

# The zoo Package

July 9, 2008

**Version** 1.5-4

**Date** 2008-07-09

**Title** Z's ordered observations

**Author** Achim Zeileis, Gabor Grothendieck

**Maintainer** Achim Zeileis <Achim.Zeileis@R-project.org>

**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.

**Depends** R (>= 2.4.1), stats

**Suggests** chron, fCalendar, fSeries, its, tseries, lattice, strucchange, DAAG, xts

**Imports** stats, utils, graphics, grDevices, lattice

**LazyLoad** yes

**License** GPL-2

**URL** <http://R-Forge.R-project.org/projects/zoo/>

## R topics documented:

MATCH . . . . .	2
ORDER . . . . .	3
aggregate.zoo . . . . .	3
as.Date.numeric . . . . .	6
as.zoo . . . . .	7
coredata . . . . .	9
frequency<- . . . . .	10
index . . . . .	10
is.regular . . . . .	12
lag.zoo . . . . .	13
make.par.list . . . . .	14
merge.zoo . . . . .	15

na.approx . . . . .	17
na.locf . . . . .	18
na.trim . . . . .	20
plot.zoo . . . . .	21
read.zoo . . . . .	25
rollapply . . . . .	28
rollmean . . . . .	29
window.zoo . . . . .	31
xyplot.zoo . . . . .	32
yearmon . . . . .	36
yearqtr . . . . .	38
zoo . . . . .	39
zooreg . . . . .	44

<b>Index</b>	<b>47</b>
--------------	-----------

---

MATCH	<i>Value Matching</i>
-------	-----------------------

---

### Description

MATCH is a generic function for value matching.

### Usage

```
MATCH(x, table, nomatch = NA, ...)
```

### 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 <code>integer</code> .
...	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 only a default method which simply calls `match`.

### See Also

[match](#)

### Examples

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

---

ORDER	<i>Ordering Permutation</i>
-------	-----------------------------

---

**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`.

**See Also**

`order`

**Examples**

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

---

aggregate.zoo	<i>Compute Summary Statistics of zoo Objects</i>
---------------	--

---

**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, ..., regular = NULL, frequency = NULL)
```

**Arguments**

<code>x</code>	an object of class "zoo".
<code>by</code>	index vector of the same length as <code>index(x)</code> which defines aggregation groups and the new index to be associated with each group. If <code>by</code> is a function, then it is applied to <code>index(x)</code> to obtain the aggregation groups.
<code>FUN</code>	a scalar function to compute the summary statistics which can be applied to all subsets.
<code>...</code>	further arguments passed to <code>FUN</code> .
<code>regular</code>	logical. Should the aggregated series be coerced to class "zooreg" (if the series is regular)? The default is <code>FALSE</code> for "zoo" series and <code>TRUE</code> for "zooreg" series.
<code>frequency</code>	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 <code>regular</code> is <code>TRUE</code> . If <code>frequency</code> is specified, it sets <code>regular</code> to <code>TRUE</code> . See examples for illustration.

**Value**

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

**See Also**

[zoo](#)

**Examples**

```
## averaging over values in a month:
# long series
x.date <- as.Date(paste(2004, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
x <- zoo(rnorm(12), x.date)
# coarser dates
x.date2 <- as.Date(paste(2004, rep(1:4, 4:1), 1, sep = "-"))
x2 <- aggregate(x, x.date2, mean)
# 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 explicitly
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 representative 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
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
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)

```

---

as.Date.numeric      *Date Conversion Functions from Numeric, Integer and ts Objects*

---

### Description

Functions to convert numeric and related classes to objects of class "Date" representing calendar dates.

### Usage

```

## S3 method for class 'numeric':
as.Date(x, origin = "1970-01-01", ...)
## S3 method for class 'ts':
as.Date(x, offset = 0, ...)

```

### Arguments

x	numeric or an object of class "ts", respectively.
origin	A specification of the origin for the dates (x days since origin).
offset	A value added to time(x).
...	Further arguments. Currently not used.

## Details

The `as.Date` method for numeric arguments `x` interprets `x` the number of days since the `origin` (default: 1970-01-01). Negative values are allowed.

The `as.Date.ts` inspects `time(x)` and `frequency(x)`: If the frequency is 1 or 4 or 12, `time(x)` is regarded to be annual, quarterly or monthly data respectively. If the frequency is something else, no coercion is done.

## Value

The `as.Date` methods return an object of class `"Date"`. In the case of `as.Date.ts` applied to a yearly, quarterly or monthly series the earliest date, i.e., first of year, quarter or month, is returned.

## See Also

[Date](#) for details of the date class

## Examples

