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 indexed totally ordered observations, such as irregular time series. Its key design goals are independence of a particular index/time/date class and consistency 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 for `"zoo"` objects which include plotting, merging and binding, several mathematical operations, extracting and replacing data and index, coercion and `NA` handling.

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 2004) and `"irts"` in package **tseries** (Trapletti 2004) and the S4 class `"its"` in package **its** (Heywood 2004). 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"` built on top of the `"POSIXt"` classes available in base R whereas the latter two use `"POSIXct"` directly. 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 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 few additional commands have to be learned. This paper describes how these design goals are implemented in **zoo**. The resulting package provides the `"zoo"` class which offers an extensive (and still growing) set of standard and new methods for working on indexed observations and 'talks' to the classes `"ts"`, `"its"`, `"irts"` and `"timeSeries"`.

The remainder of the paper is organized as follows: Section 2 explains how `"zoo"` objects are created and illustrates how the corresponding methods for plotting, merging and binding, several mathematical operations, extracting and replacing data and index, coercion and `NA` handling can be used. Section 3 outlines how other packages can build on this basic infrastructure. Section 4 gives a few summarizing remarks and an outlook on future developments. Finally, an appendix provides a reference card that gives an overview of the functionality contained in **zoo**.

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 a 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[1] 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. (This constraint is not imposed for zero length vectors.) The "zoo" object created is essentially the vector/matrix as before but has an additional "index" attribute in which the index is stored.[2] Both the observations in the vector/matrix x and the index order.by 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 observations x. 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 last case, the solution is to provide methods for the above mentioned functions so that indexing "zoo" objects with "dateTime" vectors works (see Section 3.3 for an example). To achieve this independence of the index class, new generic functions for ordering (ORDER()) and value matching (MATCH()) are introduced as the corresponding base functions order() and match() are non-generic. The default methods simply call the corresponding base functions, i.e., no new method needs to be introduced for a particular index class if the non-generic functions order() and match() work for this class.

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.

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

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

```
> z1.index <- ISOdatetime(2004, rep(1:2, 5), sample(28, 10), 0,
+     0, 0)
> z1.data <- rnorm(10)
> z1 <- zoo(z1.data, z1.index)
```

and one with a sine wave

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

[1]In principle, more general objects can be indexed, but currently **zoo** does not support this. Development plans are that **zoo** should eventually support indexed factors, data frames and lists.

[2]There is some limited support for indexed factors available in which case the "zoo" object also has an attribute "oclass" with the original class of x. This feature is still under development and might change in future versions.

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

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

In the examples above, the generation of indexes looks a bit awkward due to the fact the indexes need to be randomly generated (and there are no special functions for random indexes because these is rarely needed in practice). In "real world" applications, the indexes are typically part of the raw data set read into R so the code would be even simpler. See Section 3 for such examples.[3]

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 by default printed in "horizontal" style

```
> 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
```

```
> 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

```
> 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
```

```
> 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
```

[3] Note, that in the code above a new `as.Date` method, provided in **zoo**, is used to convert days since 1970-01-01 to class "Date". See the respective help page for more details.

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

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

```
> 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
```

```
> 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. It relies on plot and lines methods being available for the index class which can plot the index against the observations.

