zoo: An S3 Class and Methods for Indexed Totally Ordered Observations

Achim Zeileis
Wirtschaftsuniversität Wien

Gabor Grothendieck

Abstract

zoo is an R package providing an S3 class with methods for totally ordered observations, such as irregular time series. Its key design goals are independence of a particular index/time/date class and consistency (to the extent possible) with base R and the `"ts"` class for regular time series. This paper describes how these are achieved within **zoo** and provides several illustrations of the available methods.

Keywords: totally ordered observations, irregular time series, S3, R.

1. Introduction

The R system for statistical computing (R Development Core Team 2004, `http://www.R-project.org/`) ships with a a class for regularly spaced time series, `"ts"` in package **stats**, but has no native class for irregularly spaced time series. With the increased interest in computational finance with R over the last years several implementations of classes for irregular time series emerged which are aimed particularly at finance applications. These include the S3 classes `"timeSeries"` in package **fBasics** from the **Rmetrics** bundle (Wuertz, FIXME) and `"irts"` in package **tseries** (Trapletti and Hornik, FIXME) and the S4 class `"its"` in package **its** (Heywood, FIXME). With these packages available, why would anybody want yet another package providing infrastructure for irregular time series? The above mentioned implementations have in common that they are restricted to a particular class for the time scale: the former implementation comes with its own time class `"timeDate"` whereas the latter two use the `"POSIXct"` class available in base R. And this was the starting point for the **zoo** project: the first author of the present paper needed more general support for ordered observations, independent of a particular index class, for the package **strucchange** (Zeileis, Leisch, Hornik, and Kleiber 2002). Hence the package was called **zoo** which stands for Z's ordered observations. Since the first release, a major part of the additions to **zoo** were provided by the second author of this paper, so that the name of the package does not really reflect the authorship anymore. Nevertheless, independence of a particular index class remained one the most important design goal. While the package evolved to its current status, a second key design goal became more and more clear: to provide methods to standard generic functions for the `"zoo"` class that are similar to those for the `"ts"` class (and base R in general) such that the usage of **zoo** is rather intuitive because no new set of commands has to be learned.

This paper...

2. The class `"zoo"` and its methods

2.1. Creation of `"zoo"` objects

The simple idea for the creation of `"zoo"` objects is to have some vector or matrix of observations `x` which are totally ordered by some index vector. In time series applications this index is measure of time but every other numeric, character or even more abstract vector that provides a total ordering of the observations is also suitable. Objects of class "zoo" are created by the function

```
zoo(x, order.by)
```

where x is the vector or matrix of observations and order.by is the index by which the observations should be ordered. It has to be of the same length as NROW(x), i.e., either the same length as x for vectors or the same number of rows for matrices. The "zoo" object created is essentially the vector/matrix as before but has an additional "index" attribute in which the index is stored. Both the value x and the index can, in principle, be of arbitrary classes. However, most of the following methods (plotting, aggregating, mathematical operations) for "zoo" objects are typically only useful for numeric values x. In contrast, special effort in the design was put into independence from a particular class for the index vector. In **zoo** it is assumed that combination c(), querying the length(), value matching match(), subsetting [,, and, of course, ordering order() work when applied to the index. This is the case, e.g., for standard numeric and character vectors and for vectors of classes "Date", "POSIXct" or "times" from package **chron**, but not for the class "dateTime" in **fBasics**. In the latter case, the solution is to provide methods for the above mentioned functions so that indexing "zoo" objects with "dateTime" vectors works. To achieve this independence of the index class the non-generic functions order and match are made S3 generics in **zoo** with their base definition as the default method.

To illustrate the usage of zoo, we first load the package and set the random seed to make the examples in this paper exactly reproducible.

```
R> library(zoo)
R> set.seed(1071)
```

Then, we create two vectors z1 and z2 with "POSIXct" indexes, one with random values

```
R> z1.index <- as.POSIXct(paste("2004-", rep(1:2, 5), "-", sample(1:28,
+     10), sep = ""))
R> z1.value <- rnorm(10)
R> z1 <- zoo(z1.value, z1.index)
```

and one with a sinus wave

```
R> z2.index <- as.POSIXct(paste("2004-", rep(1:2, 5), "-", sample(1:28,
+     10), sep = ""))
R> z2.value <- sin(2 * 1:10/pi)
R> z2 <- zoo(z2.value, z2.index)
```

Furthermore, we create a matrix Z with random values and a "Date" index

```
R> Z.index <- structure(sample(12450:12500, 10), class = "Date")
R> Z.value <- matrix(rnorm(30), ncol = 3)
R> colnames(Z.value) <- c("Aa", "Bb", "Cc")
R> Z <- zoo(Z.value, Z.index)
```

Note, that in the above examples the creation of indexes might seem a bit awkward at first sight, but this is only an artefact of the need for random generation of random dates for this illustration. In "real world" applications, the indexes are typically part of the raw data set read into R. See Section 3 for such examples.

