z, z), type = "b", screen = 1)
# use na.approx to force lines to appear
plot(cbind(z, na.approx(z)), type = "b", screen = 1)
```

```
# Workaround where less than 2 NAs can appear in a column
za <- zoo(cbind(1:5, NA, c(1:3, NA, 5), NA)); za

ix <- colSums(!is.na(za)) > 0
za[, ix] <- na.approx(za[, ix]); za

# using na.approx to create regularly spaced series
# z has points at 10, 20 and 40 minutes while output also has a point at 30
if(require("chron")) {
  tt <- as.chron("2000-01-01 10:00:00") + c(1, 2, 4) * as.numeric(times("00:10:00"))
  z <- zoo(1:3, tt)
  tseq <- seq(start(z), end(z), by = times("00:10:00"))
  na.approx(z, xout = tseq)
}
```

`na.fill` *Fill NA or specified positions.*

Description

Generic function for filling `NA` values or specified positions.

Usage

```
na.fill(object, fill, ...)
## S3 method for class 'ts'
na.fill(object, fill, ix, ...)
## S3 method for class 'zoo'
na.fill(object, fill, ix, ...)
## Default S3 method:
na.fill(object, fill, ix, ...)

na.fill0(object, fill, ix = !is.na(object))
```

Arguments

`object`	an object.
`fill`	a three component list or a vector that is coerced to a list. Shorter objects are recycled. The three components represent the fill value to the left of the data, within the interior of the data and to the right of the data, respectively. The value of any component may be the keyword `"extend"` to indicate repetition of the leftmost or rightmost non-NA value or linear interpolation in the interior. `NULL` means that items are dropped rather than filled.
`ix`	logical. Should be the same length as the number of time points. Indicates which time points not to fill. This defaults to the non-NA values.
`...`	further arguments passed to methods.

Details

na.fill is a generic function for filling NA or indicated values. It currently has methods for the time series classes "zoo" and "ts" and a default method based on the "zoo" method.

Furthermore, na.fill0 works with plain vectors and "Date" objects. It also works with "zoo" objects provided that no fill component is NULL.

See Also

na.approx

Examples

```
z <- zoo(c(NA, 2, NA, 1, 4, 5, 2, NA))
na.fill(z, "extend")
na.fill(z, c("extend", NA))
na.fill(z, -(1:3))
na.fill(z, list(NA, NULL, NA))
```

---

na.locf *Last Observation Carried Forward*

Description

Generic function for replacing each NA with the most recent non-NA prior to it.

Usage

```
na.locf(object, na.rm = TRUE, ...)
## Default S3 method:
na.locf(object, na.rm = TRUE, fromLast, rev,
        maxgap = Inf, rule = 2, ...)

na.locf0(object, fromLast = FALSE, maxgap = Inf, coredata = NULL)
```

Arguments

object	an object.
na.rm	logical. Should leading NAs be removed?
fromLast	logical. Causes observations to be carried backward rather than forward. Default is FALSE. With a value of TRUE this corresponds to NOCB (next observation carried backward). It is not supported if x or xout is specified.
rev	Use fromLast instead. This argument will be eliminated in the future in favor of fromLast.

maxgap	Runs of more than maxgap NAs are retained, other NAs are removed and the last occurrence in the resulting series prior to each time point in xout is used as that time point's output value. (If xout is not specified this reduces to retaining runs of more than maxgap NAs while filling other NAs with the last occurrence of a non-NA.)
rule	See approx.
...	further arguments passed to methods.
coredata	logical. Should LOCF be applied to the core data of a (time series) object and then assigned to the original object again? By default, this strategy is applied to time series classes (e.g., ts, zoo, xts, etc.) where it preserves the time index.

Value

An object in which each NA in the input object is replaced by the most recent non-NA prior to it. If there are no earlier non-NAs then the NA is omitted (if na.rm = TRUE) or it is not replaced (if na.rm = FALSE).

The arguments x and xout can be used in which case they have the same meaning as in approx.

Note that if a multi-column zoo object has a column entirely composed of NA then with na.rm = TRUE, the default, the above implies that the resulting object will have zero rows. Use na.rm = FALSE to preserve the NA values instead.

The function na.locf0 is the workhorse function underlying the default na.locf method. It has more limited capabilities but is faster for the special cases it covers. Implicitly, it uses na.rm=FALSE.

See Also

zoo

Examples

```
az <- zoo(1:6)

bz <- zoo(c(2,NA,1,4,5,2))
na.locf(bz)
na.locf(bz, fromLast = TRUE)

cz <- zoo(c(NA,9,3,2,3,2))
na.locf(cz)

# generate and fill in missing dates
z <- zoo(c(0.007306621, 0.007659046, 0.007681013,
 0.007817548, 0.007847579, 0.007867313),
 as.Date(c("1993-01-01", "1993-01-09", "1993-01-16",
 "1993-01-23", "1993-01-30", "1993-02-06")))
g <- seq(start(z), end(z), "day")
na.locf(z, xout = g)

# similar but use a 2 second grid

z <- zoo(1:9, as.POSIXct(c("2010-01-04 09:30:02", "2010-01-04 09:30:06",
```

```
 "2010-01-04 09:30:07", "2010-01-04 09:30:08", "2010-01-04 09:30:09",
 "2010-01-04 09:30:10", "2010-01-04 09:30:11", "2010-01-04 09:30:13",
 "2010-01-04 09:30:14")))

g <- seq(start(z), end(z), by = "2 sec")
na.locf(z, xout = g)

## get 5th of every month or most recent date prior to 5th if 5th missing.
## Result has index of the date actually used.

z <- zoo(c(1311.56, 1309.04, 1295.5, 1296.6, 1286.57, 1288.12,
1289.12, 1289.12, 1285.33, 1307.65, 1309.93, 1311.46, 1311.28,
1308.11, 1301.74, 1305.41, 1309.72, 1310.61, 1305.19, 1313.21,
1307.85, 1312.25, 1325.76), as.Date(c(13242, 13244,
13245, 13248, 13249, 13250, 13251, 13252, 13255, 13256, 13257,
13258, 13259, 13262, 13263, 13264, 13265, 13266, 13269, 13270,
13271, 13272, 13274)))

# z.na is same as z but with missing days added (with NAs)
# It is formed by merging z with a zero with series having all the dates.

rng <- range(time(z))
z.na <- merge(z, zoo(, seq(rng[1], rng[2], by = "day")))

# use na.locf to bring values forward picking off 5th of month
na.locf(z.na)[as.POSIXlt(time(z.na))$mday == 5]

## this is the same as the last one except instead of always using the
## 5th of month in the result we show the date actually used

# idx has NAs wherever z.na does but has 1, 2, 3, ... instead of
# z.na's data values (so idx can be used for indexing)

idx <- coredata(na.locf(seq_along(z.na) + (0 * z.na)))

# pick off those elements of z.na that correspond to 5th

z.na[idx[as.POSIXlt(time(z.na))$mday == 5]]

## only fill single-day gaps

merge(z.na, filled1 = na.locf(z.na, maxgap = 1))

## fill NAs in first column by inflating the most recent non-NA
## by the growth in second column.  Note that elements of x-x
## are NA if the corresponding element of x is NA and zero else

m <- zoo(cbind(c(1, 2, NA, NA, 5, NA, NA), seq(7)^2), as.Date(1:7))

r <- na.locf(m[,1]) * m[,2] / na.locf(m[,2] + (m[,1]-m[,1]))
cbind(V1 = r, V2 = m[,2])

## repeat a quarterly value every month
```

```
## preserving NAs
zq <- zoo(c(1, NA, 3, 4), as.yearqtr(2000) + 0:3/4)
tt <- as.yearmon(start(zq)) + seq(0, len = 3 * length(zq))/12
na.locf(zq, xout = tt, maxgap = 0)

## na.locf() can also be mimicked with ave()
x <- c(NA, 10, NA, NA, 20, NA)
f <- function(x) x[1]
ave(x, cumsum(!is.na(x)), FUN = f)

## by replacing f() with other functions various generalizations can be
## obtained, e.g.,
f <- function(x) if (length(x) > 3) x else x[1]  # like maxgap
f <- function(x) replace(x, 1:min(length(x)), 3) # replace up to 2 NAs
f <- function(x) if (!is.na(x[1]) && x[1] > 0) x[1] else x  # only positve numbers
```

`na.StructTS` *Fill NA or specified positions.*

Description

Generic function for filling `NA` values using seasonal Kalman filter.

Usage

```
na.StructTS(object, ...)
## S3 method for class 'ts'
na.StructTS(object, ..., na.rm = FALSE, maxgap = Inf)
## S3 method for class 'zoo'
na.StructTS(object, ..., na.rm = FALSE, maxgap = Inf)
```

Arguments

`object`	an object.
`...`	other arguments passed to methods.
`na.rm`	logical. Whether to remove end portions or fill them with NA.
`maxgap`	Runs of more than `maxgap` `NA`s are retained, other `NA`s are removed and the last occurrence in the resulting series prior to each time point in `xout` is used as that time point's output value.

Details

Interpolate with seasonal Kalman filter, using `StructTS`, followed by `tsSmooth`. The input object should be a regular time series and have a frequency. It is assumed the cycle length is 1.

See Also

`StructTS`, `tsSmooth`, `na.approx`

Examples

```
z <- zooreg(rep(10 * seq(8), each = 4) + rep(c(3, 1, 2, 4), times = 8),
 start = as.yearqtr(2000), freq = 4)
z[25] <- NA

zout <- na.StructTS(z)

plot(cbind(z, zout), screen = 1, col = 1:2, type = c("l", "p"), pch = 20)
```

---

na.trim *Trim Leading/Trailing Missing Observations*

Description

Generic function for removing leading and trailing NAs.

Usage

```
na.trim(object, ...)
## Default S3 method:
na.trim(object, sides = c("both", "left", "right"),
 is.na = c("any", "all"), maxgap = Inf, ...)
```

Arguments

object	an object.
sides	character specifying whether NAs are to be removed from both sides, just from the left side or just from the right side.
is.na	If "any" then a row will be regarded as NA if it has any NAs. If "all" then a row will be regarded as NA only if all elements in the row are NA. For one dimensional zoo objects this argument has no effect.
maxgap	maximum number of consecutive NAs to trim. Any longer gaps will be left unchanged.
...	further arguments passed to methods.

Value

An object in which leading and/or trailing NAs have been removed.

See Also

na.approx, na.contiguous, na.locf, na.omit, na.spline, stinterp, zoo

Examples

```
# examples of na.trim
x <- zoo(c(1, 4, 6), c(2, 4, 6))
xx <- zoo(matrix(c(1, 4, 6, NA, 5, 7), 3), c(2, 4, 6))
na.trim(x)
na.trim(xx)

# using na.trim for alignment
# cal defines the legal dates
# all dates within the date range of x should be present
cal <- zoo(,c(1, 2, 3, 6, 7))
x <- zoo(c(12, 16), c(2, 6))
na.trim(merge(x, cal))
```

ORDER *Ordering Permutation*

Description

ORDER is a generic function for computing ordering permutations.

Usage

```
ORDER(x, ...)
## Default S3 method:
ORDER(x, ..., na.last = TRUE, decreasing = FALSE)
```

Arguments

x	an object.
...	further arguments to be passed to methods.
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
decreasing	logical. Should the sort order be increasing or decreasing?

Details

ORDER is a new generic function which aims at providing the functionality of the non-generic base function order for arbitrary objects. Currently, there is only a default method which simply calls order. For objects (more precisely if is.object is TRUE) order leverages the generic xtfrm. Thus, to assure ordering works, one can supply either a method to xtfrm or to ORDER (or both).

See Also

order

Examples