By default the plot method creates a panel for each series

```
> plot(Z)
```

but can also display all series in a single panel

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

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

```
> 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 the col argument is set to display the series "Bb" 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.

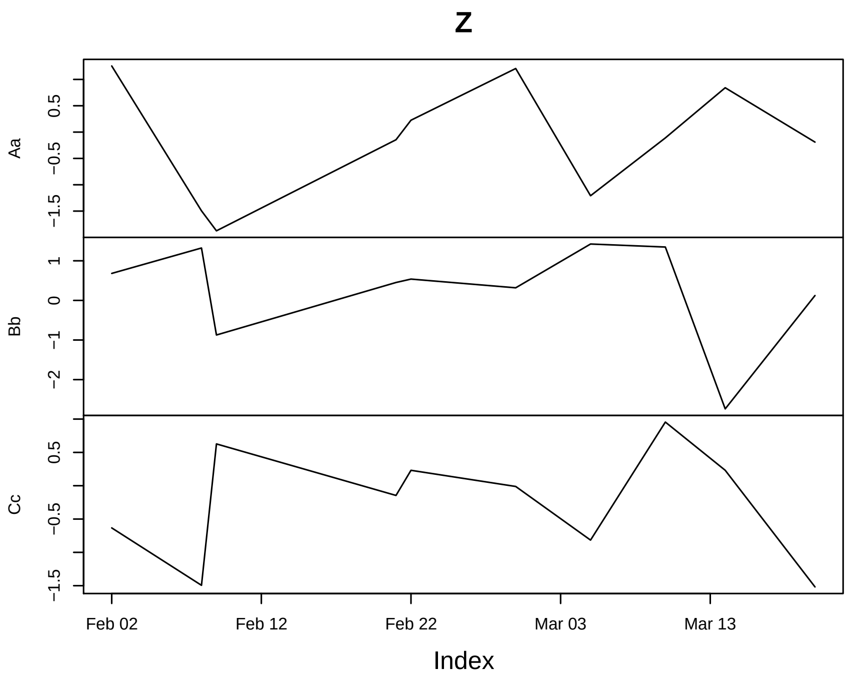
Figure 1: Example of a single panel plot

```
> 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.[4]

```
> cbind(z1, z2)

                z1          z2
2004-01-03          NA  0.94306673
2004-01-05  0.74675994 -0.04149429
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
```

[4]Note, that `cbind` currently is inferior to `merge` regarding the column naming of the resulting object.



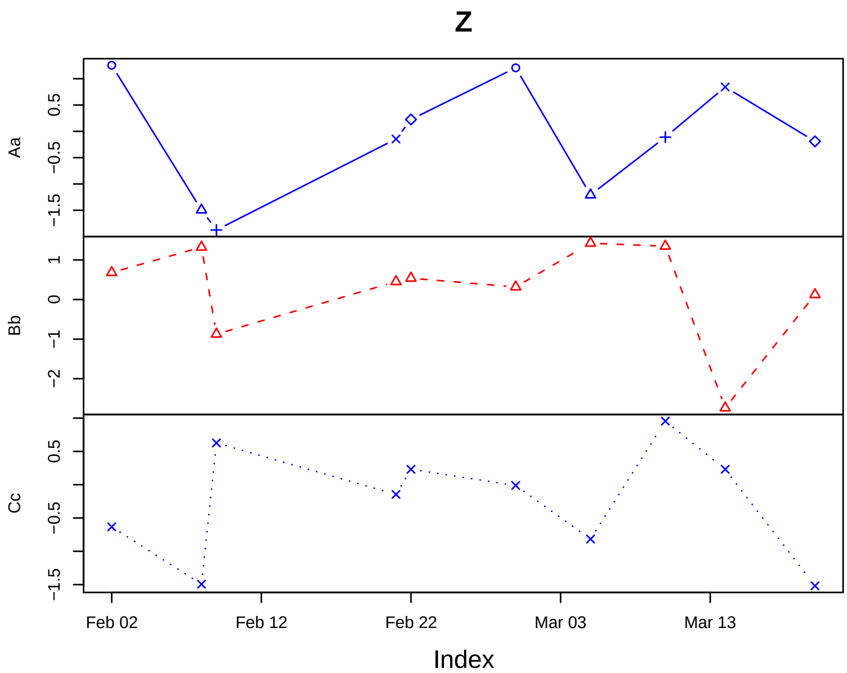


Figure 2: Examples of multiple panel plots

In fact, the `cbind` method is synonymous with the `merge` method except that the latter provides additional arguments which allow for combining the columns by the intersection of the indexes using the argument `all = FALSE`

```
> 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 in `merge`, the naming of the columns can be modified and the return class of the result can be specified. In the case of merging of objects with different index classes, R gives a warning and tries to coerce the indexes. Merging objects with different index classes is generally discouraged—if it is used nevertheless, it is the responsibility of the user to ensure that the result is as intended.

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

```
> firstofmonth <- function(x) as.Date(sub("..$", "01", format(x)))
> 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

> aggregate(Z, firstofmonth(Z.index), head, 1)

                  Aa         Bb         Cc
2004-02-01  1.2554339  0.6815732 -0.6329205
2004-03-01 -1.4945833  1.3234122 -1.4944227
```

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. As an example, the summation and logical comparison with $<$ of `z1` and `z2` yield

```
> z1 + z2

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

> z1 < z2

2004-01-05 2004-01-19 2004-02-12
     FALSE      FALSE      FALSE
```

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.