Methods to several standard generic functions are available for "zoo" objects, such as print, summary, str, head, tail and [(subsetting), a few of which are illustrated in the following.

There are three printing code styles for "zoo" objects: vectors are default printed in "horizontal" style

```
R> z1

 2004-01-05  2004-01-14  2004-01-19  2004-01-25  2004-01-27  2004-02-07
 0.74675994  0.02107873 -0.29823529  0.68625772  1.94078850  1.27384445
 2004-02-12  2004-02-16  2004-02-20  2004-02-24
 0.22170438 -2.07607585 -1.78439244 -0.19533304
```

```
R> z1[3:7]

2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12
-0.2982353  0.6862577  1.9407885  1.2738445  0.2217044
```

and matrices in "vertical" style

```
R> Z

                   Aa          Bb          Cc
2004-02-02  1.25543390  0.68157316 -0.63292049
2004-02-08 -1.49458326  1.32341223 -1.49442269
2004-02-09 -1.87462247 -0.87329289  0.62733971
2004-02-21 -0.14538608  0.45234903 -0.14597401
2004-02-22  0.22542418  0.53838938  0.23136133
2004-02-29  1.20695518  0.31814222 -0.01129202
2004-03-05 -1.20861025  1.42379785 -0.81614483
2004-03-10 -0.11039563  1.34774254  0.95522468
2004-03-14  0.84202385 -2.73842019  0.23150695
2004-03-20 -0.19019104  0.12308872 -1.51862157
```

```
R> Z[1:3, 2:3]

                   Bb         Cc
2004-02-02  0.6815732 -0.6329205
2004-02-08  1.3234122 -1.4944227
2004-02-09 -0.8732929  0.6273397
```

Additionally, there is a "plain" style which simply first prints the value and then the index.

Summaries and most other methods for "zoo" objects are carried out column wise, reflecting the rectangular structure indexed by rows. In addition, a summary of the index is provided.

```
R> summary(z1)

     Index                              z1
 Min.   :2004-01-05 00:00:00   Min.   :-2.07608
 1st Qu.:2004-01-20 12:00:00   1st Qu.:-0.27251
 Median :2004-02-01 12:00:00   Median : 0.12139
 Mean   :2004-02-01 09:36:00   Mean   : 0.05364
 3rd Qu.:2004-02-15 00:00:00   3rd Qu.: 0.73163
 Max.   :2004-02-24 00:00:00   Max.   : 1.94079
```

```
R> summary(Z)

     Index                  Aa                 Bb                 Cc
 Min.   :2004-02-02   Min.   :-1.8746   Min.   :-2.7384   Min.   :-1.51862
 1st Qu.:2004-02-12   1st Qu.:-0.9540   1st Qu.: 0.1719   1st Qu.:-0.77034
 Median :2004-02-25   Median :-0.1279   Median : 0.4954   Median :-0.07863
 Mean   :2004-02-25   Mean   :-0.1494   Mean   : 0.2597   Mean   :-0.25739
 3rd Qu.:2004-03-08   3rd Qu.: 0.6879   3rd Qu.: 1.1630   3rd Qu.: 0.23147
 Max.   :2004-03-20   Max.   : 1.2554   Max.   : 1.4238   Max.   : 0.95522
```

2.2. Plotting

The `plot` method for `"zoo"` objects, in particular for multivariate `"zoo"` series, is based on the corresponding method for multivariate regular time series (class `"mts"` which inherits from `"ts"`). By default it creates a panel for each series

```
R> plot(Z)
```

but can also display all series in a single panel

```
R> plot(Z, plot.type = "single", col = 2:4)
```

where in both cases additional graphical parameters like color `col`, plotting character `pch` and line type `lty` can be expanded to the number of series. But the `plot` method for `"zoo"` objects offers some more flexibility in specification of graphical parameters as in

```
R> plot(Z, type = "b", lty = 1:3, pch = list(Aa = 1:5, Bb = 2, Cc = 4),
+     col = list(Bb = 2, 4))
```