```
# uses origin = "1970-01-01"
as.Date(0)

# all three result in origin and next 9 days
as.Date(0:9)
as.Date(0) + 0:9
x <- ts(rnorm(10), start = 0)
as.Date(unclass(time(x)))

# annual/quarterly/monthly series:
xa <- ts(rnorm(10), start = 2001)
xq <- ts(rnorm(10), start = 2001, freq = 4)
xm <- ts(rnorm(10), start = 2001, freq = 12)
as.Date(time(xa))
as.Date(time(xq))
as.Date(time(xm))

# using offset argument
xa2 <- ts(rnorm(10))
as.Date(time(xa2), offset = 2000)
```

## 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`, `its` and `irts` 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 the coercion function `as.its.zoo` is provided, but works only if the corresponding package is attached.

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

**Value**

`as.zoo` returns a `zoo` object.

**See Also**

`zoo`, `zooreg`, `ts`, `its`, `irts`

**Examples**

```
## 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)
```

**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. There are also corresponding replacement methods for these classes.

**See Also**

[zoo](#)

**Examples**

```
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)
```

---

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

```
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)

## 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, ...)

```

## Arguments

x	a "zoo" object.
k, lag	the number of lags (in units of observations). Note the sign of k behaves as in <a href="#">lag</a> .
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.
...	currently not used.

**Details**

These methods for "zoo" objects behave analogously to the default methods. The only additional arguments are `arithmetic` in `diff` `na.pad` in `lag.zoo` which can also be specified in `diff.zoo` as part of the dots. 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^2)
diff(x^2, na.pad = TRUE)
```

---

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)
```

---

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,
      retclass = c("zoo", "list", "data.frame"))
```

**Arguments**

<code>...</code>	two or more objects, usually of class "zoo".
<code>all</code>	logical vector having the same length as the number of "zoo" objects to be merged (otherwise expanded).
<code>fill</code>	an element for filling gaps in merged "zoo" objects (if any).
<code>suffixes</code>	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.
<code>retclass</code>	character that specifies the class of the returned result. It can be "zoo" (the default), "list" or NULL. For details see below.

**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, coredat(z), 1)
```

---

na.approx

*Replace NA by Interpolation*


---

**Description**

Generic functions for replacing each NA with interpolated values.

**Usage**

```
na.approx(object, ...)
## Default S3 method:
na.approx(object, along = index(object), na.rm = TRUE, ...)

na.spline(object, ...)
## Default S3 method:
na.spline(object, along = index(object), na.rm = TRUE, ...)
```

**Arguments**

<code>object</code>	object in which NAs are to be replaced
<code>along</code>	variable to use for interpolation. Has to be numeric, is otherwise coerced to numeric.
<code>na.rm</code>	logical. Should leading NAs be removed?
<code>...</code>	further arguments passed to methods.

**Details**

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

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.

**Value**

An object in which each NA in the input object is replaced by interpolating the non-NA values before and after it. Leading NAs are omitted (if `na.rm = TRUE`) or not replaced (if `na.rm = FALSE`).

**See Also**

[zoo](#), [approx](#), [na.contiguous](#), [link{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)
```

---

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, ...)
```

**Arguments**

<code>object</code>	an object.
<code>na.rm</code>	logical. Should leading NAs be removed?
<code>fromLast</code>	logical. Causes observations to be carried backward rather than forward. Default is <code>FALSE</code> . This corresponds to NOCB (next observation carried backward).
<code>rev</code>	Use <code>fromLast</code> instead. This argument will be eliminated in the future in favor of <code>fromLast</code> .
<code>...</code>	further arguments passed to methods.

**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`).

**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
# by merging with a zero width series having those dates
# and then applying na.locf
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")))
dd <- seq(start(z), end(z), "day")
na.locf(merge(z, zoo(, dd)))
```

---

`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"), ...)
```

## Arguments

<code>object</code>	an object.
<code>sides</code>	character specifying whether NAs are to be removed from both sides, just from the left side or just from the right side.
<code>...</code>	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))
```

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,
     oma = c(6, 0, 5, 0), mar = c(0, 5.1, 0, 2.1),
     col = 1, lty = 1, pch = 1, type = "l", 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**