```
> 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 the data and the index

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

The data stored in `"zoo"` objects can be extracted by `coredata` which strips off all `"zoo"`-specific attributes and it can be replaced using `coredata<-`. Both are new generic functions[5] with methods for `"zoo"` objects as illustrated in the following example.

```
> coredata(z1)

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

> coredata(z1) <- 1:10
> 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 natural, there are also synonymous methods `time` and `time<-`. Hence, the commands `index(z2)` and `time(z2)` return equivalent results.

```
> 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"
```

[5]The `coredata` functionality is similar in spirit to the `core` function in **its** and `value` in **tseries**. However, the focus of those functions is somewhat narrower and we try to provide more general purpose generic functions. See the respective manual page for more details.

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

```
> index(z2) <- index(z1)
> 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`:

```
> start(z1)

[1] "2004-01-05 CET"

> end(z1)

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

To work on both data and index/time, **zoo** provides `window` and `window<-` methods 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.

```
> 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

> 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 first example selects all observations starting from 2004-03-01 whereas the second selects from the from the 5th to 8th observation those up to 2004-03-01.

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

```
> window(z1, end = as.POSIXct("2004-02-01")) <- 9:5
> 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 that are standard in time series applications are `lag` and `diff`. These are available with the same arguments as the `"ts"` methods.[6]

```
> 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

> merge(z1, lag(z1, k = 1))

           z1 lag(z1, k = 1)
2004-01-05  9  8
2004-01-14  8  7
2004-01-19  7  6
2004-01-25  6  5
2004-01-27  5  6
2004-02-07  6  7
2004-02-12  7  8
2004-02-16  8  9
2004-02-20  9 10
2004-02-24 10 NA

> 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
```

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"` using the respective `as.zoo` method. The reverse coercion is available for `"its"` and for `"irts"` (the latter in package **tseries**). Furthermore, `"zoo"` objects can be coerced to vectors, matrices, lists and data frames (the latter dropping the index/time attribute). A simple example is

```
> as.data.frame(Z)

           Aa         Bb          Cc
1   1.2554339  0.6815732 -0.63292049
2  -1.4945833  1.3234122 -1.49442269
3  -1.8746225 -0.8732929  0.62733971
4  -0.1453861  0.4523490 -0.14597401
5   0.2254242  0.5383894  0.23136133
6   1.2069552  0.3181422 -0.01129202
7  -1.2086102  1.4237978 -0.81614483
8  -0.1103956  1.3477425  0.95522468
9   0.8420238 -2.7384202  0.23150695
10 -0.1901910  0.1230887 -1.51862157
```

[6] `diff` also has an additional argument that also allows for geometric and not only allows arithmetic differences. Furthermore, note the sign of the lag in `lag`: by default it is positive and shifts the observations *forward*, to obtain the more standard *backward* shift the lag has to be negative.

2.7. NA handling

Four methods for dealing with NAs (missing observations) in the observations are applicable to "zoo" objects: na.omit, na.contiguous, na.approx and na.locf. na.omit—or its default method to be more precise—returns a "zoo" object with incomplete observations removed. na.contiguous extracts the longest consecutive stretch of non-missing values. This function is currently made generic in **zoo** with a "zoo" method and the **stats** function as the default.[7] Furthermore, new generic functions na.approx and na.locf and corresponding default methods are introduced in **zoo**. The former replaces NAs by linear interpolation (using the function approx) and the name of the latter stands for last observation carried forward. It replaces missing observations by the most recent non-NA prior to it. Leading NAs, which cannot be replaced by precious observations, are removed in both functions by default.

```
> z1[sample(1:10, 3)] <- NA
> z1

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

> na.omit(z1)

2004-01-05 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-16 2004-02-20
         9          7          6          5          6          8          9

> na.contiguous(z1)

2004-01-19 2004-01-25 2004-01-27 2004-02-07
         7          6          5          6

> na.approx(z1)

2004-01-05 2004-01-14 2004-01-19 2004-01-25 2004-01-27 2004-02-07 2004-02-12
  9.000000   7.714286   7.000000   6.000000   5.000000   6.000000   7.111111
2004-02-16 2004-02-20
  8.000000   9.000000

> na.approx(z1, 1:NROW(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
         8          9

> na.locf(z1)

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

[7] na.contiguous will be generic in base R from version 2.1.0 on.