The argument `lty` behaves as before and sets every series in another line type. The `pch` argument is a named list that assigns to each series a different vector of plotting characters each of which is expanded to the number of observations. Such a list does not necessarily have to include the names of all series, but can also specify a subset. For the remaining series the default parameter is then used which can again be changed: e.g., in the above example series `"Bb"` is plotted in red and all remaining series in blue. The results of the multiple panel plots are depicted in Figure 2 and the single panel plot in 1.

2.3. Merging and binding

As for many rectangular data formats in R, there are both methods for combining the rows and columns of `"zoo"` objects respectively. For the `rbind` method the number of columns of the combined objects has to be identical and the indexes may not overlap.

```
R> rbind(z1[5:10], z1[2:3])

 2004-01-14  2004-01-19  2004-01-27  2004-02-07  2004-02-12  2004-02-16
 0.02107873 -0.29823529  1.94078850  1.27384445  0.22170438 -2.07607585
 2004-02-20  2004-02-24
-1.78439244 -0.19533304
```

The `cbind` method by default combines the columns by the union of the indexes and fills the created gaps by `NA`s.

```
R> cbind(z1, z2)

                  ..1         ..2
2004-01-03         NA  0.94306673
2004-01-05  0.74675994 -0.04149429
```

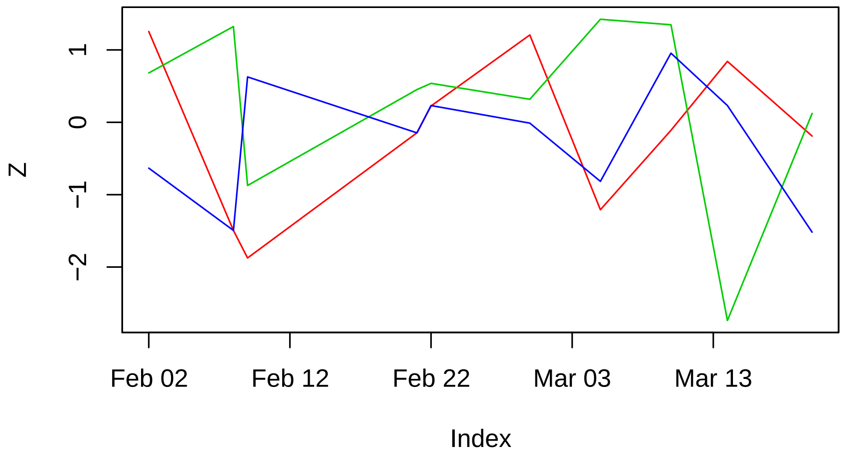


Figure 1: Example of a single panel plot

```
2004-01-14  0.02107873          NA
2004-01-17          NA  0.59448077
2004-01-19 -0.29823529 -0.52575918
2004-01-24          NA -0.96739776
2004-01-25  0.68625772          NA
2004-01-27  1.94078850          NA
2004-02-07  1.27384445          NA
2004-02-08          NA  0.95605566
2004-02-12  0.22170438 -0.62733473
2004-02-13          NA -0.92845336
2004-02-16 -2.07607585          NA
2004-02-20 -1.78439244          NA
2004-02-24 -0.19533304          NA
2004-02-25          NA  0.56060280
2004-02-26          NA  0.08291711
```