<code>x</code>	an object of class "zoo".
<code>y</code>	an object of class "zoo". If <code>y</code> is <code>NULL</code> (the default) a time series plot of <code>x</code> is produced, otherwise if both <code>x</code> and <code>y</code> are univariate "zoo" series, a scatter plot of <code>y</code> versus <code>x</code> is produced.
<code>screens</code>	factor (or coerced to factor) whose levels specify which graph each series is to be plotted in. <code>screens=c(1,2,1)</code> would plot series 1, 2 and 3 in graphs 1, 2 and 1. If not specified then 1 is used if <code>plot.type="single"</code> and <code>seq_len(ncol(x))</code> otherwise.
<code>plot.type</code>	for multivariate zoo objects, "multiple" plots the series on multiple plots and "single" superimposes them on a single plot. Default is "single" if <code>screens</code> has only one level and "multiple" otherwise. If neither <code>screens</code> nor <code>plot.type</code> is specified then "single" is used if there is one series and "multiple" otherwise. This option is provided for back compatibility. Usually <code>screens</code> is used instead.
<code>panel</code>	a function( <code>x</code> , <code>y</code> , <code>col</code> , <code>lty</code> , ...) which gives the action to be carried out in each panel of the display for <code>plot.type = "multiple"</code> .
<code>ylim</code>	if <code>plot.type = "multiple"</code> 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 <code>plot.type = "single"</code> then it is as in plot.
<code>xy.labels</code>	logical, indicating if <code>text</code> labels should be used in the scatter plot, or character, supplying a vector of labels to be used.
<code>xy.lines</code>	logical, indicating if <code>lines</code> should be drawn in the scatter plot. Defaults to the value of <code>xy.labels</code> if that is logical, otherwise to <code>FALSE</code> .

`xlab, ylab, main, xlim, oma, mar`  
 graphical arguments, see [par](#).

`col, lty, pch, type`  
 graphical arguments that can be vectors or (named) lists. See the details for more information.

`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 multipanel 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.

In addition to classical time series line plots, there is also a simple `barplot` method for "zoo" series.

### See Also

[zoo](#), [plot.ts](#), [barplot](#), [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 multiple 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)
  # 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
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)

# automatically placed point labels
## Not run:

```

```

library("mapproj")
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)

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

# interactive plotting
## Not run:
library("TeachingDemos")
tke.test1 <- list(Parameters = list(
  pch = list("spinbox", init = 1, from = 0, to = 255, width = 5),
  cex = list("slider", init = 1.5, from = 0.1, to = 5, resolution = 0.1),
  type = list("combobox", init = "b",
    values = c("p", "l", "b", "o", "c", "h", "s", "S", "n"), width = 5),
  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)

```

---

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.

## Usage

```

read.zoo(file, format = "", tz = "", FUN = NULL,
  regular = FALSE, index.column = 1, aggregate = FALSE, ...)
write.zoo(x, file = "", index.name = "Index", row.names = FALSE, col.names = NULL,

```

## Arguments

`file` character giving the name of the file which the data are to be read from/written to. See [read.table](#) and [write.table](#) for more information.

`format` date format argument passed to `FUN`.

<code>tz</code>	time zone argument passed to <code>as.POSIXct</code> .
<code>FUN</code>	a function for computing the index from the first column of the data. See details.
<code>regular</code>	logical. Should the series be coerced to class "zooreg" (if the series is regular)?
<code>index.column</code>	integer. The column of the data frame in which the index/time is stored.
<code>x</code>	a "zoo" object.
<code>index.name</code>	character with name of the index column in the written data file.
<code>row.names</code>	logical. Should row names be written? Default is FALSE because the row names are just character representations of the index.
<code>col.names</code>	logical. Should column names be written? Default is to write column names only if <code>x</code> has column names.
<code>aggregate</code>	logical or function. If set to TRUE, then <code>aggregate.zoo</code> is applied to the zoo object created to compute the <code>mean</code> of all values with the same time index. Alternatively, <code>aggregate</code> 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 <code>aggregate.zoo</code> .
<code>...</code>	further arguments passed to <code>read.table</code> or <code>write.table</code> , respectively.

### 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` uses the first of the following that is true: 1. If `FUN` is specified then `read.zoo` calls `FUN` with the index as the first argument. 2. If `tz` is specified then the index column is converted to `POSIXct`. 3. If `format` is specified then the index column is converted to `Date`. 4. A heuristic attempts to decide among "numeric", "Date" and "POSIXct". If `format` and/or `tz` is specified then it is passed to the conversion function as well.

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

`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`.

### 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 builtin `date`

conversion routine a number, rather than a character string. Thus a date field such as code09122007 intended to represent December 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

```
## Not run:
## 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")