As the above example illustrates, `na.approx` uses by default the underlying time scale for interpolation. This can be changed, e.g., to an equidistant spacing, by setting the second argument of `na.approx`.

3. Combining zoo with other packages

The main purpose of the package **zoo** is to provide basic infrastructure for working with indexed totally ordered observations that can be either employed by users directly or can be a basic ingredient on top of which other packages can build. The latter is illustrated with a few brief examples involving the packages **strucchange**, **tseries**, **fBasics** and **stats** in this section.

3.1. strucchange: Empirical fluctuation processes

The package **strucchange** provides a collection of methods for testing, monitoring and dating structural changes, in particular in linear regression models. Tests for structural change assess whether the parameters of a model remain constant over an ordering with respect to a specified variable, usually time. To adequatly store and visualize empirical fluctuation processes which capture instabilities over this ordering, a data type for indexed ordered observations is required. This was the motivation for starting the **zoo** project.

A simple example for the need of `"zoo"` objects in **strucchange** which is not (easily) be implemented by other irregular time series classes available on CRAN is described in the following. We assess the constancy of the electrical resistance over the apparent juice content of kiwi fruits.[8] The data set `fruitohms` is contained in the **DAAG** package (Maindonald and Braun 2004). The fitted `ocus` object contains the OLS-based CUSUM process for the mean of the electrical resistance (variable `ohms`) indexed by the juice content (variable `juice`).

```
> library(strucchange)
> library(DAAG)

Loading required package: leaps
Loading required package: oz

> data(fruitohms)
> ocus <- gefp(ohms ~ 1, order.by = ~juice, data = fruitohms)
```

This OLS-based CUSUM process can be visualized using the `plot` method for `"gefp"` objects which builds on the `"zoo"` method and yields in this case the plot in Figure 3 showing the process which crosses its 5% critical value and thus signals a significant decrease in the mean electrical resistance over the juice content. for more information on the package **strucchange** and the function `gefp` see Zeileis *et al.* (2002) and Zeileis (2004).

3.2. tseries: Historical financial data

A typical application for irregular time series which became increasingly important over the last years in computational statistics and finance is daily (or higher frequent) financial data. The package **tseries** provides the function `get.hist.quote` for obtaining historical financial data by querying Yahoo! Finance at `http://finance.yahoo.com/`, an online portal quoting data provided by Reuters. The following code queries the quotes of Lucent Technologies starting from 2001-01-01:

```
> library(tseries)
> LU <- get.hist.quote(instrument = "LU", start = "2001-01-01",
+     end = "2004-09-30", origin = "1970-01-01")
```

[8] A different approach would be to test whether the slope of a regression of electrical resistance on juice content changes with increasing juice content, i.e., to test for instabilities in `ohms   juice` instead of `ohms   1`. Both lead to similar results.

```
> plot(ocus)
```

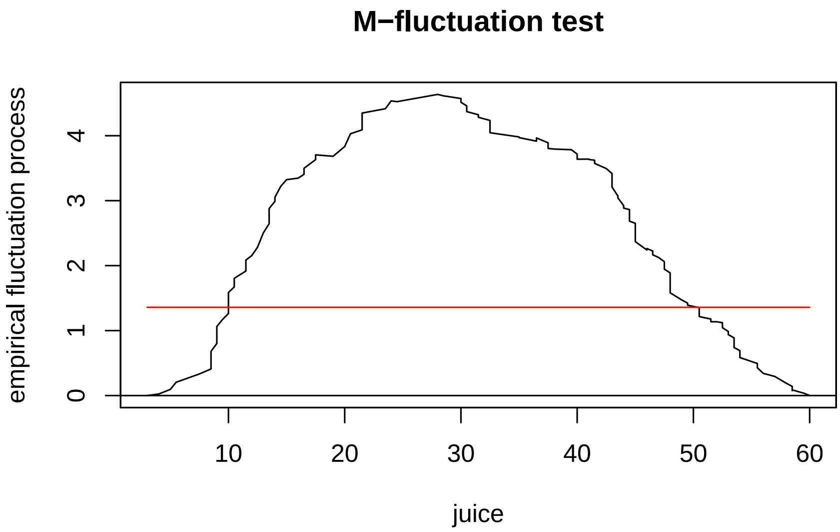


Figure 3: Empirical M-fluctuation process for `fruitohms` data

```
time series starts 2001-01-02
```