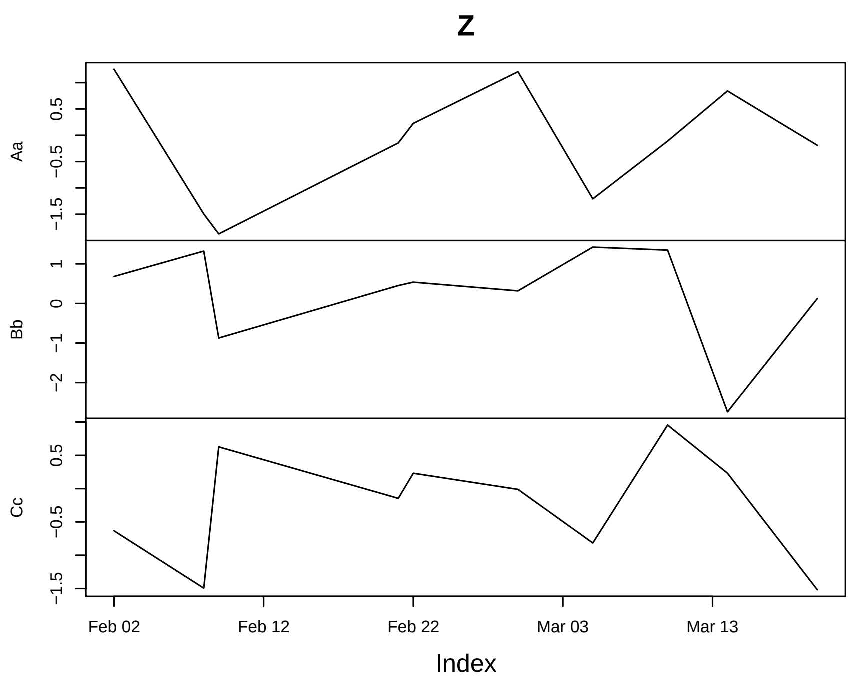
In fact, the `cbind` method is synonymous to the `merge` method which also allows for combining the columns by the intersection of the indexes using the argument `all = FALSE`.

```
R> merge(z1, z2, all = FALSE)

                    z1          z2
2004-01-05  0.74675994 -0.04149429
2004-01-19 -0.29823529 -0.52575918
2004-02-12  0.22170438 -0.62733473
```

Additionally, the filling pattern can be changed and the naming of the columns can be modified. In the case of merging of objects with different index classes, R gives a warning and tries to coerce the indexes, but this is generally rather difficult

Another function which performs operations along a subset of indexes is `aggregate`, which is therefore discussed in this section although it does not combine several objects. Using the `aggre-`



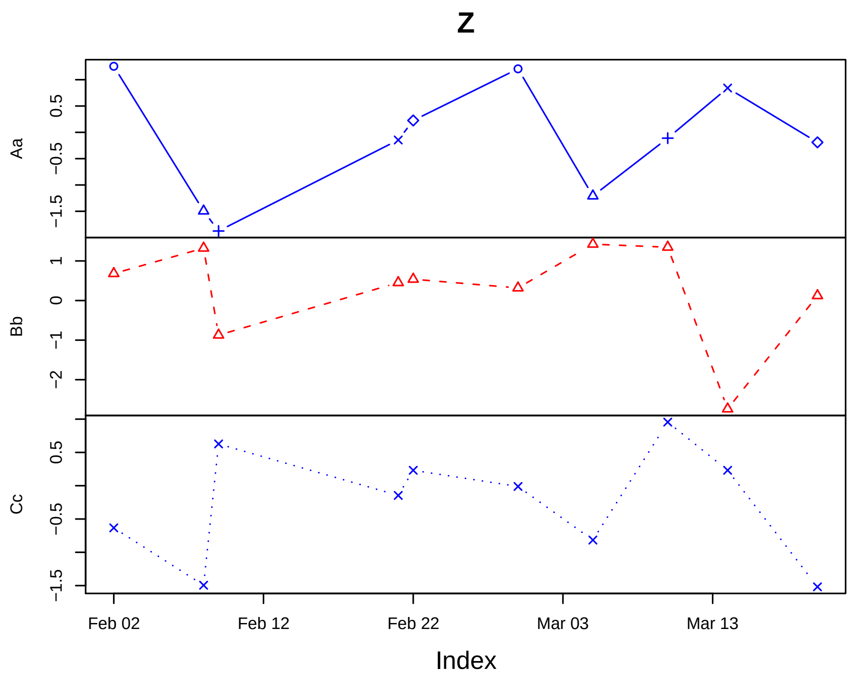


Figure 2: Examples of multiple panel plots

`gate` method, `"zoo"` objects are split into subsets along a coarser index grid, summary statistics are computed for each and then the reduced object is returned. In the following example, first a function is set up which returns for a given `"Date"` value the corresponding first of the month. This function is then used to compute the coarser grid for the `aggregate` call: in the first example the mean of the observations in the month is returned, in the second example the last observation.

```
R> firstofmonth <- function(x) as.Date(sub("..$", "01", format(x)))
R> aggregate(Z, firstofmonth(Z.index), mean)
```

```
                   Aa          Bb          Cc
2004-02-01  0.53820841  0.04508597 -0.12412352
2004-03-01 -1.18080051  0.58156655 -0.45730045
```

```
R> aggregate(Z, firstofmonth(Z.index), tail, 1)
```

```
                  Aa         Bb         Cc
2004-02-01 -0.1901910  0.1230887 -1.5186216
2004-03-01 -1.2086102  1.4237978 -0.8161448
```