```
ORDER(rnorm(5))
```

---

plot.zoo *Plotting zoo Objects*

Description

Plotting method for objects of class "zoo".

Usage

```
## S3 method for class 'zoo'
plot(x, y = NULL, screens, plot.type,
  panel = lines, xlab = "Index", ylab = NULL, main = NULL,
  xlim = NULL, ylim = NULL, xy.labels = FALSE, xy.lines = NULL,
  yax.flip = FALSE, oma = c(6, 0, 5, 0),
  mar = c(0, 5.1, 0, if(yax.flip) 5.1 else 2.1),
  col = 1, lty = 1, lwd = 1, pch = 1, type = "l", log = "",
  nc, widths = 1, heights = 1, ...)
## S3 method for class 'zoo'
lines(x, y = NULL, type = "l", ...)
## S3 method for class 'zoo'
points(x, y = NULL, type = "p", ...)
```

Arguments

x	an object of class "zoo".
y	an object of class "zoo". If y is NULL (the default) a time series plot of x is produced, otherwise if both x and y are univariate "zoo" series, a scatter plot of y versus x is produced.
screens	factor (or coerced to factor) whose levels specify which graph each series is to be plotted in. screens=c(1,2,1) would plot series 1, 2 and 3 in graphs 1, 2 and 1. If not specified then 1 is used if plot.type="single" and seq_len(ncol(x)) otherwise.
plot.type	for multivariate zoo objects, "multiple" plots the series on multiple plots and "single" superimposes them on a single plot. Default is "single" if screens has only one level and "multiple" otherwise. If neither screens nor plot.type is specified then "single" is used if there is one series and "mulitple" otherwise. This option is provided for back compatibility. Usually screens is used instead.
panel	a function(x, y, col, lty, ...) which gives the action to be carried out in each panel of the display for plot.type = "multiple".
ylim	if plot.type = "multiple" then it can be a list of y axis limits. If not a list each graph has the same limits. If any list element is not a pair then its range is used instead. If plot.type = "single" then it is as in plot.

xy.labels	logical, indicating if text labels should be used in the scatter plot, or character, supplying a vector of labels to be used.
xy.lines	logical, indicating if lines should be drawn in the scatter plot. Defaults to the value of xy.labels if that is logical, otherwise to FALSE.
yax.flip	logical, indicating if the y-axis (ticks and numbering) should flip from side 2 (left) to 4 (right) from series to series when type = "multiple".
xlab, ylab, main, xlim, oma, mar	graphical arguments, see par.
col, lty, lwd, pch, type	graphical arguments that can be vectors or (named) lists. See the details for more information.
log	specification of log scales as "x", "y" or "xy".
nc	the number of columns to use when plot.type = "multiple". Defaults to 1 for up to 4 series, otherwise to 2.
widths, heights	widths and heights for individual graphs, see layout.
...	additional graphical arguments.

Details

The methods for plot and lines are very similar to the corresponding ts methods. However, the handling of several graphical parameters is more flexible for multivariate series. These parameters can be vectors of the same length as the number of series plotted or are recycled if shorter. They can also be (partially) named list, e.g., list(A = c(1,2), c(3,4)) in which c(3, 4) is the default value and c(1, 2) the value only for series A. The screens argument can be specified in a similar way. If plot.type and screens conflict then multiple plots will be assumed. Also see the examples.

In the case of a custom panel the panel can reference parent.frame$panel.number in order to determine which frame the panel is being called from. See examples.

par(mfrow=...) and Axis can be used in conjunction with single panel plots in the same way as with other classic graphics.

For multi-panel graphics, plot.zoo takes over the layout so par(mfrow=...) cannot be used. Axis can be used within the panels themselves but not outside the panel. See examples. Also, par(new = TRUE) is not supported for multi-panel graphics.

In addition to classical time series line plots, there is also a simple barplot method for "zoo" series. Additionally, there is a boxplot method that visualizes the coredata of the "zoo" series with a box plot.

See Also

zoo, plot.ts, barplot, boxplot, xyplot.zoo

Examples

```
## example dates
x.Date <- as.Date(paste(2003, 02, c(1, 3, 7, 9, 14), sep = "-"))

## univariate plotting
```

```
x <- zoo(rnorm(5), x.Date)
x2 <- zoo(rnorm(5, sd = 0.2), x.Date)
plot(x)
lines(x2, col = 2)

## multivariate plotting
z <- cbind(x, x2, zoo(rnorm(5, sd = 0.5), x.Date))
plot(z, type = "b", pch = 1:3, col = 1:3, ylab = list(expression(mu), "b", "c"))
colnames(z) <- LETTERS[1:3]
plot(z, screens = 1, col = list(B = 2))
plot(z, type = "b", pch = 1:3, col = 1:3)
plot(z, type = "b", pch = list(A = 1:5, B = 3), col = list(C = 4, 2))
plot(z, type = "b", screen = c(1,2,1), col = 1:3)
# right axis is for broken lines
plot(x)
opar <- par(usr = c(par("usr")[1:2], range(x2)))
lines(x2, lty = 2)
# axis(4)
axis(side = 4)
par(opar)


## Custom x axis labelling using a custom panel.
# 1. test data
z <- zoo(c(21, 34, 33, 41, 39, 38, 37, 28, 33, 40),
     as.Date(c("1992-01-10", "1992-01-17", "1992-01-24", "1992-01-31",
       "1992-02-07", "1992-02-14", "1992-02-21", "1992-02-28", "1992-03-06",
       "1992-03-13")))
zz <- merge(a = z, b = z+10)
# 2. axis tick for every point. Also every 3rd point labelled.
my.panel <- function(x, y, ..., pf = parent.frame()) {
   fmt <- "%b-%d" # format for axis labels
   lines(x, y, ...)
   # if bottom panel
   if (with(pf, length(panel.number) == 0 ||
        panel.number %% nr == 0 || panel.number == nser)) {
      # create ticks at x values and then label every third tick
      axis(side = 1, at = x, labels = FALSE)
      ix <- seq(1, length(x), 3)
      labs <- format(x, fmt)
      axis(side = 1, at = x[ix], labels = labs[ix], tcl = -0.7, cex.axis = 0.7)
   }
}
# 3. plot
plot(zz, panel = my.panel, xaxt = "n")

# with a single panel plot a fancy x-axis is just the same
# procedure as for the ordinary plot command
plot(zz, screen = 1, col = 1:2, xaxt = "n")
# axis(1, at = time(zz), labels = FALSE)
tt <- time(zz)
axis(side = 1, at = tt, labels = FALSE)
ix <- seq(1, length(tt), 3)
```

```
fmt <- "%b-%d" # format for axis labels
labs <- format(tt, fmt)
# axis(1, at = time(zz)[ix], labels = labs[ix], tcl = -0.7, cex.axis = 0.7)
axis(side = 1, at = tt[ix], labels = labs[ix], tcl = -0.7, cex.axis = 0.7)
legend("bottomright", colnames(zz), lty = 1, col = 1:2)

## plot a mulitple ts series with nice x-axis using panel function
tab <- ts(cbind(A = 1:24, B = 24:1), start = c(2006, 1), freq = 12)
pnl.xaxis <- function(...) {
     lines(...)
     panel.number <- parent.frame()$panel.number
     nser <- parent.frame()$nser
     # if bottom panel
     if (!length(panel.number) || panel.number == nser) {
           tt <- list(...)[[1]]
           ym <- as.yearmon(tt)
    mon <- as.numeric(format(ym, "%m"))
    yy <- format(ym, "%y")
    mm <- substring(month.abb[mon], 1, 1)
       if (any(mon == 1))
     # axis(1, tt[mon == 1], yy[mon == 1], cex.axis = 0.7)
     axis(side = 1, at = tt[mon == 1], labels = yy[mon == 1], cex.axis = 0.7)
    # axis(1, tt[mon > 1], mm[mon > 1], cex.axis = 0.5, tcl = -0.3)
    axis(side = 1, at = tt[mon > 1], labels = mm[mon > 1], cex.axis = 0.5, tcl = -0.3)
     }
}
plot(as.zoo(tab), panel = pnl.xaxis, xaxt = "n", main = "Fancy X Axis")

## Another example with a custom axis
# test data
z <- zoo(matrix(1:25, 5), c(10,11,20,21))
colnames(z) <- letters[1:5]

plot(zoo(coredata(z)), xaxt = "n", panel = function(x, y, ..., Time = time(z)) {
    lines(x, y, ...)
    # if bottom panel
    pf <- parent.frame()
    if (with(pf, panel.number %% nr == 0 || panel.number == nser)) {
        axis(side = 1, at = x, labels = Time)
    }
})


## plot with left and right axes
## modified from http://www.mayin.org/ajayshah/KB/R/html/g6.html
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
z <- zoo(cbind(A = cumsum(rnorm(100)), B = cumsum(rnorm(100, mean = 0.2))))
opar <- par(mai = c(.8, .8, .2, .8))
plot(z[,1], type = "l",
  xlab = "x-axis label", ylab = colnames(z)[1])
par(new = TRUE)
plot(z[,2], type = "l", ann = FALSE, yaxt = "n", col = "blue")
```

```
# axis(4)
axis(side = 4)
legend(x = "topleft", bty = "n", lty = c(1,1), col = c("black", "blue"),
  legend = paste(colnames(z), c("(left scale)", "(right scale)")))
usr <- par("usr")
# if you don't care about srt= in text then mtext is shorter:
#   mtext(colnames(z)[2], 4, 2, col = "blue")
text(usr[2] + .1 * diff(usr[1:2]), mean(usr[3:4]), colnames(z)[2],
  srt = -90, xpd = TRUE, col = "blue")
par(opar)


## another plot with left and right axes
## modified from https://stat.ethz.ch/pipermail/r-help/2014-May/375293.html
d1 <- c(38.2, 18.1, 83.2, 42.7, 22.8, 48.1, 81.8, 129.6, 52.0, 110.3)
d2 <- c(2.2, 0.8, 0.7, 1.6, 0.9, 0.9, 1.1, 2.8, 5.1, 2.1)
z1 <- zooreg(d1, start = as.POSIXct("2013-01-01 00:00:01"), frequency = 0.0000006)
z2 <- zooreg(d2, start = as.POSIXct("2013-01-01 00:00:20"), frequency = 0.0000006)
zt <- zooreg(rnorm(1050), start = as.POSIXct("2013-01-01 00:00:01"), frequency = 0.00007)
z <- merge(zt, z1, z2, all = TRUE)
z <- na.spline(z[,2:3], na.rm = FALSE)
## function to round up to a number divisible by n (2011 by Owen Jones)
roundup <- function(x, n) ceiling(ceiling(x)/n) * n
## plot how to match secondary y-axis ticks to primary ones
plot(z$z1, ylim = c(0, signif(max(na.omit(z$z1)), 2)), xlab = "")
## use multiplication for even tick numbers and fake sekondary y-axis
max.yl <- roundup(max(na.omit(z$z2)), par("yaxp")[3])
multipl.yl <- max(na.omit(z$z2)) / max.yl
multipl.z2 <- signif(max(na.omit(z$z1) * 1.05), 2)/max.yl
lines(z$z2 * multipl.z2, lty = 2)
at4 <- axTicks(4)
axis(4, at = at4, seq(0, max.yl, length.out = par("yaxp")[3] + 1))


# automatically placed point labels
## Not run:
library("maptools")
pointLabel(time(z), coredata(z[,2]), labels = format(time(z)), cex = 0.5)

## End(Not run)

## plot one zoo series against the other.
plot(x, x2)
plot(x, x2, xy.labels = TRUE)
plot(x, x2, xy.labels = 1:5, xy.lines = FALSE)