In the returned `LU` object the irregular data is stored by extending it in a regular grid and filling the gaps with `NA`s. The time is stored in days starting from an `origin`, in this case specified to be 1970-01-01, the origin used by the `Date` class. This series can be transformed easily into an irregular `"zoo"` series using a `"Date"` index. The log-difference returns for Lucent Technologies is depicted in Figure 4.

```
> LU <- as.zoo(LU)
> index(LU) <- as.Date(index(LU))
> LU <- na.omit(LU)
```

3.3. fBasics: Indexes of class `"timeDate"`

Although the methods in **zoo** work out of the box for many index classes, it might be necessary for some index classes to provide `c`, `length`, `ORDER` and `MATCH` methods such that the methods in **zoo** work properly. An example for such an index class which requires a bit more attention is `"timeDate"` from the **fBasics** package.

But after the necessary methods have been defined

```
> length.timeDate <- function(x) prod(x@Dim)
> ORDER.timeDate <- function(x, ...) order(as.POSIXct(x), ...)
> MATCH.timeDate <- function(x, table, nomatch = NA, ...) match(as.POSIXct(x),
+     as.POSIXct(table), nomatch = NA, ...)
```

the class `"timeDate"` can be used for indexing `"zoo"` objects. The following example illustrates how `z2` can be transformed to use the `"timeDate"` class.

```
> plot(diff(log(LU)))
```