2.4. Mathematical operations

To allow for standard mathematical operations among `"zoo"` objects, **zoo** extends group generic functions `Ops`. These perform the operations only for the intersection of the indexes of the objects. Hence, the summation of `z1` and `z2` yields

```
R> z1 + z2
```

```
2004-01-05 2004-01-19 2004-02-12 
 0.7052657 -0.8239945 -0.4056304 
```

Additionally, methods for transposing `t` of `"zoo"` objects—which coerces to a matrix before, see below—and computing cumulative quantities such as `cumsum`, `cumprod`, `cummin`, `cummax` which are all applied column wise.

```
R> cumsum(Z)
```

```
                  Aa        Bb         Cc
2004-02-02  1.2554339 0.6815732 -0.6329205
2004-02-08 -0.2391494 2.0049854 -2.1273432
2004-02-09 -2.1137718 1.1316925 -1.5000035
2004-02-21 -2.2591579 1.5840415 -1.6459775
2004-02-22 -2.0337337 2.1224309 -1.4146162
2004-02-29 -0.8267785 2.4405731 -1.4259082
2004-03-05 -2.0353888 3.8643710 -2.2420530
2004-03-10 -2.1457844 5.2121135 -1.2868283
2004-03-14 -1.3037606 2.4736933 -1.0553214
2004-03-20 -1.4939516 2.5967820 -2.5739429
```

2.5. Extracting and replacing of the value und/or index

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

The value stored in `"zoo"` objects can be extracted by `value` which strips off all `"zoo"`-specific attributes and it can be replaced using `value<-`. Both are new generic functions with methods for `"zoo"` objects as illustrated in the following example.

```
R> value(z1)

 [1]  0.74675994  0.02107873 -0.29823529  0.68625772  1.94078850  1.27384445
 [7]  0.22170438 -2.07607585 -1.78439244 -0.19533304

R> value(z1) <- 1:10
R> z1

2004-01-05 2004-01-14 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12
         1          2          3          4          5          6          7
2004-02-16 2004-02-20 2004-02-24
         8          9         10
```

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<-`. Hence, the following two commands return equivalent results

```
R> index(z2)

 [1] "2004-01-03 CET" "2004-01-05 CET" "2004-01-17 CET" "2004-01-19 CET"
 [5] "2004-01-24 CET" "2004-02-08 CET" "2004-02-12 CET" "2004-02-13 CET"
 [9] "2004-02-25 CET" "2004-02-26 CET"

R> time(z2)

 [1] "2004-01-03 CET" "2004-01-05 CET" "2004-01-17 CET" "2004-01-19 CET"
 [5] "2004-01-24 CET" "2004-02-08 CET" "2004-02-12 CET" "2004-02-13 CET"
 [9] "2004-02-25 CET" "2004-02-26 CET"
```

The index scale of `z2` can be change to that of `z1` by

```
R> z2

 2004-01-03  2004-01-05  2004-01-17  2004-01-19  2004-01-24  2004-02-08
 0.94306673 -0.04149429  0.59448077 -0.52575918 -0.96739776  0.95605566
 2004-02-12  2004-02-13  2004-02-25  2004-02-26
-0.62733473 -0.92845336  0.56060280  0.08291711

R> index(z2) <- index(z1)
R> z2

 2004-01-05  2004-01-14  2004-01-19  2004-01-25  2004-01-27  2004-02-07
 0.94306673 -0.04149429  0.59448077 -0.52575918 -0.96739776  0.95605566
 2004-02-12  2004-02-16  2004-02-20  2004-02-24
-0.62733473 -0.92845336  0.56060280  0.08291711
```

The start and the end of the index/time vector can be queried by `start` and `end`:

```
R> start(z1)

[1] "2004-01-05 CET"

R> end(z1)
```

```
[1] "2004-02-24 CET"
```