## shade a portion of a plot and make axis fancier

v <- zooreg(rnorm(50), start = as.yearmon(2004), freq = 12)

plot(v, type = "n")
u <- par("usr")
rect(as.yearmon("2007-8"), u[3], as.yearmon("2009-11"), u[4],
```

```
   border = 0, col = "grey")
lines(v)
axis(1, floor(time(v)), labels = FALSE, tcl = -1)

## shade certain times to show recessions, etc.
v <- zooreg(rnorm(50), start = as.yearmon(2004), freq = 12)
plot(v, type = "n")
u <- par("usr")
rect(as.yearmon("2007-8"), u[3], as.yearmon("2009-11"), u[4],
   border = 0, col = "grey")
lines(v)
axis(1, floor(time(v)), labels = FALSE, tcl = -1)

## fill area under plot

pnl.xyarea <- function(x, y, fill.base = 0, col = 1, ...) {
       lines(x, y, ...)
       panel.number <- parent.frame()$panel.number
    col <- rep(col, length = panel.number)[panel.number]
       polygon(c(x[1], x, tail(x, 1), x[1]),
  c(fill.base, as.numeric(y), fill.base, fill.base), col = col)
}
plot(zoo(EuStockMarkets), col = rainbow(4), panel = pnl.xyarea)


## barplot
x <- zoo(cbind(rpois(5, 2), rpois(5, 3)), x.Date)
barplot(x, beside = TRUE)

## boxplot
boxplot(x)

## 3d plot
## The persp function in R (not part of zoo) works with zoo objects.
## The following example is by Enrico Schumann.
## https://stat.ethz.ch/pipermail/r-sig-finance/2009q1/003710.html
nC <- 10    # columns
nO <- 100 # observations
dataM <- array(runif(nC * nO), dim=c(nO, nC))
zz <- zoo(dataM, 1:nO)
persp(1:nO, 1:nC, zz)

# interactive plotting
## Not run:
library("TeachingDemos")
tke.test1 <- list(Parameters = list(
 lwd = list("spinbox", init = 1, from = 0, to = 5, increment = 1, width = 5),
 lty = list("spinbox", init = 1, from = 0, to = 6, increment = 1, width = 5)
))
z <- zoo(rnorm(25))
tkexamp(plot(z), tke.test1, plotloc = "top")

## End(Not run)
```

```
# setting ylim on a multi-panel plot - 2nd panel y axis range is 1-50
data("anscombe", package = "datasets")
ans6 <- zoo(anscombe[, 1:6])
screens <- c(1, 1, 2, 2, 3, 3)
ylim <- unname(tapply(as.list(ans6), screens, range))
ylim[[2]] <- 1:50 # or ylim[[2]] <- c(1, 50)
plot(ans6, screens = screens, ylim = ylim)
```

---

read.zoo *Reading and Writing zoo Series*

Description

read.zoo and write.zoo are convenience functions for reading and writing "zoo" series from/to text files. They are convenience interfaces to read.table and write.table, respectively. To employ read.csv, read.csv2, read.delim, read.delim2 instead of read.table additional functions read.csv.zoo etc. are provided.

Usage

```
read.zoo(file, format = "", tz = "", FUN = NULL,
  regular = FALSE, index.column = 1, drop = TRUE, FUN2 = NULL,
  split = NULL, aggregate = FALSE, ..., text, read = read.table)

write.zoo(x, file = "", index.name = "Index", row.names = FALSE, col.names = NULL,
  FUN = format, ...)

read.csv.zoo(..., read = read.csv)
read.csv2.zoo(..., read = read.csv2)
read.delim.zoo(..., read = read.delim)
read.delim2.zoo(..., read = read.delim2)
```

Arguments

file	character string or strings giving the name of the file(s) which the data are to be read from/written to. See read.table and write.table for more information. Alternatively, in read.zoo, file can be a connection or a data.frame (e.g., resulting from a previous read.table call) that is subsequently processed to a "zoo" series.
format	date format argument passed to FUN.
tz	time zone argument passed to as.POSIXct.
FUN	a function for processing the time index. In read.zoo this is the function that computes the index from the index.column, see the details. In write.zoo it is a function that formats the index prior to writing it to file.
regular	logical. Should the series be coerced to class "zooreg" (if the series is regular)?

index.column — numeric vector or list. The column names or numbers of the data frame in which the index/time is stored. If the read.table argument colClasses is used and "NULL" is among its components then index.column refers to the column numbers after the columns corresponding to "NULL" in colClasses have been removed. If specified as a list then one argument will be passed to argument FUN per component so that, for example, index.column = list(1, 2) will cause FUN(x[,1], x[,2], ...) to be called whereas index.column = list(1:2) will cause FUN(x[,1:2], ...) to be called where x is a data frame of characters data. Here ... refers to format and/or tz, if they specified as arguments. index.column = 0 can be used to specify that the row names be used as the index. In the case that no row names were input sequential numbering is used. If index.column is specified as an ordinary vector then if it has the same length as the number of arguments of FUN (or FUN2 in the event that FUN2 is specified and FUN is not) then index.column is converted to a list. Also it is always converted to a list if it has length 1.

drop — logical. If the data frame contains just a single data column, should the second dimension be dropped?

x — a "zoo" object.

index.name — character with name of the index column in the written data file.

row.names — logical. Should row names be written? Default is FALSE because the row names are just character representations of the index.

col.names — logical. Should column names be written? Default is to write column names only if x has column names.

FUN2 — function. It is applied to the time index after FUN and before aggregate. If FUN is not specified but FUN2 is specified then only FUN2 is applied.

split — NULL or column number or name or vector of numbers or names. If not NULL then the data is assumed to be in long format and is split according to the indicated columns. See the R reshape command for description of long data. If split = Inf then the first of each run among the times are made into a separate series, the second of each run and so on. If split= -Inf then the last of each run is made into a separate series, the second last and so on.

aggregate — logical or function. If set to TRUE, then aggregate.zoo is applied to the zoo object created to compute the mean of all values with the same time index. Alternatively, aggregate can be set to any other function that should be used for aggregation. If FALSE (the default), no aggregation is performed and a warning is given if there are any duplicated time indexes. Note that most zoo functions do not accept objects with duplicate time indexes. See aggregate.zoo.

... — further arguments passed to other functions. In the read.*.zoo the arguments are passed to the function specified in read (unless file is a data.frame already). In write.zoo the arguments are passed to write.table.

text — character. If file is not supplied and this is, then data are read from the value of text via a text connection. See below for an example.

read — function. The function for reading file (unless it is a data.frame already).

Details

read.zoo is a convenience function which should make it easier to read data from a text file and turn it into a "zoo" series immediately. read.zoo reads the data file via read.table(file, ...). The column index.column (by default the first) of the resulting data is interpreted to be the index/time, the remaining columns the corresponding data. (If the file only has only column then that is assumed to be the data column and 1, 2, ... are used for the index.) To assign the appropriate class to the index, FUN can be specified and is applied to the first column.

To process the index, read.zoo calls FUN with the index as the first argument. If FUN is not specified, the following default is employed:

(a) If file is a data frame with a single index column that appears to be a time index already, then FUN = identity is used. The conditions for a readily produced time index are: It is not character or factor (and the arguments tz and format must not be specified).

(b) If the conditions from (a) do not hold then the following strategy is used. If there are multiple index columns they are pasted together with a space between each. Using the (pasted) index column: (1) If tz is specified then the index column is converted to POSIXct. (2) If format is specified then the index column is converted to Date. (3) Otherwise, a heuristic attempts to decide between "numeric", "POSIXct", and "Date" by trying them in that order (which may not always succeed though). By default, only the standard date/time format is used. Hence, supplying format and/or tz is necessary if some date/time format is used that is not the default. And even if the default format is appropriate for the index, explicitly supplying FUN or at least format and/or tz typically leads to more reliable results than the heuristic.

If regular is set to TRUE and the resulting series has an underlying regularity, it is coerced to a "zooreg" series.

To employ other functions than read.table to read the initial data, further convenience interfaces read.csv.zoo etc. are provided.

write.zoo is a convenience function for writing "zoo" series to text files. It first coerces its argument to a "data.frame", adds a column with the index and then calls write.table.

See also vignette("zoo-read", package = "zoo") for detailed examples.

Value

read.zoo returns an object of class "zoo" (or "zooreg").

Note

read.zoo works by first reading the data in using read.table and then processing it. This implies that if the index field is entirely numeric the default is to pass it to FUN or the built-in date conversion routine a number, rather than a character string. Thus, a date field such as 09122007 intended to represent September 12, 2007 would be seen as 9122007 and interpreted as the 91st day thereby generating an error.

This comment also applies to trailing decimals so that if 2000.10 were intended to represent the 10th month of 2000 in fact it would receive 2000.1 and regard it as the first month of 2000 unless similar precautions were taken.

In the above cases the index field should be specified to be "character" so that leading or trailing zeros are not dropped. This can be done by specifying a "character" index column in the "colClasses" argument, which is passed to read.table, as shown in the examples below.

See Also

zoo

Examples

```
## this manual page provides a few typical examples, many more cases
## are covered in vignette("zoo-read", package = "zoo")

## read text lines with a single date column
Lines <- "2013-12-24  2
2013-12-25 3
2013-12-26 8"
read.zoo(text = Lines, FUN = as.Date)       # explicit coercion
read.zoo(text = Lines, format = "%Y-%m-%d") # same
read.zoo(text = Lines)                      # same, via heuristic

## read text lines with date/time in separate columns
Lines <- "2013-11-24 12:41:21 2
2013-12-25 12:41:22.25 3
2013-12-26 12:41:22.75 8"
read.zoo(text = Lines, index = 1:2,
  FUN = paste, FUN2 = as.POSIXct)            # explicit coercion
read.zoo(text = Lines, index = 1:2, tz = "") # same
read.zoo(text = Lines, index = 1:2)          # same, via heuristic

## read text lines with month/year in separate columns
Lines <- "Jan 1998 4.36
Feb 1998 4.34"
read.zoo(text = Lines, index = 1:2, FUN = paste, FUN2 = as.yearmon)

## read directly from a data.frame (artificial and built-in BOD)
dat <- data.frame(date = paste("2000-01-", 10:15, sep = ""),
  a = sin(1:6), b = cos(1:6))
read.zoo(dat)
data("BOD", package = "datasets")
read.zoo(BOD)

## Not run:
## descriptions of typical examples

## turn *numeric* first column into yearmon index
## where number is year + fraction of year represented by month
z <- read.zoo("foo.csv", sep = ",", FUN = as.yearmon)

## first column is of form yyyy.mm
## (Here we use format in place of as.character so that final zero
## is not dropped in dates like 2001.10 which as.character would do.)
f <- function(x) as.yearmon(format(x, nsmall = 2), "%Y.%m")
z <- read.zoo("foo.csv", header = TRUE, FUN = f)

## turn *character* first column into "Date" index
## Assume lines look like: 12/22/2007 1 2
```

```
z <- read.zoo("foo.tab", format = "%m/%d/%Y")

# Suppose lines look like: 09112007 1 2 and there is no header
z <- read.zoo("foo.txt", format = "%d%m%Y")

## csv file with first column of form YYYY-mm-dd HH:MM:SS
## Read in times as "chron" class. Requires chron 2.3-22 or later.
z <- read.zoo("foo.csv", header = TRUE, sep = ",", FUN = as.chron)

## same but with custom format.  Note as.chron uses POSIXt-style
## Read in times as "chron" class. Requires chron 2.3-24 or later.
z <- read.zoo("foo.csv", header = TRUE, sep = ",", FUN = as.chron,
 format = "