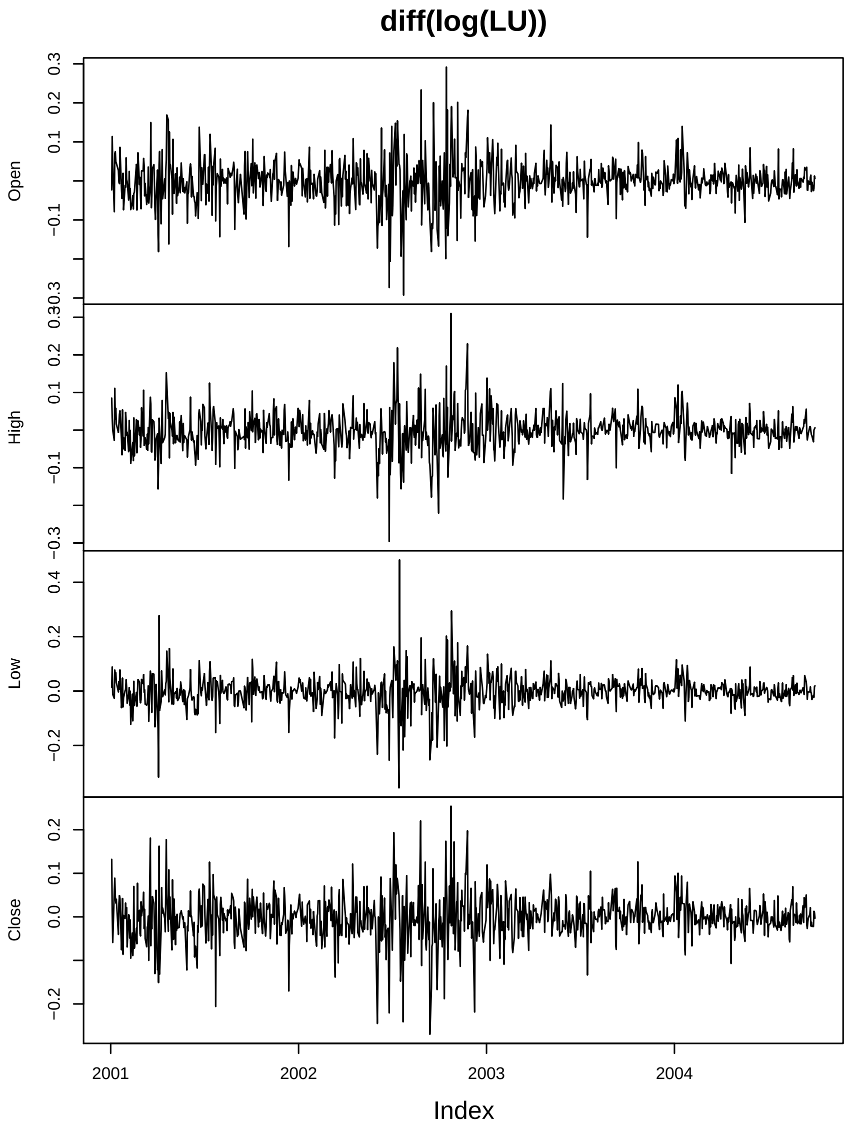


Figure 4: Log-difference returns for Lucent Technologies

```
> library(fBasics)
> z2td <- zoo(coredata(z2), timeDate(index(z2), FinCenter = "GMT"))
> z2td

 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 
```

3.4. stats: (Dynamic) regression modelling

`zoo` provides a facility for extending regression functions such as `lm` to handle time series. One simply encloses the `formula` argument in `I(...)` and ensures that all variables in the formula are of class `"zoo"` or all are of class `"ts"`.

Basic regression functions, like `lm` or `glm`, in which regression relationships are specified via a `formula` only have limited support for time series regression. The reason is that `lm(formula, ...)` calls the generic function `model.frame(formula, ...)` to create a a data frame with the variables required. This dispatches to `model.frame.formula` which does not deal specifically with (various types of) time series data. Therefore, it would be desirable to dispatch to a specialized `model.frame` method depending on the type of the dependent variable. As this is a non-standard dispatch, **zoo** provides the following mechanism: In the call to the regression function, the `formula` is insulated by `I()`, e.g., as in `lm(I(formula), ...)`, leaving `formula` unaltered but returning an object of class `"AsIs"`. Then, `model.frame.AsIs` is called which examines the dependent variable of the `formula` and then dispatches to `model.frame.foo` if this is of class `"foo"`. In **zoo**, the methods `model.frame.zoo` and `model.frame.ts` are provided which are able to create model frames from formulas in which *all* variables are of class `"zoo"` or `"ts"`, respectively. The advantage of `model.frame.zoo` is that it aligns the variables along a common index, it allows the usage of `lag` and `diff` in the model specification and works with the `NA` handling methods described in Section 2.7. Therefore, dynamic linear regression models can be fit easily using the standard `lm` function by just insulating `I(formula)` in the corresponding call[9].

A simple example based on artificial data is given below: the lag of a dependent variable is explained by the first differences of a numeric regressor and an explanatory factor. Note, that the variables have different indexes. First, a linear regression model is fitted, then a quantile regression is carried out for the same equation.

```
> yz <- zoo(1:20)^2
> xz <- zoo(1:18)^2
> fz <- zoo(gl(4, 5))
> lm(I(lag(yz) ~ diff(xz) + fz))

Call:
lm(formula = I(lag(yz) ~ diff(xz) + fz))

Coefficients:
(Intercept)     diff(xz)           fz  
    -68.275       10.092        8.922  

> library(quantreg)
> rq(I(lag(yz) ~ diff(xz) + fz))
```

[9] In addition to `lm` and `glm`, this approach works for many other regression functions including `randomForest` ensembles from **randomForest**, `svm` support vector machines from **e1071**, `lqs` resistant regression from **MASS**, `nnet` neural networks from **nnet**, `rq` quantile regression from **quantreg**, and possibly many others.

```
Call:
rq(formula = I(lag(yz) ~ diff(xz) + fz))

Coefficients:
(Intercept)    diff(xz)          fz
  -58.50000    11.64286   -10.28571

Degrees of freedom: 17 total; 14 residual
```

See the help page of `model.frame.zoo` for more examples and additional information. Furthermore, note that this feature is under development and might subject to changes in future versions.

4. Summary and outlook

The package **zoo** provides an S3 class and methods for indexed totally ordered observations, such as irregular time series. Its key design goals are independence of a particular index class and compatibility with standard generics similar to the behaviour of the corresponding `"ts"` methods. This paper describes how these are implemented in **zoo** and illustrates the usage of the methods for plotting, merging and binding, several mathematical operations, extracting and replacing data and index, coercion and `NA` handling.