# ensure that first column is interpreted as character so leading 0 not dropped
# Suppose links look like: 09112007 1 2 and there is no header
z <- read.zoo("foo.txt", format = "%d%m%Y", colClasses = c(V1 = "character"))

## 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 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)

## End(Not run)
```

---

rollapply	<i>Apply Rolling Functions</i>
-----------	--------------------------------

---

**Description**

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

**Usage**

```
rollapply(data, width, FUN, ..., by = 1, ascending = TRUE, by.column = TRUE,
  na.pad = FALSE, align = c("center", "left", "right"))
```

**Arguments**

data	the data to be used (representing a series of observations).
width	number of points per group.
FUN	the function to be applied. In the case of functions like <code>+</code> , <code>%*%</code> , etc., the function name must be quoted.
...	optional arguments to FUN.
by	calculate FUN for trailing width points at every by-th time point.
ascending	logical. If TRUE then points are passed to FUN in ascending order of time; otherwise, they are passed in descending order.
by.column	logical. If TRUE, FUN is applied to each column separately.
na.pad	logical. If TRUE then additional elements or rows of NAs are added so that result has same number of elements or rows as data.
align	character specifying whether result should be left- or right-aligned or centered (default).

**Details**

Groups time points in successive sets of `width` time points and applies `FUN` to the corresponding values. If `FUN` is `mean`, `max` or `median` and `by.column` is `TRUE` and there are no extra arguments then special purpose code is used to enhance performance. See [rollmean](#), [rollmax](#) and [rollmedian](#) for more details.

Currently, there are methods for "zoo" and "ts" series.

In previous versions, this function was called `rapply`. It was renamed because from R 2.4.0 on, base R provides a different function `rapply` for recursive (and not rolling) application of functions.

**Value**

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

**See Also**

[rollmean](#)

**Examples**

```
## 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)
fm <- 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
plot(fm)
## showing the shifts after the oil crisis in Oct 1973
## and after the seatbelt legislation change in Jan 1983
```

---

rollmean

*Rolling Means/Maximums/Medians*


---

**Description**

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

**Usage**

```
rollmean(x, k, na.pad = FALSE, align = c("center", "left", "right"), ...)
rollmax(x, k, na.pad = FALSE, align = c("center", "left", "right"), ...)
rollmedian(x, k, na.pad = FALSE, align = c("center", "left", "right"), ...)
```

**Arguments**

<code>x</code>	an object (representing a series of observations).
<code>k</code>	integer width of the rolling window. Must be odd for <code>rollmedian</code> .
<code>na.pad</code>	logical. Should NA padding be added at beginning?
<code>align</code>	character specifying whether result should be left- or right-aligned or centered (default).
<code>...</code>	Further arguments passed to methods.

**Details**

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

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

**Value**

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

**See Also**

`rollapply`, `zoo`

**Examples**

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

rollmean(x, 3)
rollmax(x, 3)
rollmedian(x, 3)

xm <- zoo(matrix(1:12, 4, 3), x.Date[1:4])
rollmean(xm, 3)
rollmax(xm, 3)
rollmedian(xm, 3)

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.zoo(x), start = NULL, end = NULL, ...)
## S3 replacement method for class 'zoo':
window(x, index = index.zoo(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 <code>index</code> which are greater or equal to <code>start</code> are used. If the index class supports comparisons to character variables, as does "Date" class, "yearmon" class, "yearqtr" class and the <code>chron</code> package classes "dates" and "times" then <code>start</code> may alternately be a character variable.
end	an index/time value. Only the indexes in <code>index</code> which are lower or equal to <code>end</code> are used. Similar comments about character variables mentioned under <code>start</code> apply here too.
value	a suitable value object for use with <code>window(x)</code> .
...	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

```
## 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))

```

---

xyplot.zoo

*Plot zoo Series with Lattice*


---

### Description

xyplot methods for time series objects (of class "zoo" or "ts", "its"). These functions are still under development and the interface and functionality might be modified/extended in future releases.

### Usage

```

## S3 method for class 'zoo':
xyplot(x, data, screens = seq(length = NCOL(x)),
      default.scales = list(y = list(relation = "free")),
      layout = NULL, xlab = "Index", ylab = NULL,
      lty = trellis.par.get("plot.line")$lty,
      lwd = trellis.par.get("plot.line")$lwd,
      pch = trellis.par.get("plot.symbol")$pch,
      type = "l",
      col = trellis.par.get("plot.line")$col,
      strip = TRUE, panel = panel.plot.default, ...)