## same file format but read it in times as "POSIXct" class.
z <- read.zoo("foo.csv", header = TRUE, sep = ",", tz = "")

## csv file with first column mm-dd-yyyy. Read times as "Date" class.
z <- read.zoo("foo.csv", header = TRUE, sep = ",", format = "%m-%d-%Y")

## whitespace separated file with first column of form YYYY-mm-ddTHH:MM:SS
## and no headers.  T appears literally.  Requires chron 2.3-22 or later.
z <- read.zoo("foo.csv", FUN = as.chron)

# read in all csv files in the current directory and merge them
read.zoo(Sys.glob("*.csv"), header = TRUE, sep = ",")

# We use "NULL" in colClasses for those columns we don't need but in
# col.names we still have to include dummy names for them.  Of what
# is left the index is the first three columns (1:3) which we convert
# to chron class times in FUN and then truncate to 5 seconds in FUN2.
# Finally we use aggregate = mean to average over the 5 second intervals.
library("chron")

Lines <- "CVX 20070201 9 30 51 73.25 81400 0
CVX 20070201 9 30 51 73.25 100 0
CVX 20070201 9 30 51 73.25 100 0
CVX 20070201 9 30 51 73.25 300 0
CVX 20070201 9 30 51 73.25 81400 0
CVX 20070201 9 40 51 73.25 100 0
CVX 20070201 9 40 52 73.25 100 0
CVX 20070201 9 40 53 73.25 300 0"

z <- read.zoo(text = Lines,
  colClasses = c("NULL", "NULL", "numeric", "numeric", "numeric",
    "numeric", "numeric", "NULL"),
  col.names = c("Symbol", "Date", "Hour", "Minute", "Second", "Price", "Volume", "junk"),
  index = 1:3,  # do not count columns that are "NULL" in colClasses
  FUN = function(h, m, s) times(paste(h, m, s, sep = ":")),
  FUN2 = function(tt) trunc(tt, "00:00:05"),
  aggregate = mean)

## End(Not run)
```

`rollapply` *Apply Rolling Functions*

Description

A generic function for applying a function to rolling margins of an array.

Usage

```
rollapply(data, ...)
## S3 method for class 'ts'
rollapply(data, ...)
## S3 method for class 'zoo'
rollapply(data, width, FUN, ..., by = 1, by.column = TRUE,
    fill = if (na.pad) NA, na.pad = FALSE, partial = FALSE,
    align = c("center", "left", "right"), coredata = TRUE)
## Default S3 method:
rollapply(data, ...)
rollapplyr(..., align = "right")
```

Arguments

`data`	the data to be used (representing a series of observations).
`width`	numeric vector or list. In the simplest case this is an integer specifying the window width (in numbers of observations) which is aligned to the original sample according to the `align` argument. Alternatively, `width` can be a list regarded as offsets compared to the current time, see below for details.
`FUN`	the function to be applied.
`...`	optional arguments to `FUN`.
`by`	calculate FUN at every by-th time point rather than every point. by is only used if `width` is length 1 and either a plain scalar or a list.
`by.column`	logical. If `TRUE`, `FUN` is applied to each column separately.
`fill`	a three-component vector or list (recycled otherwise) providing filling values at the left/within/to the right of the data range. See the `fill` argument of `na.fill` for details.
`na.pad`	deprecated. Use `fill = NA` instead of `na.pad = TRUE`.
`partial`	logical or numeric. If `FALSE` (default) then `FUN` is only applied when all indexes of the rolling window are within the observed time range. If `TRUE`, then the subset of indexes that are in range are passed to `FUN`. A numeric argument to `partial` can be used to determin the minimal window size for partial computations. See below for more details.

align	specifyies whether the index of the result should be left- or right-aligned or centered (default) compared to the rolling window of observations. This argument is only used if width represents widths.
coredata	logical. Should only the coredata(data) be passed to every width window? If set to FALSE the full zoo series is used.

Details

If width is a plain numeric vector its elements are regarded as widths to be interpreted in conjunction with align whereas if width is a list its components are regarded as offsets. In the above cases if the length of width is 1 then width is recycled for every by-th point. If width is a list its components represent integer offsets such that the i-th component of the list refers to time points at positions i + width[[i]]. If any of these points are below 1 or above the length of index(data) then FUN is not evaluated for that point unless partial = TRUE and in that case only the valid points are passed.

The rolling function can also be applied to partial windows by setting partial = TRUE For example, if width = 3, align = "right" then for the first point just that point is passed to FUN since the two points to its left are out of range. For the same example, if partial = FALSE then FUN is not invoked at all for the first two points. If partial is a numeric then it specifies the minimum number of offsets that must be within range. Negative partial is interpreted as FALSE.

If width is a scalar then partial = TRUE and fill = NA are mutually exclusive but if offsets are specified for the width and 0 is not among the offsets then the output will be shorter than the input even if partial = TRUE is specified. In that case it may still be useful to specify fill in addition to partial.

If FUN is mean, max or median and by.column is TRUE and width is a plain scalar and there are no other arguments then special purpose code is used to enhance performance. Also in the case of mean such special purpose code is only invoked if the data argument has no NA values. See rollmean, rollmax and rollmedian for more details.

Currently, there are methods for "zoo" and "ts" series and "default" method for ordinary vectors and matrices.

rollapplyr is a wrapper around rollapply that uses a default of align = "right".

If data is of length 0, data is returned unmodified.

Value

A object of the same class as data with the results of the rolling function.

See Also

rollmean

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## rolling mean
z <- zoo(11:15, as.Date(31:35))
rollapply(z, 2, mean)
```

```
## non-overlapping means
z2 <- zoo(rnorm(6))
rollapply(z2, 3, mean, by = 3)      # means of nonoverlapping groups of 3
aggregate(z2, c(3,3,3,6,6,6), mean) # same

## optimized vs. customized versions
rollapply(z2, 3, mean)   # uses rollmean which is optimized for mean
rollmean(z2, 3)          # same
rollapply(z2, 3, (mean)) # does not use rollmean


## rolling regression:
## set up multivariate zoo series with
## number of UK driver deaths and lags 1 and 12
seat <- as.zoo(log(UKDriverDeaths))
time(seat) <- as.yearmon(time(seat))
seat <- merge(y = seat, y1 = lag(seat, k = -1),
  y12 = lag(seat, k = -12), all = FALSE)

## run a rolling regression with a 3-year time window
## (similar to a SARIMA(1,0,0)(1,0,0)_12 fitted by OLS)
rr <- rollapply(seat, width = 36,
  FUN = function(z) coef(lm(y ~ y1 + y12, data = as.data.frame(z))),
  by.column = FALSE, align = "right")

## plot the changes in coefficients
## showing the shifts after the oil crisis in Oct 1973
## and after the seatbelt legislation change in Jan 1983
plot(rr)


## rolling mean by time window (e.g., 3 days) rather than
## by number of observations (e.g., when these are unequally spaced):
#
## - test data
tt <- as.Date("2000-01-01") + c(1, 2, 5, 6, 7, 8, 10)
z <- zoo(seq_along(tt), tt)
## - fill it out to a daily series, zm, using NAs
## using a zero width zoo series g on a grid
g <- zoo(, seq(start(z), end(z), "day"))
zm <- merge(z, g)
## - 3-day rolling mean
rollapply(zm, 3, mean, na.rm = TRUE, fill = NA)
##
## - without expansion to regular grid: find interval widths
## that encompass the previous 3 days for each Date
w <- seq_along(tt) - findInterval(tt - 3, tt)
## a solution to computing the widths 'w' that is easier to read but slower
## w <- sapply(tt, function(x) sum(tt >= x - 2 & tt <= x))
##
## - rolling sum from 3-day windows
## without vs. with expansion to regular grid
```

```
rollapplyr(z, w, sum)
rollapplyr(zm, 3, sum, partial = TRUE, na.rm = TRUE)


## rolling weekly sums (with some missing dates)
z <- zoo(1:11, as.Date("2016-03-09") + c(0:7, 9:10, 12))
weeksum <- function(z) sum(z[time(z) > max(time(z)) - 7])
zs <- rollapplyr(z, 7, weeksum, fill = NA, coredata = FALSE)
merge(value = z, weeksum = zs)


## replicate cumsum with either 'partial' or vector width 'k'
cumsum(1:10)
rollapplyr(1:10, 10, sum, partial = TRUE)
rollapplyr(1:10, 1:10, sum)


## different values of rule argument
z <- zoo(c(NA, NA, 2, 3, 4, 5, NA))
rollapply(z, 3, sum, na.rm = TRUE)
rollapply(z, 3, sum, na.rm = TRUE, fill = NULL)
rollapply(z, 3, sum, na.rm = TRUE, fill = NA)
rollapply(z, 3, sum, na.rm = TRUE, partial = TRUE)

# this will exclude time points 1 and 2
# It corresponds to align = "right", width = 3
rollapply(zoo(1:8), list(seq(-2, 0)), sum)

# but this will include points 1 and 2
rollapply(zoo(1:8), list(seq(-2, 0)), sum, partial = 1)
rollapply(zoo(1:8), list(seq(-2, 0)), sum, partial = 0)

# so will this
rollapply(zoo(1:8), list(seq(-2, 0)), sum, fill = NA)

# by = 3, align = "right"
L <- rep(list(NULL), 8)
L[seq(3, 8, 3)] <- list(seq(-2, 0))
str(L)
rollapply(zoo(1:8), L, sum)

rollapply(zoo(1:8), list(0:2), sum, fill = 1:3)
rollapply(zoo(1:8), list(0:2), sum, fill = 3)

L2 <- rep(list(-(2:0)), 10)
L2[5] <- list(NULL)
str(L2)
rollapply(zoo(1:10), L2, sum, fill = "extend")
rollapply(zoo(1:10), L2, sum, fill = list("extend", NULL))

rollapply(zoo(1:10), L2, sum, fill = list("extend", NA))

rollapply(zoo(1:10), L2, sum, fill = NA)
```

```
rollapply(zoo(1:10), L2, sum, fill = 1:3)
rollapply(zoo(1:10), L2, sum, partial = TRUE)
rollapply(zoo(1:10), L2, sum, partial = TRUE, fill = 99)

rollapply(zoo(1:10), list(-1), sum, partial = 0)
rollapply(zoo(1:10), list(-1), sum, partial = TRUE)

rollapply(zoo(cbind(a = 1:6, b = 11:16)), 3, rowSums, by.column = FALSE)

# these two are the same
rollapply(zoo(cbind(a = 1:6, b = 11:16)), 3, sum)
rollapply(zoo(cbind(a = 1:6, b = 11:16)), 3, colSums, by.column = FALSE)

# these two are the same
rollapply(zoo(1:6), 2, sum, by = 2, align = "right")
aggregate(zoo(1:6), c(2, 2, 4, 4, 6, 6), sum)

# these two are the same
rollapply(zoo(1:3), list(-1), c)
lag(zoo(1:3), -1)

# these two are the same
rollapply(zoo(1:3), list(1), c)
lag(zoo(1:3))

# these two are the same
rollapply(zoo(1:5), list(c(-1, 0, 1)), sum)
rollapply(zoo(1:5), 3, sum)

# these two are the same
rollapply(zoo(1:5), list(0:2), sum)
rollapply(zoo(1:5), 3, sum, align = "left")

# these two are the same
rollapply(zoo(1:5), list(-(2:0)), sum)
rollapply(zoo(1:5), 3, sum, align = "right")

# these two are the same
rollapply(zoo(1:6), list(NULL, NULL, -(2:0)), sum)
rollapply(zoo(1:6), 3, sum, by = 3, align = "right")

# these two are the same
rollapply(zoo(1:5), list(c(-1, 1)), sum)
rollapply(zoo(1:5), 3, function(x) sum(x[-2]))

# these two are the same
rollapply(1:5, 3, rev)
embed(1:5, 3)

# these four are the same
x <- 1:6
rollapply(c(0, 0, x), 3, sum, align = "right") - x
rollapply(x, 3, sum, partial = TRUE, align = "right") - x
```

```
rollapply(x, 3, function(x) sum(x[-3]), partial = TRUE, align = "right")
rollapply(x, list(-(2:1)), sum, partial = 0)

# same as Matlab's buffer(x, n, p) for valid non-negative p
# See http://www.mathworks.com/help/toolbox/signal/buffer.html
x <- 1:30; n <- 7; p <- 3
t(rollapply(c(rep(0, p), x, rep(0, n-p)), n, by = n-p, c))

# these three are the same
y <- 10 * seq(8); k <- 4; d <- 2
# 1
# from http://ucfagls.wordpress.com/2011/06/14/embedding-a-time-series-with-time-delay-in-r-part-ii/
Embed <- function(x, m, d = 1, indices = FALSE, as.embed = TRUE) {
    n <- length(x) - (m-1)*d
    X <- seq_along(x)
    if(n <= 0)
        stop("Insufficient observations for the requested embedding")
    out <- matrix(rep(X[seq_len(n)], m), ncol = m)
    out[,-1] <- out[,-1, drop = FALSE] +
        rep(seq_len(m - 1) * d, each = nrow(out))
    if(as.embed)
        out <- out[, rev(seq_len(ncol(out)))]
    if(!indices)
        out <- matrix(x[out], ncol = m)
    out
}
Embed(y, k, d)
# 2
rollapply(y, list(-d * seq(0, k-1)), c)
# 3
rollapply(y, d*k-1, function(x) x[d * seq(k-1, 0) + 1])


## mimic convolve() using rollapplyr()
A <- 1:4
B <- 5:8
## convolve(..., type = "open")
cross <- function(x) x
rollapplyr(c(A, 0*B[-1]), length(B), cross, partial = TRUE)
convolve(A, B, type = "open")

# convolve(..., type = "filter")
rollapplyr(A, length(B), cross)
convolve(A, B, type = "filter")


# weighted sum including partials near ends, keeping
## alignment with wts correct
points <- zoo(cbind(lon = c(11.8300715, 11.8296697,
    11.8268708, 11.8267236, 11.8249612, 11.8251062),
  lat = c(48.1099048, 48.10884, 48.1067431, 48.1066077,
    48.1037673, 48.103318),
  dist = c(46.8463805878941, 33.4921440879536, 10.6101735030534,
```

```
    18.6085009578724, 6.97253109610173, 9.8912817449265)))
mysmooth <- function(z, wts = c(0.3, 0.4, 0.3)) {
  notna <- !is.na(z)
  sum(z[notna] * wts[notna]) / sum(wts[notna])
}
points2 <- points
points2[, 1:2] <- rollapply(rbind(NA, coredata(points)[, 1:2], NA), 3, mysmooth)
points2
```