To work on both value and index/time, **zoo** provides method a method to `window` and also adds a new generic `window<-` with a method for `"zoo"` objects. In both cases the window is specified by

```
window(x, index, start, end)
```

where `x` is the `"zoo"` object, `index` is a set of indexes to be selected (by default the full index of `x`) and `start` and `end` can be used to restrict the `index` set. Thus, the first command in the following example selects all observations starting from 2004–03–01 whereas the second selects only from the observations with the 5th to 8th index those up to 2004–03–01.

```
R> window(Z, start = as.Date("2004-03-01"))

                   Aa          Bb          Cc
2004-03-05 -1.2086102  1.4237978 -0.8161448
2004-03-10 -0.1103956  1.3477425  0.9552247
2004-03-14  0.8420238 -2.7384202  0.2315069
2004-03-20 -0.1901910  0.1230887 -1.5186216
```

```
R> window(Z, index = index(Z)[5:8], end = as.Date("2004-03-01"))

                   Aa          Bb           Cc
2004-02-22  0.22542418  0.53838938  0.23136133
2004-02-29  1.20695518  0.31814222 -0.01129202
```

The same syntax can be used for the corresponding replacement function.

```
R> window(z1, end = as.POSIXct("2004-02-01")) <- 9:5
R> z1

2004-01-05 2004-01-14 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12
         9          8          7          6          5          6          7
2004-02-16 2004-02-20 2004-02-24
         8          9         10
```

Two methods to standard generic functions in time series applications are `lag` and `diff` which are available with the same arguments as the `"ts"` methods—with the only exception that `diff`.

```
R> lag(z1, k = -1)

2004-01-14 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12 2004-02-16
         9          8          7          6          5          6          7
2004-02-20 2004-02-24
         8          9
```

```
R> diff(z1)

2004-01-14 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12 2004-02-16
        -1         -1         -1         -1          1          1          1
2004-02-20 2004-02-24
         1          1
```

```
R> lu <- get.hist.quote(instrument = "LU", start = "2001-01-01",
+     origin = "1970-01-01")
R> LU <- zoo(value(lu), structure(time(lu) * 86400, class = c("POSIXt",
+     "POSIXct")))
R> LU <- na.omit(LU)
R> LU2 <- zoo(value(lu), structure(time(lu), class = "Date"))
R> LU2 <- na.omit(LU2)
R> plot(diff(log(LU)), col = list(High = 4, 2))
```

2.6. Coercion to and from `"zoo"`

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 data frames (dropping the index/time attribute). See `as.zoo`.

2.7. `NA` handling

Two methods are available for `NA` handling in the data of `"zoo"` objects: `na.omit` which returns a `"zoo"` object with incomplete observations removed and `na.contiguous` which extracts the longest consecutive stretch of non-missing values in a `"zoo"` object. Note, that the latter function is made a generic in `zoo` with the base function being the default.

```
R> library(zoo)
R> x.date <- as.POSIXct(paste("2003-02-", c(1, 3, 7, 9, 14), sep = ""))
R> x <- zoo(rnorm(5), x.date)
R> plot(x)
R> time(x)

[1] "2003-02-01 CET" "2003-02-03 CET" "2003-02-07 CET" "2003-02-09 CET"
[5] "2003-02-14 CET"

R> x[1:3]

2003-02-01 2003-02-03 2003-02-07
 2.0041709 -0.7730781 -0.5471904

R> x.Date <- as.Date(paste("2003-02-", c(1, 3, 7, 9, 14), sep = ""))
R> x <- zoo(rnorm(5), x.Date)
R> plot(x)
R> y.POSIXct <- ISOdatetime(2003, 2, c(1, 3, 7, 9, 14), 0, 0, 0)
R> y <- zoo(rnorm(5), y.POSIXct)
R> plot(y)
R> z <- zoo(rnorm(5), runif(5))
R> plot(z)
R> z <- zoo(1, seq(4)[-2])
R> z0 <- zoo(, 1:4)[, -1]
```

3. Combining zoo with other packages

References

R Development Core Team (2004). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL `http://www.R-project.org/`.

Zeileis A, Leisch F, Hornik K, Kleiber C (2002). "`strucchange`: An R Package for Testing for Structural Change in Linear Regression Models." *Journal of Statistical Software*, **7**(2), 1–38. URL `http://www.jstatsoft.org/v07/i02/`.