panel.plot.default(x, y, subscripts, groups,
  panel = panel.xyplot,
  col = 1, type = "p", pch = 20, lty = 1, lwd = 1, ...)

panel.plot.custom(...)

panel.lines.zoo(x, ...)
panel.points.zoo(x, ...)
panel.segments.zoo(x0, x1, ...)
panel.text.zoo(x, ...)
panel.rect.zoo(x0, x1, ...)
panel.arrows.zoo(x0, x1, ...)
panel.polygon.zoo(x, ...)

```

**Arguments**

<code>x, x0, x1</code>	time series object of class "zoo", "ts" or "its". For <code>panel.plot.default</code> it should be a numeric vector.
<code>y</code>	numeric vector or matrix.
<code>subscripts, groups, panel</code>	arguments for panel functions, see description of <code>panel</code> argument in <code>xyplot</code> .
<code>data</code>	currently not used.
<code>screens</code>	factor (or coerced to factor) whose levels specify which graph each series is to be plotted in. <code>screens = c(1, 2, 1)</code> would plot series 1, 2 and 3 in graphs 1, 2 and 1.
<code>default.scales</code>	scales specification. The default is set so that all series have the "same" X axis but "free" Y axis. See <code>xyplot</code> in the <b>lattice</b> package for more information on scales. For users, it is recommended to set the <code>scales</code> argument instead of <code>default.scales</code> .
<code>layout</code>	numeric vector of length 2 specifying number of columns and rows in the plot, see <code>xyplot</code> for more details. The default is to fill columns with up to 5 rows.
<code>xlab</code>	character string used as the X axis label.
<code>ylab</code>	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.
<code>lty, lwd, pch, type, col</code>	graphical arguments passed to <code>xyplot</code> . These arguments can also be vectors or (named) lists, see details for more information.
<code>strip</code>	logical, character or function specifying headings used for panels. If character, should be a vector the same length as the number of panels. If TRUE column names are used for headers. If FALSE, no headings are produced. See <code>xyplot</code> for the case in which <code>strip</code> is a function.
<code>...</code>	additional arguments passed to <code>xyplot</code> .

**Details**

`xyplot.zoo` plots a "zoo", "ts" or "its" object using `xyplot` 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.

`plot.panel.default` is the default panel function. `plot.panel.custom` facilitates the development of custom panels. Usually it has one argument "panel" which specifies the custom panel. That panel typically calls `plot.panel.default`. The panel function may use the `panel.number()` function to find out which panel is currently being executed. See the examples.

**Value**

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

**See Also**

[zoo](#), [plot.ts](#), [barplot](#), [plot.zoo](#)

**Examples**

```

library("lattice")
library("grid")

# 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)))

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

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

# Number 1. Using trellis.focus.
print(xyplot(z))
trellis.focus("panel", 1, 2, highlight = FALSE)
z.mean <- rollmean(z, 3)
# uncomment next line and remove line after that when
# lattice makes panel.lines generic
# print(panel.lines(time(z.mean), z.mean[,2], col = "blue"))
print(panel.lines.zoo(z.mean[,2], col = "blue"))
print(panel.grid(h = 10, v = 10, col = "grey", lty = 3))
trellis.unfocus()

# Number 2. Using a custom panel routine.
# This example relies on R version 2.40 or higher.
p <- function(x, y, groups = NULL, ...) {
  panel.xyplot(x, y, groups = groups, ...)
  if (panel.number() == 2) {
    panel.lines.zoo(rollmean(zoo(y, x), 3), col = "blue")
    panel.grid(h = 10, v = 10, col = "grey", lty = 3)
  }
}
print(xyplot(z, panel = panel.plot.custom(panel = p)))

# 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.plot.default", trellis.panelArgs())
trellis.unfocus()

# same but make first (i.e. bottom) panel twice as large as others
print(xyplot(z), heights = list(c(2,1,1), units = "null"))
# add a grid - this method does not confine grid to frames

```

```

# To do that see prior example.
panel.grid()

# Plot all in one panel.
print(xyplot(z, screens = 1))

# 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 (upper) panel 3x the height of the 1st (lower) panel
# Also make the strip background orange.
p <- xyplot(z, screens = c(1,1,2), type = c("p", "l", "o"), col = 1:3,
  par.setting = 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")
z <- zoo(seq(length(Days))^2, Days)
Months <- Days[format(Days, "%d") == "01"]
Weeks <- Days[format(Days, "%w") == "0"]
xyplot(z, 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
my.axis <- function(side, ...) if (side != "top") axis.default(side, ...)
my.panel <- function(...) {
  panel.axis(outside = TRUE, lab = FALSE)
  panel.plot.default(...)
}
set.seed(1)
z <- zoo(cbind(a = 1:5, b = 11:15, c = 21:25) + rnorm(5))
xyplot(z, between = list(x = 1.2, y = 1), par.settings = list(panel = "off"),
  axis = my.axis, panel = my.panel)