`rollmean` *Rolling Means/Maximums/Medians/Sums*

Description

Generic functions for computing rolling means, maximums, medians, and sums of ordered observations.

Usage

```
rollmean(x, k, fill = if (na.pad) NA, na.pad = FALSE,
  align = c("center", "left", "right"), ...)

rollmax(x, k, fill = if (na.pad) NA, na.pad = FALSE,
  align = c("center", "left", "right"), ...)

rollmedian(x, k, fill = if (na.pad) NA, na.pad = FALSE,
  align = c("center", "left", "right"), ...)

rollsum(x, k, fill = if (na.pad) NA, na.pad = FALSE,
  align = c("center", "left", "right"), ...)

rollmeanr(..., align = "right")
rollmaxr(..., align = "right")
rollmedianr(..., align = "right")
rollsumr(..., align = "right")
```

Arguments

`x`	an object (representing a series of observations).
`k`	integer width of the rolling window. Must be odd for `rollmedian`.
`fill`	a three-component vector or list (recycled otherwise) providing filling values at the left/within/to the right of the data range. See the `fill` argument of `na.fill` for details.
`na.pad`	deprecated. Use `fill = NA` instead of `na.pad = TRUE`.
`align`	character specifying whether the index of the result should be left- or right-aligned or centered (default) compared to the rolling window of observations.
`...`	Further arguments passed to methods.

Details

These functions compute rolling means, maximums, medians, and sums respectively and are thus similar to rollapply but are optimized for speed.

Currently, there are methods for "zoo" and "ts" series and default methods. The default method of rollmedian is an interface to runmed. The default methods of rollmean and rollsum do not handle inputs that contain NAs. In such cases, use rollapply instead.

If x is of length 0, x is returned unmodified.

Value

An object of the same class as x with the rolling mean/max/median/sum.

See Also

rollapply, zoo, na.fill

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

x.Date <- as.Date(paste(2004, rep(1:4, 4:1), sample(1:28, 10), sep = "-"))
x <- zoo(rnorm(12), x.Date)

## rolling operations for univariate series
rollmean(x, 3)
rollmax(x, 3)
rollmedian(x, 3)
rollsum(x, 3)

## rolling operations for multivariate series
xm <- zoo(matrix(1:12, 4, 3), x.Date[1:4])
rollmean(xm, 3)
rollmax(xm, 3)
rollmedian(xm, 3)
rollsum(xm, 3)

## rollapply vs. dedicated rollmean
rollapply(xm, 3, mean) # uses rollmean
rollapply(xm, 3, function(x) mean(x)) # does not use rollmean
```

---

window.zoo *Extract/Replacing the Time Windows of Objects*

Description

Methods for extracting time windows of "zoo" objects and replacing it.

Usage

```
## S3 method for class 'zoo'
window(x, index. = index(x), start = NULL, end = NULL, ...)
## S3 replacement method for class 'zoo'
window(x, index. = index(x), start = NULL, end = NULL, ...) <- value
```

Arguments

x	an object.
index.	the index/time window which should be extracted.
start	an index/time value. Only the indexes in index which are greater or equal to start are used. If the index class supports comparisons to character variables, as does "Date" class, "yearmon" class, "yearqtr" class and the chron package classes "dates" and "times" then start may alternately be a character variable.
end	an index/time value. Only the indexes in index which are lower or equal to end are used. Similar comments about character variables mentioned under start apply here too.
value	a suitable value object for use with window(x).
...	currently not used.

Value

Either the time window of the object is extracted (and hence return a "zoo" object) or it is replaced.

See Also

zoo

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## zoo example
x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,19,2), sep = "-"))
x <- zoo(matrix(rnorm(20), ncol = 2), x.date)
x

window(x, start = as.Date("2003-02-01"), end = as.Date("2003-03-01"))
window(x, index = x.date[1:6], start = as.Date("2003-02-01"))
window(x, index = x.date[c(4, 8, 10)])
window(x, index = x.date[c(4, 8, 10)]) <- matrix(1:6, ncol = 2)
x

## for classes that support comparisons with "character" variables
## start and end may be "character".
window(x, start = "2003-02-01")
```

```
## zooreg example (with plain numeric index)
z <- zooreg(rnorm(10), start = 2000, freq = 4)
window(z, start = 2001.75)
window(z, start = c(2001, 4))

## replace data at times of d0 which are in dn
d1 <- d0 <- zoo(1:10) + 100
dn <- - head(d0, 4)
window(d1, time(dn)) <- coredata(dn)

## if the underlying time index is a float, note that the index may
## print in the same way but actually be different (e.g., differing
## by 0.1 second in this example)
zp <- zoo(1:4, as.POSIXct("2000-01-01 00:00:00") + c(-3600, 0, 0.1, 3600))
## and then the >= start and <= end may not select all intended
## observations and adding/subtracting some "fuzz" may be needed
window(zp, end = "2000-01-01 00:00:00")
window(zp, end = as.POSIXct("2000-01-01 00:00:00") + 0.5)
```

`xblocks` *Plot contiguous blocks along x axis.*

Description

Plot contiguous blocks along x axis. A typical use would be to highlight events or periods of missing data.

Usage

```
xblocks(x, ...)

## Default S3 method:
xblocks(x, y, ..., col = NULL, border = NA,
        ybottom = par("usr")[3], ytop = ybottom + height,
        height = diff(par("usr")[3:4]),
        last.step = median(diff(tail(x))))

## S3 method for class 'zoo'
xblocks(x, y = x, ...)

## S3 method for class 'ts'
xblocks(x, y = x, ...)
```

Arguments

`x, y`	In the default method, `x` gives the ordinates along the `x` axis and must be in increasing order. `y` gives the color values to plot as contiguous blocks. If `y` is numeric, data coverage is plotted, by converting it into a logical (`!is.na(y)`). Finally, if `y` is a function, it is applied to `x` (`time(x)` in the time series methods).

If y has character (or factor) values, these are interpreted as colors – and should therefore be color names or hex codes. Missing values in y are not plotted. The default color is taken from `palette()[1]`. If `col` is given, this over-rides the block colors given as y.

The `ts` and `zoo` methods plot the `coredata(y)` values against the time index `index(x)`.

`...` In the default method, further arguments are graphical parameters passed on to `gpar`.

`col` if `col` is specified, it determines the colors of the blocks defined by y. If multiple colors are specified they will be repeated to cover the total number of blocks.

`border` border color.

`ybottom, ytop, height`

y axis position of the blocks. The default it to fill the whole plot region, but by setting these values one can draw blocks along the top of bottom of the plot. Note that `height` is not used directly, it only sets the default value of `ytop`.

`last.step` width (in native units) of the final block. Defaults to the median of the last 5 time steps (assuming steps are regular).

Details

Blocks are drawn forward in "time" from the specified x locations, up until the following value. Contiguous blocks are calculated using `rle`.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`rect`

Examples

```
## example time series:
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
flow <- ts(filter(rlnorm(200, mean = 1), 0.8, method = "r"))

## highlight values above and below thresholds.
## this draws on top using semi-transparent colors.
rgb <- hcl(c(0, 0, 260), c = c(100, 0, 100), l = c(50, 90, 50), alpha = 0.3)
plot(flow)
xblocks(flow > 30, col = rgb[1]) ## high values red
xblocks(flow < 15, col = rgb[3]) ## low value blue
xblocks(flow >= 15 & flow <= 30, col = rgb[2]) ## the rest gray

## same thing:
plot(flow)
xblocks(time(flow), cut(flow, c(0,15,30,Inf), labels = rev(rgb)))
```

```
## another approach is to plot blocks underneath without transparency.
plot(flow)
## note that 'ifelse' keeps its result as class 'ts'
xblocks(ifelse(flow < mean(flow), hcl(0, 0, 90), hcl(0, 80, 70)))
## need to redraw data series on top:
lines(flow)
box()

## for single series only: plot.default has a panel.first argument
plot(time(flow), flow, type = "l",
  panel.first = xblocks(flow > 20, col = "lightgray"))
## (see also the 'panel' argument for use with multiple series, below)

## insert some missing values
flow[c(1:10, 50:80, 100)] <- NA

## the default plot shows data coverage
## (most useful when displaying multiple series, see below)
plot(flow)
xblocks(flow)

## can also show gaps:
plot(flow, type = "s")
xblocks(time(flow), is.na(flow), col = "gray")

## Example of alternating colors, here showing calendar months
flowdates <- as.Date("2000-01-01") + as.numeric(time(flow))
flowz <- zoo(coredata(flow), flowdates)
plot(flowz)
xblocks(flowz, months, ## i.e. months(time(flowz)),
  col = gray.colors(2, start = 0.7), border = "slategray")
lines(flowz)

## Example of multiple series.
## set up example data
z <- ts(cbind(A = 0:5, B = c(6:7, NA, NA, 10:11), C = c(NA, 13:17)))

## show data coverage only (highlighting gaps)
plot(z, panel = function(x, ...)
  xblocks(x, col = "darkgray"))

## draw gaps in darkgray
plot(z, type = "s", panel = function(x, ...) {
  xblocks(time(x), is.na(x), col = "darkgray")
  lines(x, ...); points(x)
})

## Example of overlaying blocks from a different series.
## Are US presidential approval ratings linked to sunspot activity?
## Set block height to plot blocks along the bottom.
plot(presidents)
xblocks(sunspot.year > 50, height = 2)
```

xyplot.zoo *Plot zoo Series with Lattice*

Description

xyplot methods for time series objects (of class "zoo", "its", or "tis").