An indexed object of class `"zoo"` can be thought of as data plus index where the data are essentially vectors or matrices and the index can be a vector of (in principle) arbitrary class. Therefore, objects of classes `"ts"`, `"its"`, `"irts"` and `"timeSeries"` can easily be transformed into `"zoo"` objects—the reverse transformation is also possible provided that the index fulfills the restrictions of the respective class. Hence, the `"zoo"` class can also be used as the basis for other classes of indexed and objects and more specific functionality can be built on top of it.

Whereas a lot of effort was put into achieving independence of a particular index class, the types of data that can be indexed with `"zoo"` are currently limited to vectors and matrices, typically containing numeric values. Although, there is some limited support available for indexed factors, one important direction for future development of **zoo** is to add better support for other objects that can also naturally be indexed including specifically factors, data frames and lists.

Computational details

The results in this paper were obtained using R 2.0.0 with the packages **zoo** 0.9–1, **strucchange** 1.2–7, **fBasics** 200.10058, **tseries** 0.9–24, **randomForest** 4.5–1 and **DAAG** 0.37. R itself and all packages used are available from CRAN at http://CRAN.R-project.org/.

References

Heywood G (2004). **its**: *Irregular Time Series*. Portfolio & Risk Advisory Group and Commerzbank Securities. R package version 1.0.3.

Maindonald J, Braun WJ (2004). **DAAG**: *Data Analysis And Graphics*. R package version 0.37, URL http://www.stats.uwo.ca/DAAG/.

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/.

Trapletti A (2004). **tseries**: *Time Series Analysis and Computational Finance*. R package version 0.9-23.

Wuertz D (2004). **Rmetrics***: An Environment and Software Collection for Teaching Financial Engineering and Computational Finance*. R package **fBasics**, version 200.10058, URL http://www.itp.phys.ethz.ch/econophysics/R/2.0.

Zeileis A (2004). "Implementing a Class of Structural Change Tests: An Econometric Computing Approach." *Report 7*, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, Research Report Series. URL http://epub.wu-wien.ac.at/.

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/.

A. Reference card

Creation

`zoo(x, order.by)`	creation of a `"zoo"` object from the observations `x` (a vector or a matrix) and an index `order.by` by which the observations are ordered. For computations on arbitrary index classes, methods to the following genric functions are assumed to work: combining `c()`, querying length `length()`, subsetting `[`, ordering `ORDER()` and value matching `MATCH()`.

Standard methods

`plot`	plotting
`lines`	adding a `"zoo"` series to a plot
`print`	printing
`summary`	summarizing (column-wise)
`str`	displaying structure of `"zoo"` objects
`head, tail`	head and tail of `"zoo"` objects

Coercion

`as.zoo`	coercion to `"zoo"` is available for objects of class `"ts"`, `"its"`, `"irts"` (plus a default method).
`as.`*class*`.zoo`	coercion from `"zoo"` to other classes. Currently available for *class* in `"matrix"`, `"vector"`, `"data.frame"`, `"list"`, `"irts"` and `"its"`.
`is.zoo`	querying wether an object is of class `"zoo"`

Merging and binding

`merge`	union, intersection, left join, right join along indexes
`cbind`	column binding along the intersection of the index
`rbind`	row binding (indexes may not overlap)
`aggregate`	compute summary statistics along a coarser grid of indexes

Mathematical operations

`Ops`	group generic functions performed along the intersection of indexes
`t`	transposing (coerces to `"matrix"` before)
`cumsum`	compute (columnwise) cumulative quantities: sums `cumsum()`, products `cumprod()`, maximum `cummax()`, minimum `cummin()`.

Extracting and replacing data and index

`index, time`	extract the index of a series
`index<-, time<-`	replace the index of a series
`coredata, coredata<-`	extract and replace the data associated with a `"zoo"` object
`lag`	lagged observations
`diff`	arithmetic and geometric differences
`start, end`	querying start and end of a series
`window, window<-`	subsetting of `"zoo"` objects using their index

`NA` handling

`na.omit`	omit `NA`s
`na.contiguous`	compute longest sequence of non-`NA` observations
`na.locf`	impute `NA`s by carrying forward the last observation
`na.approx`	impute `NA`s by interpolation