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

# same but more complex
xyplot(z, strip = FALSE, strip.left = strip.custom(factor.levels = colnames(z)))

## Not run:
# playwith (>= 0.8-55)
library("playwith")

```

```

z3 <- zoo(cbind(a = rnorm(100), b = rnorm(100) + 1), as.Date(1:100))
playwith(xyplot(z3), time.mode = TRUE)

# after running this click on Identify Points and then click on
# points to identify in graph; right click to finish
labs <- paste(z3, index(z3), sep = "@")
playwith(xyplot(z3, type = "o"), labels = labs, label.args = list(cex = 0.7))

# for playwith identify tool this returns indexes into times of clicked points
ids <- do.call(rbind, playDevCur()$ids)$which
z3[ids,]
## 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 coercion from "yearmon" to "Date" (see below), "POSIXct", "POSIXlt", "numeric", "character". In the case of `as.yearmon.character` the format argument is the same as for "Date". One can specify a year and month with no day.

`as.Date.yearmon` and `as.yearmon.yearqtr` both have 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.

### Value

Returns its argument converted to class yearmon.

**See Also**

[yearqtr](#), [zoo](#), [zooreg](#), [ts](#)

**Examples**

```
x <- 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)

# 3rd Friday in last month of the quarter of Date x
as.Date(as.yearmon(as.yearqtr(x)) + 2/12) + 14

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(textConnection(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 '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. "%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.
...	other arguments. Currently not used.

**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).

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".

**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](#), [codestrptime](#).

## Examples

```
x <- 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)

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)
## S3 method for class 'zoo':
print(x, style = , quote = FALSE, ...)
```

## Arguments

<code>x</code>	a numeric vector, matrix or a factor.
<code>order.by</code>	an index vector with unique entries by which the observations in <code>x</code> are ordered. See the details for support of non-unique indexes.
<code>frequency</code>	numeric indicating frequency of <code>order.by</code> . If specified, it is checked whether <code>order.by</code> and <code>frequency</code> comply. If so, a regular "zoo" series is returned, i.e., an object of class <code>c("zooreg", "zoo")</code> . See below and <code>zooreg</code> for more details.
<code>style</code>	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).
<code>quote</code>	logical. Should characters be quoted?
<code>...</code>	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`. 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`), `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.

To "pretty" 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](#).

Six methods are available for NA handling in the data of "zoo" objects: `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.locf` which replaces NAs by the last previous non-NA, `na.omit` which returns a "zoo" object with incomplete observations removed, `na.spline` which uses linear interpolation to fill in NA values and `na.trim` which trims runs of NAs off the beginning and end but not in the interior. A 7th NA routine can be found in the `stinepack` package where `na.stinterp` which 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.

## 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**

```

## 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
## 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))

## even though time index must be unique zoo (and read.zoo)
## will both allow creation of such illegal objects with
## a warning (rather than an 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, force, mean)