Usage

```
## S3 method for class 'zoo'
xyplot(x, data, ...)

## S3 method for class 'zoo'
llines(x, y = NULL, ...)
## S3 method for class 'zoo'
lpoints(x, y = NULL, ...)
## S3 method for class 'zoo'
ltext(x, y = NULL, ...)

panel.segments.zoo(x0, x1, ...)
panel.rect.zoo(x0, x1, ...)
panel.polygon.zoo(x, ...)
```

Arguments

x, x0, x1	time series object of class "zoo", "its" or "tis". For panel.plot.default it should be a numeric vector.
y	numeric vector or matrix.
data	not used.
...	arguments are passed to xyplot.ts, and may be passed through to xyplot and panel.xyplot.

Some of the commonly used arguments are:

screens factor (or coerced to factor) whose levels specify which graph each series is to be plotted in. screens = c(1, 2, 1) would plot series 1, 2 and 3 in graphs 1, 2 and 1. This also defines the strip text in multi-panel plots.

scales the default is set so that all series have the "same" X axis but "free" Y axis. See xyplot in the **lattice** package for more information on scales.

layout numeric vector of length 2 specifying number of columns and rows in the plot, see xyplot for more details. The default is to fill columns with up to 6 rows.

xlab character string used as the X axis label.

ylab character string used as the Y axis label. If there are multiple panels it may be a character vector the same length as the number of panels, but *NOTE* in this case the vector should be reversed OR the argument as.table set to FALSE.

`lty, lwd, pch, type, col` graphical arguments passed to `panel.xyplot`. These arguments can also be vectors or (named) lists, see details for more information.

Details

`xyplot.zoo` plots a `"zoo"`, `"its"` or `"tis"` object using `xyplot.ts` from **lattice**. Series of other classes are coerced to `"zoo"` first.

The handling of several graphical parameters is more flexible for multivariate series. These parameters can be vectors of the same length as the number of series plotted or are recycled if shorter. They can also be (partially) named list, e.g., `list(A = c(1,2), c(3,4))` in which `c(3, 4)` is the default value and `c(1, 2)` the value only for series `A`. The `screens` argument can be specified in a similar way.

Note that since **zoo** 1.6-3 `plot.panel.default` and `plot.panel.custom` are no longer necessary, as normal panel functions (`panel.xyplot` by default) will work.

Similarly, there are now methods for the generic **lattice** drawing functions `llines`, `lpoints`, and `ltext`. These can also be called as `panel.lines`, `panel.points`, and `panel.text`, respectively. The old interfaces (`panel.lines.zoo`, `panel.points.zoo`, and `panel.text.zoo`), will be removed in future versions. `panel.polygon.zoo` may also be removed.

Value

Invisibly returns a `"trellis"` class object. Printing this object using `print` will display it.

See Also

`xyplot.ts`, `zoo`, `plot.ts`, `barplot`, `plot.zoo`

Examples

```
if(require("lattice") & require("grid")) {

suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
z <- zoo(cbind(a = 1:5, b = 11:15, c = 21:25) + rnorm(5))

# plot z using same Y axis on all plots
xyplot(z, scales = list(y = list(relation = "same", alternating = FALSE)))

# plot a double-line-width running mean on the panel of b.
# Also add a grid.
# We show two ways to do it.

# change strip background to levels of grey
# If you like the defaults, this can be omitted.
strip.background <- trellis.par.get("strip.background")
trellis.par.set(strip.background = list(col = grey(7:1/8)))


# Number 1.  Using trellis.focus.
```

```
print( xyplot(z) )
trellis.focus("panel", 1, 2, highlight = FALSE)
# (or just trellis.focus() for interactive use)
z.mean <- rollmean(z, 3)
panel.lines(z.mean[,2], lwd = 2)
panel.grid(h = 10, v = 10, col = "grey", lty = 3)
trellis.unfocus()

# Number 2. Using a custom panel routine.
xyplot(z, panel = function(x, y, ...) {
    if (packet.number() == 2) {
        panel.grid(h = 10, v = 10, col = "grey", lty = 3)
        panel.lines(rollmean(zoo(y, x), 3), lwd = 2)
    }
    panel.xyplot(x, y, ...)
})

# plot a light grey rectangle "behind" panel b
trellis.focus("panel", 1, 2)
grid.rect(x = 2, w = 1, default.units = "native",
  gp = gpar(fill = "light grey"))
# do.call("panel.xyplot", trellis.panelArgs())
do.call("panel.lines", trellis.panelArgs()[1:2])
trellis.unfocus()
# a better method is to use a custom panel function.
# see also panel.xblocks() and layer() in the latticeExtra package.

# same but make first panel twice as large as others
lopt <- list(layout.heights = list(panel = list(x = c(2,1,1))))
xyplot(z, lattice.options = lopt)
# add a grid
update(trellis.last.object(), type = c("l", "g"))

# Plot all in one panel.
xyplot(z, screens = 1)
# Same with default styles and auto.key:
xyplot(z, superpose = TRUE)

# Plot first two columns in first panel and third column in second panel.
# Plot first series using points, second series using lines and third
#  series via overprinting both lines and points
# Use colors 1, 2 and 3 for the three series (1=black, 2=red, 3=green)
# Make 2nd (lower) panel 3x the height of the 1st (upper) panel
# Also make the strip background orange.
p <- xyplot(z, screens = c(1,1,2), type = c("p", "l", "o"), col = 1:3,
  par.settings = list(strip.background = list(col = "orange")))
print(p, panel.height = list(y = c(1, 3), units = "null"))

# Example of using a custom axis
# Months are labelled with smaller ticks for weeks and even smaller
# ticks for days.
Days <- seq(from = as.Date("2006-1-1"), to = as.Date("2006-8-8"), by = "day")
z1 <- zoo(seq(length(Days))^2, Days)
```

```
Months <- Days[format(Days, "%d") == "01"]
Weeks <- Days[format(Days, "%w") == "0"]
print( xyplot(z1, scales = list(x = list(at = Months))) )
trellis.focus("panel", 1, 1, clip.off = TRUE)
panel.axis("bottom", check.overlap = TRUE, outside = TRUE, labels = FALSE,
  tck = .7, at = as.numeric(Weeks))
panel.axis("bottom", check.overlap = TRUE, outside = TRUE, labels = FALSE,
  tck = .4, at = as.numeric(Days))
trellis.unfocus()

trellis.par.set(strip.background = strip.background)

# separate the panels and suppress the ticks on very top
xyplot(z, between = list(y = 1), scales = list(tck = c(1,0)))

# left strips but no top strips
xyplot(z, screens = colnames(z), strip = FALSE, strip.left = TRUE)

# plot list of zoo objects using different x scales
z.l <- list(
 zoo(cbind(a = rnorm(10), b = rnorm(10)), as.Date("2006-01-01") + 0:9),
 zoo(cbind(c = rnorm(10), d = rnorm(10)), as.Date("2006-12-01") + 0:9)
)
zm <- do.call(merge, z.l)
xlim <- lapply(zm, function(x) range(time(na.omit(x))))
xyplot(zm, xlim = xlim, scale = list(relation = "free"))
# to avoid merging see xyplot.list() in the latticeExtra package.

}

## Not run:
## playwith (>= 0.9)
library("playwith")

z3 <- zoo(cbind(a = rnorm(100), b = rnorm(100) + 1), as.Date(1:100))
playwith(xyplot(z3), time.mode = TRUE)
# hold down Shift key and drag to zoom in to a time period.
# then use the horizontal scroll bar.

# set custom labels; right click on points to view or add labels
labs <- paste(round(z3,1), index(z3), sep = "@")
trellis.par.set(user.text = list(cex = 0.7))
playwith(xyplot(z3, type = "o"), labels = labs)

# this returns indexes into times of clicked points
ids <- playGetIDs()
z3[ids,]

## another example of using playwith with zoo
# set up data
dat <- zoo(matrix(rnorm(100*100),ncol=100), Sys.Date()+1:100)
colnames(dat) <- paste("Series", 1:100)
```

```
# This will give you a spin button to choose the column to plot,
# and a button to print out the current series number.
playwith(xyplot(dat[,c(1,i)]),  parameters = list(i = 1:100,
   do_something = function(playState) print(playState$env$i))


## End(Not run)
```

---

yearmon *An Index Class for Monthly Data*

Description

"yearmon" is a class for representing monthly data.

Usage

```
yearmon(x)
```

Arguments

x	numeric (interpreted as being "in years").

Details

The "yearmon" class is used to represent monthly data. Internally it holds the data as year plus 0 for January, 1/12 for February, 2/12 for March and so on in order that its internal representation is the same as ts class with frequency = 12. If x is not in this format it is rounded via floor(12*x + .0001)/12.

There are coercion methods available for various classes including: default coercion to "yearmon" (which coerces to "numeric" first) and coercions to and from "yearmon" to "Date" (see below), "POSIXct", "POSIXlt", "numeric", "character" and "jul". The last one is from the "tis" package available on CRAN. In the case of as.yearmon.POSIXt the conversion is with respect to GMT. (Use as.yearmon(format(...)) for other time zones.) In the case of as.yearmon.character the format argument uses the same percent code as "Date". These are described in strptime. Unlike "Date" one can specify a year and month with no day. Default formats of "%Y-%m", "%Y-%m-%d" and "%b %Y".

There is an is.numeric method which returns FALSE.

as.Date.yearmon and as.yearmon.yearqtr each has an optional second argument of "frac" which is a number between 0 and 1 inclusive that indicates the fraction of the way through the period that the result represents. The default is 0 which means the beginning of the period.

There is also a date method for as.yearmon usable with objects created with package date.

Sys.yearmon() returns the current year/month and methods for min, max and range are defined (by defining a method for Summary).

A yearmon mean method is also defined.

Value

Returns its argument converted to class yearmon.

See Also

yearqtr, zoo, zooreg, ts

Examples

```
Sys.setenv(TZ = "GMT")

x <- as.yearmon(2000 + seq(0, 23)/12)
x

as.yearmon("mar07", "%b%y")
as.yearmon("2007-03-01")
as.yearmon("2007-12")

# returned Date is the fraction of the way through
# the period given by frac (= 0 by default)
as.Date(x)
as.Date(x, frac = 1)
as.POSIXct(x)

# given a Date, x, return the Date of the next Friday
nextfri <- function(x) 7 * ceiling(as.numeric(x - 1)/7) + as.Date(1)

# given a Date, d, return the same Date in the following month
# Note that as.Date.yearmon gives first Date of the month.
d <- as.Date("2005-1-1") + seq(0,90,30)
next.month <- function(d) as.Date(as.yearmon(d) + 1/12) +
 as.numeric(d - as.Date(as.yearmon(d)))
next.month(d)

# 3rd Friday in last month of the quarter of Date x
## first day of last month of quarter
y <- as.Date(zoo::as.yearmon(zoo::as.yearqtr(x), frac = 1))
## number of days to first Friday
n <- sapply(y, function(z) which(format(z + 0:6, "%w") == "5")) - 1
## add number of days to third Friday
y + n + 14


suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

z <- zoo(rnorm(24), x, frequency = 12)
z
as.ts(z)

## convert data fram to multivariate monthly "ts" series
## 1.read raw data
```

```
Lines.raw <- "ID Date Count
123 20 May 1999 1
123 21 May 1999 3
222 1 Feb 2000 2
222 3 Feb 2000 4
"
DF <- read.table(text = Lines.raw, skip = 1,
 col.names = c("ID", "d", "b", "Y", "Count"))
## 2. fix raw date
DF$yearmon <- as.yearmon(paste(DF$b, DF$Y), "%b %Y")
## 3. aggregate counts over months, convert to zoo and merge over IDs
ag <- function(DF) aggregate(zoo(DF$Count), DF$yearmon, sum)
z <- do.call("merge.zoo", lapply(split(DF, DF$ID), ag))
## 4. convert to "zooreg" and then to "ts"
frequency(z) <- 12
as.ts(z)

xx <- zoo(seq_along(x), x)

## aggregating over year
as.year <- function(x) as.numeric(floor(as.yearmon(x)))
aggregate(xx, as.year, mean)
```

---

yearqtr *An Index Class for Quarterly Data*

Description

"yearqtr" is a class for representing quarterly data.

Usage

```
yearqtr(x)
as.yearqtr(x, ...)
## S3 method for class 'character'
as.yearqtr(x, format, ...)
## S3 method for class 'yearqtr'
format(x, format = "%Y Q%q", ...)
```

Arguments

x	for yearqtr a numeric (interpreted as being "in years"). For as.yearqtr another date class object. For the "yearqtr" method of format an object of class "yearqtr" or if called as format.yearqtr then an object with an as.yearqtr method that can be coerced to "yearqtr".
format	character string specifying format. For coercing to "yearqtr" from character: "%Y" and "%q" have to be specified. For formatting an existing "yearqtr": "%C", "%Y", "%y" and "%q", if present, are replaced with the century, year, last two digits of the year, and quarter (i.e. a number between 1 and 4), respectively.

... arguments passed ot other methods.

Details

The "yearqtr" class is used to represent quarterly data. Internally it holds the data as year plus 0 for Quarter 1, 1/4 for Quarter 2 and so on in order that its internal representation is the same as ts class with frequency = 4. If x is not in this format it is rounded via floor(4*x + .0001)/4.

as.yearqtr.character uses a default format of "%Y Q%q", "%Y q%q" or "%Y-%q" according to whichever matches. %q accepts the numbers 1-4 (possibly with leading zeros). Due to this %q does not match to single digits only and consequently formats such as as.yearqtr("Q12000", "Q%q%Y") are ambiguous and do not work (i.e., result in NA).

There are coercion methods available for various classes including: default coercion to "yearqtr" (which coerces to "numeric" first) and coercion from "yearqtr" to "Date" (see below), "POSIXct", "POSIXlt", "numeric", "character" and "jul". The last one is from the frame package on CRAN.

There is an is.numeric method which returns FALSE.

There is also a date method for as.yearqtr usable with objects created with package date.

Sys.yearqtr() returns the current year/month and methods for min, max and range are defined (by defining a method for Summary).

A yearqtr mean method is also defined.

Certain methods support a frac argument. See yearmon.

Value

yearqtr and as.yearqtr return the first argument converted to class yearqtr. The format method returns a character string representation of its argument first argument.

See Also

yearmon, zoo, zooreg, ts, strptime.

Examples

```
Sys.setenv(TZ = "GMT")

x <- as.yearqtr(2000 + seq(0, 7)/4)
x

format(x, "%Y Quarter %q")
as.yearqtr("2001 Q2")
as.yearqtr("2001 q2") # same
as.yearqtr("2001-2") # same

# returned Date is the fraction of the way through
# the period given by frac (= 0 by default)
dd <- as.Date(x)
format.yearqtr(dd)
as.Date(x, frac = 1)
```

```
as.POSIXct(x)

suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

zz <- zoo(rnorm(8), x, frequency = 4)
zz
as.ts(zz)
```

---

zoo *Z's Ordered Observations*

Description

zoo is the creator for an S3 class of indexed totally ordered observations which includes irregular time series.

Usage

```
zoo(x = NULL, order.by = index(x), frequency = NULL,
  calendar = getOption("zoo.calendar", TRUE))
## S3 method for class 'zoo'
print(x, style = , quote = FALSE, ...)
```

Arguments

x	a numeric vector, matrix or a factor.
order.by	an index vector with unique entries by which the observations in x are ordered. See the details for support of non-unique indexes.
frequency	numeric indicating frequency of order.by. If specified, it is checked whether order.by and frequency comply. If so, a regular "zoo" series is returned, i.e., an object of class c("zooreg", "zoo"). See below and zooreg for more details.
calendar	logical. If frequency is specified and is 4 or 12: Should yearqtr or yearmon be used for a numeric index order.by?
style	a string specifying the printing style which can be "horizontal" (the default for vectors), "vertical" (the default for matrices) or "plain" (which first prints the data and then the index).
quote	logical. Should characters be quoted?
...	further arguments passed to the print methods of the data and the index.

Details

zoo provides infrastructure for ordered observations which are stored internally in a vector or matrix with an index attribute (of arbitrary class, see below). The index must have the same length as NROW(x) except in the case of a zero length numeric vector in which case the index length can be any length. Emphasis has been given to make all methods independent of the index/time class (given in order.by). In principle, the data x could also be arbitrary, but currently there is only support for vectors and matrices and partial support for factors.

zoo is particularly aimed at irregular time series of numeric vectors/matrices, but it also supports regular time series (i.e., series with a certain frequency). zoo's key design goals are independence of a particular index/date/time class and consistency with ts and base R by providing methods to standard generics. Therefore, standard functions can be used to work with "zoo" objects and memorization of new commands is reduced.

When creating a "zoo" object with the function zoo, the vector of indexes order.by can be of (a single) arbitrary class (if x is shorter or longer than order.by it is expanded accordingly), but it is essential that ORDER(order.by) works. For other functions it is assumed that c(), length(), MATCH() and subsetting [, work. If this is not the case for a particular index/date/time class, then methods for these generic functions should be created by the user. Note, that to achieve this, new generic functions ORDER and MATCH are created in the zoo package with default methods corresponding to the non-generic base functions order and match. Note that the order and hence the default ORDER typically work if there is a xtfrm method. Furthermore, for certain (but not for all) operations the index class should have an as.numeric method (in particular for regular series) and an as.character method might improve printed output (see also below).

The index observations order.by should typically be unique, such that the observations can be totally ordered. Nevertheless, zoo() is able to create "zoo" objects with duplicated indexes (with a warning) and simple methods such as plot() or summary() will typically work for such objects. However, this is not formally supported as the bulk of functionality provided in **zoo** requires unique index observations/time stamps. See below for an example how to remove duplicated indexes.

If a frequency is specified when creating a series via zoo, the object returned is actually of class "zooreg" which inherits from "zoo". This is a subclass of "zoo" which relies on having a "zoo" series with an additional "frequency" attribute (which has to comply with the index of that series). Regular "zooreg" series can also be created by zooreg, the zoo analogue of ts. See the respective help page and is.regular for further details.

Methods to standard generics for "zoo" objects currently include: print (see above), summary, str, head, tail, [(subsetting), rbind, cbind, merge (see merge.zoo), aggregate (see aggregate.zoo), rev, split (see aggregate.zoo), barplot, plot and lines (see plot.zoo). For multivariate "zoo" series with column names the $ extractor is available, behaving similar as for "data.frame" objects. Methods are also available for median and quantile.

ifelse.zoo is not a method (because ifelse is not a generic) but must be written out including the .zoo suffix.

To "prettify" printed output of "zoo" series the generic function index2char is used for turning index values into character values. It defaults to using as.character but can be customized if a different printed display should be used (although this should not be necessary, usually).

The subsetting method [work essentially like the corresponding functions for vectors or matrices respectively, i.e., takes indexes of type "numeric", "integer" or "logical". But additionally, it can be used to index with observations from the index class of the series. If the index class of the

series is one of the three classes above, the corresponding index has to be encapsulated in `I()` to enforce usage of the index class (see examples). Subscripting by a zoo object whose data contains logical values is undefined.

Additionally, `zoo` provides several generic functions and methods to work (a) on the data contained in a `"zoo"` object, (b) the index (or time) attribute associated to it, and (c) on both data and index:

(a) The data contained in `"zoo"` objects can be extracted by `coredata` (strips off all `"zoo"`-specific attributes) and modified using `coredata<-`. Both are new generic functions with methods for `"zoo"` objects, see `coredata`.

(b) The index associated with a `"zoo"` object can be extracted by `index` and modified by `index<-`. As the interpretation of the index as "time" in time series applications is more natural, there are also synonymous methods `time` and `time<-`. The start and the end of the index/time vector can be queried by `start` and `end`. See `index`.

(c) To work on both data and index/time, `zoo` provides methods `lag`, `diff` (see `lag.zoo`) and `window`, `window<-` (see `window.zoo`).

In addition to standard group generic function (see `Ops`), the following mathematical operations are available as methods for `"zoo"` objects: transpose `t` which coerces to a matrix first, and `cumsum`, `cumprod`, `cummin`, `cummax` which are applied column wise.

Coercion to and from `"zoo"` objects is available for objects of various classes, in particular `"ts"`, `"irts"` and `"its"` objects can be coerced to `"zoo"`, the reverse is available for `"its"` and for `"irts"` (the latter in package `tseries`). Furthermore, `"zoo"` objects can be coerced to vectors, matrices and lists and data frames (dropping the index/time attribute). See `as.zoo`.

Several methods are available for `NA` handling in the data of `"zoo"` objects: `na.aggregate` which uses group means to fill in `NA` values, `na.approx` which uses linear interpolation to fill in `NA` values. `na.contiguous` which extracts the longest consecutive stretch of non-missing values in a `"zoo"` object, `na.fill` which uses fixed specified values to replace `NA` values, `na.locf` which replaces `NA`s by the last previous non-`NA`, `na.omit` which returns a `"zoo"` object with incomplete observations removed, `na.spline` which uses spline interpolation to fill in `NA` values and `na.StructTS` which uses a seasonal Kalman filter to fill in `NA` values, `na.trim` which trims runs of `NA`s off the beginning and end but not in the interior. Yet another `NA` routine can be found in the `stinepack` package where `na.stinterp` performs Stineman interpolation.

A typical task to be performed on ordered observations is to evaluate some function, e.g., computing the mean, in a window of observations that is moved over the full sample period. The generic function `rollapply` provides this functionality for arbitrary functions and more efficient versions `rollmean`, `rollmax`, `rollmedian` are available for the mean, maximum and median respectively.

The **zoo** package has an `as.Date numeric` method which is similar to the one in the core of R except that the `origin` argument defaults to January 1, 1970 (whereas the one in the core of R has no default).

Note that since `zoo` uses date/time classes from base R and other packages, it may inherit bugs or problems with those date/time classes. Currently, there is one such known problem with the `c` method for the `POSIXct` class in base R: If `x` and `y` are `POSIXct` objects with `tzone` attributes, the attribute will always be dropped in `c(x, y)`, even if it is the same across both `x` and `y`. Although this is documented at `c.POSIXct`, one may want to employ a workaround as shown at https://stat.ethz.ch/pipermail/r-devel/2010-August/058112.html.