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

```

```

# fix it up via interpolation of duplicate times
time(z6) <- na.approx(iffelse(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
## 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)

```

## Arguments

<code>data</code>	a numeric vector, matrix or a factor.
<code>start</code>	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.
<code>end</code>	the time of the last observation, specified in the same way as <code>start</code> .
<code>frequency</code>	the number of observations per unit of time.
<code>deltat</code>	the fraction of the sampling period between successive observations; e.g., 1/12 for monthly data. Only one of <code>frequency</code> or <code>deltat</code> should be provided.
<code>ts.eps</code>	time series comparison tolerance. Frequencies are considered equal if their absolute difference is less than <code>ts.eps</code> .
<code>order.by</code>	a vector by which the observations in <code>x</code> are ordered. If this is specified the arguments <code>start</code> and <code>end</code> are ignored and <code>zoo(data, order.by, frequency)</code> is called. See <code>zoo</code> for more information.

## 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` 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 coercable 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.

## 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 = Sys.Date())
## 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, 4, 5, 6)
zz <- zoo(x, x)
zr <- as.zooreg(zz)
lag(zz)
lag(zr)
diff(zz)
diff(zr)

## 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", "1960-05-29")))
# 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

- \*Topic **array**
  - rollapply, 27
- \*Topic **chron**
  - as.Date.numeric, 5
- \*Topic **iteration**
  - rollapply, 27
- \*Topic **manip**
  - MATCH, 1
  - ORDER, 2
- \*Topic **ts**
  - aggregate.zoo, 3
  - as.zoo, 7
  - coredata, 8
  - frequency<-, 9
  - index, 10
  - is.regular, 11
  - lag.zoo, 12
  - make.par.list, 14
  - merge.zoo, 15
  - na.approx, 17
  - na.locf, 18
  - na.trim, 19
  - plot.zoo, 20
  - read.zoo, 24
  - rollapply, 27
  - rollmean, 29
  - window.zoo, 30
  - xyplot.zoo, 31
  - yearmon, 35
  - yearqtr, 37
  - zoo, 38
  - zooreg, 43
- \*Topic **utilities**
  - as.Date.numeric, 5
  - .yearmon (yearmon), 35
  - .yearqtr (yearqtr), 37
  - [.yearmon (yearmon), 35
  - [.yearqtr (yearqtr), 37
  - [.zoo (zoo), 38
  - \$.zoo (zoo), 38
  - \$<-.zoo (zoo), 38
  - aggregate.zoo, 3, 25, 40
  - approx, 17
  - as.character.yearmon (yearmon), 35
  - as.character.yearqtr (yearqtr), 37
  - as.data.frame, 7
  - as.data.frame.yearmon (yearmon), 35
  - as.data.frame.yearqtr (yearqtr), 37
  - as.Date, 6
  - as.Date.numeric, 5
  - as.Date.ts (as.Date.numeric), 5
  - as.Date.yearmon (yearmon), 35
  - as.Date.yearqtr (yearqtr), 37
  - as.list, 7
  - as.list.ts (as.zoo), 7
  - as.list.zoo (as.zoo), 7
  - as.matrix, 7
  - as.matrix.zoo (as.zoo), 7
  - as.numeric.yearmon (yearmon), 35
  - as.numeric.yearqtr (yearqtr), 37
  - as.POSIXct, 25
  - as.POSIXct.yearmon (yearmon), 35
  - as.POSIXct.yearqtr (yearqtr), 37
  - as.POSIXlt.yearmon (yearmon), 35
  - as.POSIXlt.yearqtr (yearqtr), 37
  - as.ts, 7
  - as.ts.zoo (as.zoo), 7
  - as.ts.zooreg (zooreg), 43
  - as.vector, 7
  - as.vector.zoo (as.zoo), 7
  - as.yearmon (yearmon), 35
  - as.yearqtr (yearqtr), 37
  - as.zoo, 7, 40
  - as.zoo.factor (zoo), 38
  - as.zoo.zooreg (zooreg), 43

- as.zooreg (*zooreg*), 43
- barplot, 21, 22, 33
- barplot.zoo (*plot.zoo*), 20
- c.yearmon (*yearmon*), 35
- c.yearqtr (*yearqtr*), 37