Value

A vector or matrix with an "index" attribute of the same dimension (NROW(x)) by which x is ordered.

References

Achim Zeileis and Gabor Grothendieck (2005). **zoo**: S3 Infrastructure for Regular and Irregular Time Series. *Journal of Statistical Software*, **14(6)**, 1-27. URL http://www.jstatsoft.org/v14/i06/ and available as vignette("zoo").

Ajay Shah, Achim Zeileis and Gabor Grothendieck (2005). **zoo** Quick Reference. Package vignette available as vignette("zoo-quickref").

See Also

zooreg, plot.zoo, index, merge.zoo

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)

## simple creation and plotting
x.Date <- as.Date("2003-02-01") + c(1, 3, 7, 9, 14) - 1
x <- zoo(rnorm(5), x.Date)
plot(x)
time(x)

## subsetting with numeric indexes
x[c(2, 4)]
## subsetting with index class
x[as.Date("2003-02-01") + c(2, 8)]

## different classes of indexes/times can be used, e.g. numeric vector
x <- zoo(rnorm(5), c(1, 3, 7, 9, 14))
## subsetting with numeric indexes then uses observation numbers
x[c(2, 4)]
## subsetting with index class can be enforced by I()
x[I(c(3, 9))]

## visualization
plot(x)
## or POSIXct
y.POSIXct <- ISOdatetime(2003, 02, c(1, 3, 7, 9, 14), 0, 0, 0)
y <- zoo(rnorm(5), y.POSIXct)
plot(y)

## create a constant series
z <- zoo(1, seq(4)[-2])

## create a 0-dimensional zoo series
z0 <- zoo(, 1:4)
```

```
## create a 2-dimensional zoo series
z2 <- zoo(matrix(1:12, 4, 3), as.Date("2003-01-01") + 0:3)

## create a factor zoo object
fz <- zoo(gl(2,5), as.Date("2004-01-01") + 0:9)

## create a zoo series with 0 columns
z20 <- zoo(matrix(nrow = 4, ncol = 0), 1:4)

## arithmetic on zoo objects intersects them first
x1 <- zoo(1:5, 1:5)
x2 <- zoo(2:6, 2:6)
10 * x1 + x2

## $ extractor for multivariate zoo series with column names
z <- zoo(cbind(foo = rnorm(5), bar = rnorm(5)))
z$foo
z$xyz <- zoo(rnorm(3), 2:4)
z

## add comments to a zoo object
comment(x1) <- c("This is a very simple example of a zoo object.",
  "It can be recreated using this R code: example(zoo)")
## comments are not output by default but are still there
x1
comment(x1)

# ifelse does not work with zoo but this works
# to create a zoo object which equals x1 at
# time i if x1[i] > x1[i-1] and 0 otherwise
(diff(x1) > 0) * x1

## zoo series with duplicated indexes
z3 <- zoo(1:8, c(1, 2, 2, 2, 3, 4, 5, 5))
plot(z3)
## remove duplicated indexes by averaging
lines(aggregate(z3, index(z3), mean), col = 2)
## or by using the last observation
lines(aggregate(z3, index(z3), tail, 1), col = 4)

## x1[x1 > 3] is not officially supported since
## x1 > 3 is of class "zoo", not "logical".
## Use one of these instead:
x1[which(x1 > 3)]
x1[coredata(x1 > 3)]
x1[as.logical(x1 > 3)]
subset(x1, x1 > 3)

## any class supporting the methods discussed can be used
## as an index class. Here are examples using complex numbers
## and letters as the time class.
```

```
z4 <- zoo(11:15, complex(real = c(1, 3, 4, 5, 6), imag = c(0, 1, 0, 0, 1)))
merge(z4, lag(z4))

z5 <- zoo(11:15, letters[1:5])
merge(z5, lag(z5))

# index values relative to 2001Q1
zz <- zooreg(cbind(a = 1:10, b = 11:20), start = as.yearqtr(2000), freq = 4)
zz[] <- mapply("/", as.data.frame(zz), coredata(zz[as.yearqtr("2001Q1")]))


## even though time index must be unique zoo (and read.zoo)
## will both allow creation of such illegal objects with
## a warning (rather than ana error) to give the user a
## chance to fix them up.  Extracting and replacing times
## and aggregate.zoo will still work.
## Not run:
# this gives a warning
# and then creates an illegal zoo object
z6 <- zoo(11:15, c(1, 1, 2, 2, 5))
z6

# fix it up by averaging duplicates
aggregate(z6, identity, mean)

# or, fix it up by taking last in each set of duplicates
aggregate(z6, identity, tail, 1)

# fix it up via interpolation of duplicate times
time(z6) <- na.approx(ifelse(duplicated(time(z6)), NA, time(z6)), na.rm = FALSE)
# if there is a run of equal times at end they
# wind up as NAs and we cannot have NA times
z6 <- z6[!is.na(time(z6))]
z6

x1. <- x1 <- zoo (matrix (1:12, nrow = 3), as.Date("2008-08-01") + 0:2)
colnames (x1) <- c ("A", "B", "C", "D")
x2 <- zoo (matrix (1:12, nrow = 3), as.Date("2008-08-01") + 1:3)
colnames (x2) <- c ("B", "C", "D", "E")

both.dates = as.Date (intersect (index (t1), index (t2)))
both.cols = intersect (colnames (t1), colnames (t2))

x1[both.dates, both.cols]
## there is "[.zoo" but no "[<-.zoo" however four of the following
## five examples work

## wrong
## x1[both.dates, both.cols] <- x2[both.dates, both.cols]

# 4 correct alternatives
# #1
window(x1, both.dates)[, both.cols] <- x2[both.dates, both.cols]
```

```
# #2. restore x1 and show a different way
x1 <- x1.
window(x1, both.dates)[, both.cols] <- window(x2, both.dates)[, both.cols]

# #3. restore x1 and show a different way
x1 <- x1.
x1[time(x1)

# #4. restore x1 and show a different way
x1 <- x1.
x1[time(x1)


## End(Not run)
```

---

zooreg *Regular zoo Series*

Description

zooreg is the creator for the S3 class "zooreg" for regular "zoo" series. It inherits from "zoo" and is the analogue to ts.

Usage

```
zooreg(data, start = 1, end = numeric(), frequency = 1,
  deltat = 1, ts.eps = getOption("ts.eps"), order.by = NULL,
  calendar = getOption("zoo.calendar", TRUE))
```

Arguments

data	a numeric vector, matrix or a factor.
start	the time of the first observation. Either a single number or a vector of two integers, which specify a natural time unit and a (1-based) number of samples into the time unit.
end	the time of the last observation, specified in the same way as start.
frequency	the number of observations per unit of time.
deltat	the fraction of the sampling period between successive observations; e.g., 1/12 for monthly data. Only one of frequency or deltat should be provided.
ts.eps	time series comparison tolerance. Frequencies are considered equal if their absolute difference is less than ts.eps.
order.by	a vector by which the observations in x are ordered. If this is specified the arguments start and end are ignored and zoo(data, order.by, frequency) is called. See zoo for more information.
calendar	logical. Should yearqtr or yearmon be used for a numeric time index with frequency 4 or 12, respectively?

Details

Strictly regular series are those whose time points are equally spaced. Weakly regular series are strictly regular time series in which some of the points may have been removed but still have the original underlying frequency associated with them. "zooreg" is a subclass of "zoo" that is used to represent both weakly and strictly regular series. Internally, it is the same as "zoo" except it also has a "frequency" attribute. Its index class is more restricted than "zoo". The index: 1. must be numeric or a class which can be coerced via as.numeric (such as yearmon, yearqtr, Date, POSIXct, tis, xts, etc.). 2. when converted to numeric must be expressible as multiples of 1/frequency. 3. group generic functions Ops should be defined, i.e., adding/subtracting a numeric to/from the index class should produce the correct value of the index class again.

zooreg is the zoo analogue to ts. The arguments are almost identical, only in the case where order.by is specified, zoo is called with zoo(data, order.by, frequency). It creates a regular series of class "zooreg" which inherits from "zoo". It is essentially a "zoo" series with an additional "frequency" attribute. In the creation of "zooreg" objects (via zoo, zooreg, or coercion functions) it is always check whether the index specified complies with the frequency specified.

The class "zooreg" offers two advantages over code "ts": 1. The index does not have to be plain numeric (although that is the default), it just must be coercible to numeric, thus printing and plotting can be customized. 2. This class can not only represent strictly regular series, but also series with an underlying regularity, i.e., where some observations from a regular grid are omitted.

Hence, "zooreg" is a bridge between "ts" and "zoo" and can be employed to coerce back and forth between the two classes. The coercion function as.zoo.ts returns therefore an object of class "zooreg" inheriting from "zoo". Coercion between "zooreg" and "zoo" is also available and drops or tries to add a frequency respectively.

For checking whether a series is strictly regular or does have an underlying regularity the generic function is.regular can be used.

Methods to standard generics for regular series such as frequency, deltat and cycle are available for both "zooreg" and "zoo" objects. In the latter case, it is checked first (in a data-driven way) whether the series is in fact regular or not.

as.zooreg.tis has a class argument whose value represents the class of the index of the zooreg object into which the tis object is converted. The default value is "ti". Note that the frequency of the zooreg object will not necessarily be the same as the frequency of the tis object that it is converted from.

Value

An object of class "zooreg" which inherits from "zoo". It is essentially a "zoo" series with a "frequency" attribute.

See Also

zoo, is.regular

Examples

```
## equivalent specifications of a quarterly series
## starting in the second quarter of 1959.
zooreg(1:10, frequency = 4, start = c(1959, 2))
```

```
as.zoo(ts(1:10, frequency = 4, start = c(1959, 2)))
zoo(1:10, seq(1959.25, 1961.5, by = 0.25), frequency = 4)

## use yearqtr class for indexing the same series
z <- zoo(1:10, yearqtr(seq(1959.25, 1961.5, by = 0.25)), frequency = 4)
z
z[-(3:4)]

## create a regular series with a "Date" index
zooreg(1:5, start = as.Date("2000-01-01"))
## or with "yearmon" index
zooreg(1:5, end = yearmon(2000))

## lag and diff (as diff is defined in terms of lag)
## act differently on zoo and zooreg objects!
## lag.zoo moves a point to the adjacent time whereas
## lag.zooreg moves a point by deltat
x <- c(1, 2, 3, 6)
zz <- zoo(x, x)
zr <- as.zooreg(zz)
lag(zz, k = -1)
lag(zr, k = -1)
diff(zz)
diff(zr)

## lag.zooreg wihtout and with na.pad
lag(zr, k = -1)
lag(zr, k = -1, na.pad = TRUE)

## standard methods available for regular series
frequency(z)
deltat(z)
cycle(z)
cycle(z[-(3:4)])

zz  <-  zoo(1:6, as.Date(c("1960-01-29", "1960-02-29", "1960-03-31",
  "1960-04-29", "1960-05-31", "1960-06-30")))
# this converts zz to "zooreg" and then to "ts" expanding it to a daily
# series which is 154 elements long, most with NAs.
## Not run:
length(as.ts(zz)) # 154

## End(Not run)
# probably a monthly "ts" series rather than a daily one was wanted.
# This variation of the last line gives a result only 6 elements long.
length(as.ts(aggregate(zz, as.yearmon, c))) # 6

zzr <- as.zooreg(zz)

dd <- as.Date(c("2000-01-01", "2000-02-01", "2000-03-01", "2000-04-01"))
zrd <- as.zooreg(zoo(1:4, dd))
```

Index