- c.zoo (*merge.zoo*), 15
- cbind.zoo (*merge.zoo*), 15
- coredata, 8, 40
- coredata<- (*coredata*), 8
- cummax.zoo (*ZOO*), 38
- cummin.zoo (*ZOO*), 38
- cumprod.zoo (*ZOO*), 38
- cumsum.zoo (*ZOO*), 38
- cycle, 44
- cycle.yearmon (*yearmon*), 35
- cycle.yearqtr (*yearqtr*), 37
- cycle.zoo (*zooreg*), 43
- cycle.zooreg (*zooreg*), 43
- Date, 6, 44
- deltat, 44
- deltat.zoo (*zooreg*), 43
- deltat.zooreg (*zooreg*), 43
- diff, 13
- diff.zoo (*lag.zoo*), 12
- end.zoo (*index*), 10
- format.yearmon (*yearmon*), 35
- format.yearqtr (*yearqtr*), 37
- frequency, 44
- frequency.zoo (*zooreg*), 43
- frequency.zooreg (*zooreg*), 43
- frequency<-, 9
- head.ts (*ZOO*), 38
- head.zoo (*ZOO*), 38
- index, 9, 10, 40, 41
- index2char (*ZOO*), 38
- index<- (*index*), 10
- index<- .zooreg (*zooreg*), 43
- irts, 7
- is.regular, 11, 40, 44
- is.zoo (*ZOO*), 38
- its, 7
- lag, 13
- lag.zoo, 12, 40
- lag.zooreg (*zooreg*), 43
- layout, 21
- lines, 21
- lines.zoo (*plot.zoo*), 20
- make.par.list, 14
- MATCH, 1, 39
- match, 2, 39
- MATCH.yearmon (*yearmon*), 35
- MATCH.yearqtr (*yearqtr*), 37
- mean, 25
- merge.zoo, 15, 40, 41
- na.approx, 17, 19, 40
- na.contiguous, 17, 19, 40
- na.contiguous (*ZOO*), 38
- na.locf, 18, 19, 40
- na.omit, 17, 19, 40
- na.spline, 19, 40
- na.spline (*na.approx*), 17
- na.stinterp, 40
- na.trim, 17, 19, 40
- Ops, 40, 44
- Ops.yearmon (*yearmon*), 35
- Ops.yearqtr (*yearqtr*), 37
- Ops.zoo (*ZOO*), 38
- ORDER, 2, 39
- order, 2, 3, 39
- panel.arrows.its (*xyplot.zoo*), 31
- panel.arrows.ts (*xyplot.zoo*), 31
- panel.arrows.zoo (*xyplot.zoo*), 31
- panel.lines.its (*xyplot.zoo*), 31
- panel.lines.ts (*xyplot.zoo*), 31
- panel.lines.zoo (*xyplot.zoo*), 31
- panel.plot.custom (*xyplot.zoo*), 31
- panel.plot.default (*xyplot.zoo*), 31
- panel.points.its (*xyplot.zoo*), 31
- panel.points.ts (*xyplot.zoo*), 31
- panel.points.zoo (*xyplot.zoo*), 31
- panel.polygon.its (*xyplot.zoo*), 31
- panel.polygon.ts (*xyplot.zoo*), 31
- panel.polygon.zoo (*xyplot.zoo*), 31
- panel.rect.its (*xyplot.zoo*), 31
- panel.rect.ts (*xyplot.zoo*), 31
- panel.rect.zoo (*xyplot.zoo*), 31

- panel.segments.its (*xyplot.zoo*),  
31
- panel.segments.ts (*xyplot.zoo*), 31
- panel.segments.zoo (*xyplot.zoo*),  
31
- panel.text.its (*xyplot.zoo*), 31
- panel.text.ts (*xyplot.zoo*), 31
- panel.text.zoo (*xyplot.zoo*), 31
- par, 21
- plot.ts, 22, 33
- plot.zoo, 20, 33, 40, 41
- points.zoo (*plot.zoo*), 20
- POSIXct, 44
- print.yearmon (*yearmon*), 35
- print.yearqtr (*yearqtr*), 37
- print.zoo (*ZOO*), 38
  
- range.zoo (*ZOO*), 38
- rbind.zoo (*merge.zoo*), 15
- read.table, 25
- read.zoo, 24
- rollapply, 27, 29, 40
- rollmax, 28, 40
- rollmax (*rollmean*), 29
- rollmean, 28, 29, 40
- rollmedian, 28, 40
- rollmedian (*rollmean*), 29
- runmed, 29
  
- scale.zoo (*ZOO*), 38
- spline, 17
- start.zoo (*index*), 10
- stinterp, 17, 19
- str.zoo (*ZOO*), 38
- strptime, 38
- summary.yearmon (*yearmon*), 35
- summary.yearqtr (*yearqtr*), 37
- summary.zoo (*ZOO*), 38
  
- t.zoo (*ZOO*), 38
- tail.ts (*ZOO*), 38
- tail.zoo (*ZOO*), 38
- text, 21
- time, 10
- time.zoo (*index*), 10
- time<- (*index*), 10
- time<- .zooreg (*zooreg*), 43
- ts, 7, 36, 38, 40, 43, 44
  
- value (*index*), 10
  
- value<- (*index*), 10
  
- window.zoo, 30, 40
- window<- .zoo (*window.zoo*), 30
- with.zoo (*ZOO*), 38
- write.table, 25, 26
- write.zoo (*read.zoo*), 24
  
- xyplot, 32
- xyplot.its (*xyplot.zoo*), 31
- xyplot.ts (*xyplot.zoo*), 31
- xyplot.zoo, 22, 31
  
- yearmon, 35, 38, 44
- yearqtr, 36, 37, 44
  
- zoo, 3, 7, 8, 10, 11, 13, 16–19, 22, 26, 29, 30,  
33, 36, 38, 38, 44
- zooreg, 7, 9, 11, 36, 38–41, 43, 44